The Development of an Aid for Teaching Sorting Algorithms to Students

Kevin White

BSc in Computer Science

2004

Restrictions

This thesis may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purpose of consultation.

Signed: Kevin White
The Development of an Aid for Teaching Sorting Algorithms to Students

Submitted by Kevin White

Copyright

Attention is drawn to the fact that copyright of this thesis rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the University of Bath (see http://www.bath.ac.uk/ordinances/#intelprop).

The copy of this thesis has been supplied on the condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the thesis and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institute of learning. Except where specially acknowledged, it is the work of the author.

Signed: Kevin White
Abstract

The problem of sorting data in computers is key to the majority of computer programs. Due to this sorting algorithms such as Bubble Sort and Quick Sort are taught to students studying Computer Science as part of their degree. The different sorting algorithms work in different ways with each technique having advantages and disadvantages. Typically sorting algorithms have been taught using static techniques such as slides. This appears not to be the most intuitive of learning styles.

The first algorithm animation system was developed by Baecker in 1981. This was a video titled ‘Sorting Out Sorting’. Numerous different techniques have been developed since to try to help people understand how the different sorting algorithms work.

This project attempts to build an algorithm animation system that will assist students in learning how sorting algorithms work. It will build on previous work in the field and it is hoped that a system will be developed that will be usable by lecturers to assist in their teaching of sorting algorithms.

Acknowledgements

I would like to thank my project supervisor Dr Claire Willis for her time, help and support during the course of this project.

Project Details

Author: Kevin White (ma0kjw@bath.ac.uk)
Supervisor: Dr. Claire Willis (cpw@cs.bath.ac.uk)
Title: The Development of an Aid for Teaching Sorting Algorithms to Students
URL: http://www.bath.ac.uk/~ma0kjw
Timescale: September 2003 – May 2004
Deadline: May 13th 2004
CD

Please find below a CD containing a zip file that contains the following:

- A copy of the dissertation in PDF format.
- The source code for the system.
- The class files for the system.
## Contents

1.0 Introduction 9
  1.1 Why is this project meaningful and relevant? 9
  1.2 Project aims 9
  1.3 How the project will be judged 10
  1.4 What new skills need to be learnt 11
  1.5 Similar work 11
  1.6 What areas may require deeper investigation? 11
  1.7 What resources are needed? 12
  1.8 Which aims are achievable? 12
  1.9 Functionality that will not be considered 12

2.0 Literature Review 12
  2.1 Sorting algorithms 13
  2.2 The animation of algorithms 15
  2.3 Existing sorting algorithm animation systems 17
  2.4 A critical evaluation of existing systems 21
  2.5 Graphical User Interfaces 22
  2.6 A conclusion of the research 23

3.0 Requirements Analysis 25
  3.1 System assessment surveys 25
  3.2 HCI design principles for algorithm animation systems 28
  3.3 Prototypes 31
  3.4 An evaluation of the prototypes 34
  3.5 A revision of the prototypes 34
  3.6 The final prototype 35

4.0 Requirements Specification 37
  4.1 Gathering the requirements 37
  4.2 User requirements 39

5.0 Design 42
  5.1 Design considerations 42
  5.2 System architecture design 43
  5.3 Animation design 43
  5.4 User interface design 44
  5.5 The sorting components 51
  5.6 A summary of the design 53

6.0 Detailed Design and Implementation 53
  6.1 Detailed design considerations 53
  6.2 The graphical user interface 53
  6.3 The setup screens 55
  6.4 The sorting screens 56
  6.5 The sorting algorithms 59
  6.6 The data array 62
Figures

Figure 1.1: The block approach to displaying data
Figure 1.2: The stack approach to displaying data

Figure 2.1: User input in Polka
Figure 2.2: Polka displaying an animation of Heap Sort
Figure 2.3: The control bar in Polka
Figure 2.4: The animation setup panel for an online system
Figure 2.5: Data before sorting has started in an online system
Figure 2.6: Insertion Sort being animated in an online system
Figure 2.7: Sorted data in an online system
Figure 2.8: The dot approach to animating in an online system

Figure 2.9: HCI factor results
Figure 2.10: Algorithm animation technique results
Figure 2.11: Prototype 1
Figure 2.12: Prototype 2
Figure 2.13: Prototype 3
Figure 2.14: Revised prototype
Figure 2.15: Final prototype

Figure 3.1: System breakdown diagram
Figure 3.2: High level system design
Figure 3.3: Block view with sorted data
Figure 3.4: Stack view with sorted data
Figure 3.5: High level screen layout
Figure 3.6: Menu bar container for the system
Figure 3.7: Single animation setup screen for the system
Figure 3.8: Algorithm race setup screen for the system
Figure 3.9: Algorithm analysis screen for the system
Figure 3.10: Help screen for the system
Figure 3.11: About screen for the system
Figure 3.12: Welcome screen for the system
Figure 3.13: Single animation screen for the system
Figure 3.14: Animation race screen for the system

Figure 4.1: NetBeans IDE form editor
Figure 4.2: NetBeans IDE layout designer
Figure 4.3: Adding an event listener to a button
Figure 4.4: Detecting an event
Figure 4.5: The data type combo box
Figure 4.6: Start button event handling
Figure 4.7: Creating random values for the data
Figure 4.8: Reverse sorting data
Figure 4.9: Displaying data in the GUI
Figure 4.10: Setting the background colour of text areas
Figure 4.11: Calculating swaps, comparisons and the time to sort data
Figure 6.12: Bubble Sort code
Figure 6.13: Insertion Sort code
Figure 6.14: Selection Sort code
Figure 6.15: Quick Sort code
Figure 6.16: Heap Sort code

Figure 7.1: Algorithm analysis screen tests
Figure 7.2: Help screen tests
Figure 7.3 – 7.4: Algorithm race screen tests
Figure 7.5 – 7.6: Single animation screen tests
Figure 7.7: HCI factor results
Figure 7.8: Animation effectiveness results
1.0 Introduction

1.1 Why is this project meaningful and relevant?
Algorithms play an important part in helping to decide on the design of a computer application, as the choice of algorithm plays an important role in determining how efficiently a program will run. Professor Donald Knuth, a well-respected global name in the field of computing has stated [1]:

'I believe that virtually every important aspect of programming arises somewhere in the context of searching or sorting.'

From this statement, it can be seen that it is critical for anyone who intends to design software to have at the very least, a sound knowledge of the fundamentals of the algorithms frequently used in computing. Within this set of algorithms there is a very important subset; these are sorting algorithms. There are numerous different algorithms for sorting numerical data into order, some being much more efficient than others, but all being useful in certain circumstances [2].

Sorting algorithms are taught to almost all Computer Science students during their courses. However, it is not always the easiest of concepts for a student to pick up, especially if encountered early on in the course. So far the standard methods of teaching how these algorithms work have consisted predominantly of using diagrams on a blackboard, and more recently slide shows using a laptop and projector. Although effective to a degree, this approach still leaves something to be desired. It can only show a certain level of clarity, because all the slides are static. Therefore, if the lecturer does not describe each algorithm in sufficient detail the student may be left not understanding fully how each algorithm works. Worse, when it comes to comparing the actual efficiency of the algorithms the student may well be confronted with mathematical notations they do not understand.

This brings about the need for this project. An application that shows exactly how all of the most commonly used sorting algorithms work by showing the processes in an animated form would make the sorting technique used far easier for the students to understand. Even if the lecturer did not go into details, the students could see what is happening by looking at the animations on the screen. Further, if this application could compare the algorithms' efficiency by showing them simultaneously processing the same piece of data, then the students should easily be able to grasp the most efficient algorithm to use in a given situation.

Currently there are numerous existing systems designed for this purpose. These invariably have the core functionality of being able to show at least a single sorting algorithm working on unsorted data. Some of the more advanced systems allow the user to specify the type of animation style to be used (e.g. the stack view), the type of data to be used (e.g. reverse sorted), while some even allow multiple sorting algorithms to be run simultaneously in an algorithm race. The vast majority of the existing systems have a Graphical User Interface (GUI) for the user to interact with, and are to be found on the Internet, often in the form of applets. The usability of these systems is extremely variable and this seems to be one of the key reasons why algorithm animation has not become accepted universally as the best technique for teaching sorting algorithms to students.

1.2 Project aims
The aim of this project is to develop an application that will complement lecturers in the teaching of sorting algorithms. The proposed system will need to clearly show how all of the most commonly used sorting algorithms work, and should be able to show them being executed simultaneously in an algorithm race. It should also have the ability to use different types of data when comparing the algorithms. For example, random data, pre-sorted data, reverse sorted data
and partially sorted data. This should ensure that it is perfectly clear which algorithm is most efficient in any given circumstance.

Before such an application can be built an investigation needs to be conducted into a number of areas that will determine the effectiveness of the finished application as a teaching aid. The research will look into the best methods of animating the sorting algorithms in terms of user learning and understanding. For example, is the block approach shown in Figure 1.1 more effective than the stack approach shown in Figure 1.2?

*Figure 1.1: Block approach.*

![Block approach diagram]

*Figure 1.2: Stack approach.*

![Stack approach diagram]

The research also needs to ascertain whether there are sufficient advantages of using a three dimensional (3D) representation over a two dimensional (2D) representation to justify the likely increase in processing and time overheads. Finally, research will be conducted into the current state of algorithm animations. The aim of this is to find out how far this field has moved on in the last ten years and see whether it has kept pace with the advances in graphics that have been made over the same time period. If there have been no real changes in this area the reasons for this will be explained. If there have been advances in this area then they will be described and implemented in the developed system if the approaches are feasible.

### 1.3 How the project will be judged

The project will be considered a success if it provides a teaching aid that lecturers would be happy to use to help them teach sorting algorithms to their students. Although this is something that cannot be measured until the application is developed, the project will have been a success if all of the requirements specified later on in this document are met. A possible method of assessing the developed system would be to give the finished system to a range of lecturers and students (in the field of Computer Science), and ask them to assess it on a number of points such as ‘clarity’, ‘ease of use’, ‘functionality’, ‘robustness’ and ‘speed’. If they were asked to score it on a scale of 1 (excellent) to 5 (very poor) for each factor mentioned then the results could be
collaborated to see how successful the system was in each area. If both the lecturers and the students rated it highly then it would prove that the developed system was a success. The ‘ease of use’ point is possibly the most important, because although many similar systems to the one proposed currently exist in various guises, almost all have been discredited in some way due to issues with the usability of the systems. Therefore, if the system scores highly in terms of ‘ease of use’, as well as meeting the requirements specified later, then the project can be considered a success.

However, this only covers the development aspect of the project. Research needs to be conducted to discover the best approaches available to implement such a system. If all of the best approaches discovered are successfully used then the project can be deemed a success. It is important to note that due to the quantity of research needed for this project there will be time constraints that may mean not everything will be implemented. Therefore, as long as all of the research findings are discussed and justifications are given to why any aspects of the final design are excluded from the developed system then the project can still be deemed a success.

### 1.4 What new skills need to be learnt?

To implement the proposed system research needs to be carried out into how animations are created and rendered graphically in the chosen programming language. Research also needs to be conducted into how best to implement algorithm races. There are likely to be other skills needed during the course of the project but these will become more apparent once the requirements gathering process is started, and later on when the actual development of the system begins. Finally, research into the various sorting algorithms to be included is clearly necessary, as is an investigation into the most recent concepts and theories of Human Computer Interaction (HCI).

### 1.5 Similar work

There are currently numerous web pages on the Internet showing sorting algorithms in animated form. These include Bubble Sort, Insertion Sort, Selection Sort and Quick Sort. For example, [3] gives detailed analysis into Bubble Sort, Quick Sort, and Insertion Sort, and shows how each works. However, in the majority of cases the animation has to be run alone, and is generally in the stack format shown previously. This means there is no option of running an algorithm race, and there is only one method of showing the animation. Each instance encountered so far also ran in an applet placed in a web page, whereas the proposed application plans to use a GUI in an applet to launch the algorithm animations. Therefore it is clear that some of what is proposed is already available, but it needs to be collated and built on to provide a truly usable system.

There is also significant research into HCI with GUIs. This will be a key aspect of the project as one of the goals is to provide a system that is easy to use, and this is directly related to HCI [4]. There are numerous sites on the Internet advocating good HCI principles to follow [5], [6], as well as books such as ‘Interaction Design’ [7]. By following these principles an excellent system will be developed in usability terms.

### 1.6 Which areas may require deeper investigation?

One area that will require serious investigation will be which methods of animating an algorithm work best as a teaching aid. At the moment the majority of implementations have been of the stack format shown previously. However, there are a number of other possibilities in both 2D and 3D. In terms of 2D there are the block and stack approaches described earlier, and the so-called dots approach. The dots technique is not used as frequently as either of the others, but may have certain advantages in terms of ease of understanding over the other approaches, thus potentially making it a better technique to implement. 3D systems are not currently widespread. Systems such as JCAT [8] do provide a 3D view, but its effectiveness in comparison with conventional 2D approaches is yet to be measured. However, large amounts of research are currently being conducted into 3D interfaces, for example ‘The Task Gallery’ is a research project run by
Microsoft that aims to see whether 3D methods of representing data on computers are more effective than 2D. Therefore it is definitely an important area that this project must study in some depth.

The project will also investigate the current state of the research for the animation of sorting algorithms. The intention is to see how this field has developed over the last 10 years, and understand why there have been advances, or if their have not then why has the field become stagnant? This area is likely to require deeper investigation because it is probable that there will be contrasting viewpoints on the best approaches for solving this problem.

1.7 What resources are needed?
The resources needed for the development of the proposed system are very modest. All that will be required is access to a development tool. This will need to provide all of the standard functionality you would expect such as error checking, compiling, and a run-time environment. On completion of the system it would be ideal if there was access to a lecturer currently teaching the course as well as a number of students, so that feedback could be given on how successful the system is. Finally, if the system is to be web based then a web host will be required so that the system will always be available to users. Sufficient storage space will need to be acquired from the host so that all of the necessary files required to run the system can be stored together.

1.8 Which aims are achievable?
The aims of the project are entirely realistic. As long as sufficient effort is put in then there is no reason why the developer should not be able to implement the system. Unexpected problems are bound to arise though. Although allowances for these occurrences can be worked into the project plan in the form of slippage allowances, it may be the case that the final application has to be scaled down to some degree. This is most likely to be in the form of not implementing all of the specified sorting algorithms.

The depth of research is also something of an unknown quantity at this time. It is possible that all of the relevant sources could be found immediately. If this turns out to be the case then there is no reason why the sources cannot be analysed thoroughly, new proposals made if appropriate, and a system implemented based on these findings. However, if this is not the case then there will be significantly less time to implement the ideas the research uncovers. Again, this could lead to a scaling down of the final application.

1.9 Functionality that will not be considered
Although the project will implement as many of the sorting algorithms as possible in as many different forms, time constraints could mean not everything is implemented. The project will not look deeply into any of the psychological theories of teaching. This is such a large area that it is beyond the scope of this project. Finally, HCI issues will be looked at as far as providing a good interface is concerned, but no new radical ideas or theories will be considered.

2.0 Literature Review
Within this project there are two specific areas that need to be investigated before the actual task of developing the final system can begin. These areas need to be understood completely for the project to have a firm foundation to stand on. There is also a third area of research, which although not fundamental to the concept of sorting algorithms, will still play a role in determining how successful the final system is.
The first key area is the very basis for this project. This is the field of sorting algorithms. The history and evolution of sorting algorithms will be looked into carefully allowing key developments to be clearly understood. Investigation into the current state of sorting algorithms, as well as where the future lies for sorting algorithms, will also be necessary, as this will be interrelated with the future possibilities for algorithm animation systems.

The other key area, and undoubtedly the most important is the animation of algorithms. The research in this area will need to discover why the animation of algorithms is important and answer the question ‘Why did such systems arise?’ Since one of the goals of this project is to provide a piece of software it will be necessary to critically evaluate existing systems. This will ascertain what functionality they provide, and how beneficial this is to users. It will also help highlight any problems with the existing systems and discover the reasons for difficulties the users experience. For example, is a problem due to a lack of functionality within a system, or is it a more fundamental flaw with the approach being used to animate algorithms? This research will highlight some of the functionality that will be required from the final system, as well as any problems with existing systems that need to be avoided.

Finally, a brief investigation will be conducted into GUIs. This will highlight the main advantages and disadvantages of using a GUI in a system. Within this section some investigation into HCI will be carried out. Although important to the usability of the final system a fundamental understanding of HCI is not crucial to the success of the project, hence it being covered in this section of the research.

2.1 Sorting Algorithms

2.1.1 Why do we need sorting algorithms?
According to Knuth [1] virtually every important aspect of programming arises somewhere in the context of searching or sorting. This highlights why it is so important to have efficient sorting algorithms. Sorting is crucial to computing, as almost all data is more meaningful if it can be sorted and displayed in various forms. If efficient sorting algorithms did not exist it is plausible that it would take a lot longer to complete tasks due to the quantity of data computers typically process for even simple tasks. It could also potentially mean an increase in the price of computers, as users would require more processing power to deal with the increase in the number of operations. One thing that is beyond question is that efficient sorting algorithms allow us to do more operations faster. However, there is also the added complication that numerous sorting algorithms are needed because different algorithms behave differently for different data so no algorithm is best under all conditions [2].

2.1.2 A brief history of sorting algorithms
According to Yung [9] the first machine for sorting was invented at the end of the 19th century, when Hollerith devised an ingenious electric tabulating machine to meet the need for better statistics gathering. This was in 1890 for the Census Bureau in the United States. When electronic computers arrived in the late 1940’s sorting was intimately involved with their development; in fact, there is evidence that a sorting routine was the first computer program ever written for a stored program computer [10]. Mauchly based the first published discussion of computer sorting on a lecture in 1946. This article described Insertion Sort and Binary Sort, and stated the complexity of each algorithm.

In 1956 a Ph.D. thesis written by Demuth [11] helped lay the foundations of the Computational Complexity Theory. This basis is still used today throughout computing to determine how efficient a sorting algorithm is. Demuth’s paper considered three abstract models of the sorting problem, using cyclic, linear, and random-access memories; optimal or near optimal methods were developed for each method. Although no practical consequences flowed immediately from
Demuth’s thesis, it established important ideas about how to link theory with practice. In the same year Friend produced a paper titled ‘Sorting on Electronic Computer Systems’ which was a major milestone in the development of sorting algorithms. Although numerous techniques have been developed since, this paper remains remarkably up-to-date in many respects. Friend gave descriptions of several internal and external sorting algorithms, and paid special attention to buffering techniques and the characteristics of magnetic tape units. From this period until the mid 1960’s numerous new sorting methods were discovered. These included Merge Sort (1959), Shell Sort (1959), Tree Sort (1960), Quick Sort (1962) and Heap Sort (1964) [13].

Since the 1970s dozens of new sorting algorithms have been invented, but almost all have been variants on previous methods. During the same time span there has been significant research into alternative ways of approaching sorting. These have included adaptive sorting schemes and randomized binary search trees. However, although these new approaches seem to have some theoretical value, none have yet been proven to be effective in practice [14]. In the 1980’s the algorithmic development was primarily based around parallel and distributed algorithms. This work has built on the research conducted in the mid to late 1970’s and has produced some excellent results. For example, NOW-Sort has held the Datamation record (see below). However, these sorting algorithms have been designed to work on ‘disk-based’ and ‘in-memory’ data, and are quite different to the sorting algorithms spoken of previously.

In 1985 the Datamation benchmark was introduced [15]. This was a test to see which was the quickest technique to sort one million records from disk to disk, with each record consisting of 100 bytes. This amounts to approximately 95Mb of data. In 1987 the Datamation record stood at 980 seconds, but now due to significant progress in sorting techniques and hardware, the record stands at a mere 0.44 seconds. The Datamation benchmark is now deprecated because it is so easy to sort a million records.

Clearly a new benchmark was needed [15]. A number were proposed including Terabyte Sort [16] and Minute Sort. Although the current record for Terabyte Sort stands at 49 minutes it seems destined to go the same way as the Datamation benchmark in the future. Therefore the most important existing sorting benchmark is Minute Sort.

Minute Sort was introduced in 1994 to replace the Datamation benchmark. Minute Sort is identical to the Datamation benchmark, but the performance metric is now the amount of data that can be processed in a single minute of elapsed time. In 1995 the record for Minute Sort stood at 1.08Gb of data. This has since been improved to 21.8Gb, the current record. This record seems destined to continue to increase rapidly as advances continue in hardware and sorting algorithms [16].

### 2.1.3 NOW-Sort

The past few years have seen dramatic improvements in the speed of sorting algorithms, largely due to increased attention to issues of computer architecture. Network Of Workstations sort (NOW-sort) [17], developed by the University of Berkeley in the US, has been one of the approaches that has helped push back the perceived limits. NOW-sort has held the Datamation and Minute Sort records in the past few years, making it one of the premier sorting algorithms at this moment in time. Previously, state-of-the-art, shared memory parallel computers had held sorting records. However, NOW-sort has achieved high standards using a relatively modest ‘shared-nothing’ network of general-purpose workstations. NOW-sort is well suited to I/O intensive applications such as sorting. This is because the cluster environment it adopts is ideal for the development of efficient parallel applications because it provides performance isolation [18]. Therefore, by monitoring the behaviour of the applications on each node, the developer is able to isolate bottlenecks, and deal with them when they occur. Clusters also enable incremental
scalability [19]. Due to the nature of the distribution of the data to be sorted across a network of machines, communication overheads between the machines is a key issue for NOW-Sort.

2.1.4 The future of sorting algorithms
Modern computers are far more sophisticated than simple sequential programs can lead one to believe; instructions are not executed sequentially and in constant time. In particular, the memory of a modern computer is structured in a hierarchy of increasingly slower, cheaper and larger storage. Accessing words in the lower, faster levels of the hierarchy can be done virtually immediately, but accessing the upper levels may cause delays of millions of processor cycles.

Consequently, recent developments in algorithm design have had a focus on developing algorithms that sought to minimize accesses to the higher levels of the hierarchy. Much experimental work has been done showing that using these algorithms can lead to higher performing algorithms. However, these algorithms are designed and implemented with a very specific level in mind, making it infeasible to adapt them to multiple levels or use them efficiently on different architectures.

To alleviate this, the cache-oblivious algorithms were developed. The goal of a cache-oblivious algorithm is to be optimal in the use of the memory hierarchy, but without using specific knowledge of its structure. This automatically makes the algorithm efficient on all levels of the hierarchy and on all implementations of such hierarchies. The experimental work done with these types of algorithms remains sparse though. [20]

Although it is possible this approach will not prove to have any practical value, it seems certain that sorting algorithms will continue to advance such that we can process more data in the same period of time. The Datamination benchmark provides evidence for this. Whatever the future does hold for sorting algorithms, one thing that can be guaranteed is continued progress in the efficiency of sorting.

2.2 The animation of algorithms

2.2.1 Why do we need to animate algorithms?
Algorithm animation is the process of abstracting the data, operations, and semantics of computer programs, and then creating animated graphical views of those abstractions [21]. Lecturers have adopted it as a means to teach algorithms to students, as it appears to give the students a much clearer picture of how the algorithms actually work, as well as showing which is the most efficient in a given situation.

2.2.2 A brief history of algorithm animation systems
In 1966 the first primitive form of algorithm animation was created. This was the animation of link-list languages by Ken Knowlton at the Bell Telephone Laboratories. However, it was not until the 1980’s when the major development of algorithm animation began. In 1981 Baeccker, at the University of Toronto, created a video titled ‘Sorting out Sorting’ [22]. This has generally been credited with initiating the field of sorting algorithm animation [23].

The film took three years to develop and lasted for thirty minutes. It was designed to be a teaching aid, and contained the animation of nine different sorting algorithms using different visualization techniques to show the sorting process. It focused on three types of sorting. These were insertion sorts, exchange sorts and selection sorts. The film ends by showing an algorithm race of the nine different algorithms.

‘Sorting out Sorting’ was followed in 1984 by the first piece of software that animated sorting algorithms. This was called Balsa-I (Brown algorithm simulator and animator). Balsa-I is an
interactive system that supports multiple simultaneous views of an algorithm’s data structures and can display multiple algorithms executing simultaneously [24]. Brown and Sedgwick at Brown University in the US developed it. From this point many lecturers adopted algorithm animation as an aid to teaching sorting algorithms in Computer Science.

Balsa was followed up in 1988 by Balsa-II. This was an improvement on the original Balsa by becoming domain independent. It also enables the manipulation of images with multiple views as well as providing a scripting facility. This was the only descendent of Balsa-I though. Brown did however create Zeus in 1991 [25]. Zeus supports multiple synchronized views and allows users to edit those views and change a data items representation. Zeus is implemented in a multi-threaded and multi-processor environment, meaning it can animate parallel programs, although this is not necessary for such animations.

In 1990 Tango (Transition based animation generation) was developed by John Stasko at Brown University. Tango introduced the path-transition paradigm for animation design as well as a new conceptual framework that has been adopted by many later systems as their fundamental architecture. This was followed up in 1992 by XTango [26]. This was an X-screen version of Tango, which utilizes a path transition paradigm to achieve smooth animation. XTango in turn spawned Polka [27]. Polka was designed to build concurrent animation for parallel programs. It is an object oriented 2-D algorithm animation system (although this has been extended into 3-D in Polka3D). Polka3D provided 3D views and 3D primitives such as cones and spheres. Users require no knowledge of 3D computer graphics to use this system.

Since the invention of the World Wide Web and the Java programming language developers have shifted their focus to building on-line algorithm animation systems. These have the advantages of being platform independent and accessible at any time from anywhere. This has helped move the animation of algorithms away from the lecture theatre, with on-line systems now being used in distance learning.

2.2.3 The future of algorithm animation systems

Algorithm animation systems have become an integral part of teaching sorting algorithms to Computer Science students, and are clearly here to stay. Although such systems initially animated algorithms only in 2D, support for 3D animations is now becoming more common in standalone packages. However, such packages are not commonly used, as they have to be downloaded and installed. The current trend for algorithm animation systems is web pages, especially those of universities and other academic institutes, which display the majority of sorting algorithms in 2D. This is probably because they are more convenient as a teaching tool as they can be accessed by anyone from anywhere at anytime, so long as they have a connection to the Internet, although they are arguably less powerful because they typically attempt to convey less information to the user. Therefore it seems likely that in the future systems for animating algorithms will be predominantly web-based, and in time will show all algorithm animations in 3D if research proves 3D animations are more effective as a teaching aid compared to 2D animations.

However, although algorithm animation systems are widespread their usage is still not universal. This could be because existing algorithm animation systems typically have been designed without formal study of related teaching practices [28]. Research is ongoing to address this problem. Hamilton-Taylor and Kraemer at the University of Georgia believe that the appropriate starting point in the design of an algorithm animation system is to examine the tasks lecturers perform and the context of these tasks. They have therefore adopted a contextual design approach, in which the first step is to study the user’s existing tasks and workflow. Design proposals are made with the objective of improving the existing system, while a careful attempt is made not to displace aspects of the existing workflow process that enable the tasks to be accomplished. They believe that this design methodology will result in algorithm animation teaching tools that are far more
relevant to the needs of both lecturers and students. It appears obvious that any approach that maps the functionality of a piece of software more accurately against a group of users' requirements is bound to be an improvement on previous approaches. This is something that this project will attempt to incorporate. By gathering the opinions of end users throughout the requirements, design, and testing phases of development their needs and opinions should be thoroughly understood. Mapping these to the final product should ensure a system that meets their requirements is created.

Members of the Computer Science Department at Princeton University in the US certainly see the future of algorithm animation as web based. According to Hausner and Dobkin, the development of Hyper Text Mark-up Language (HTML) has made the Internet accessible to people with little technical expertise. However, the amount of interactivity on the Internet is limited because users can only request prepared data, be it as images, animations or text. The interactivity needed for algorithm animation, together with the need for text explanations of the algorithms, may be supplied by carefully crafted animations in Java or some other web-based programming language [29]. This approach seems destined for success due to the explosion of learning now being done across the Internet. The proposed system will attempt to provide an animation system that gives the user a web based algorithm animation system that they can manipulate using a number of controls. This should ensure the system is available to almost all of the users who are likely to wish to use it at all times, and provide them with the functionality they need to fully understand sorting algorithms.

An ideal algorithm animation system will allow the user to run a program, choose ‘interesting’ data, change the data interactively, and ask about the program's objects and behaviour. Evidence of the need for this type of program has been highlighted in a number of papers. [30] Speaks of the need for an interactive relationship between the animation and the user to enhance learning, and also speaks of the importance of allowing the user to set up different scenarios using different types and amounts of data to enable them to discover which sorting algorithm is best in any given setting. [31] Agrees with these conclusions, and emphasises the importance of text descriptions in a system to backup what the user is learning from the animations.

Existing animation systems present users with a choice of algorithms and data, but do not allow a great deal of interaction or questioning. Real-time interaction requires a great deal of computing power. The user should be able to change the data with some interactive device, and sees the effects of the change immediately. Of course, nothing is instantaneous, but the illusion of instant response can be provided if the algorithm is run on the changed data at video rates, or 60 times per second. If there is a lot of data, the algorithm will not be able to keep up with the user's changes. As computers improve and get faster, larger data sets and more interesting animations will be accessible to real-time interaction.

As with everything in computing it is hard to predict the future of algorithm animation. It is safe to say that this approach to teaching sorting algorithms is here to stay and future systems promise to be more intuitive and beneficial to the student.

2.3 Existing sorting algorithm animation systems
Current algorithm animation systems can be divided into two groups. These are web-based systems that run in a browser window in the form of applets, and stand alone systems that require installing.

2.3.1 An overview of Polka
Polka [27] is a stand-alone general-purpose animation system that is well suited to building algorithm and program animations. Originally it was designed for Unix but now a Windows version has been created, called PolkaW. One of the first foci for the system was to build
animations of parallel programs and computations, but Polka also has the ability to animate serial programs. As stated previously, Polka is a descendent of Tango, but is more powerful and flexible.

Polka includes an interactive front-end called Samba. Samba is an animation interpreter that reads one ASCII command per line, and then performs that animation directive. The chief advantage of Samba is that a program in any language can be annotated to generate these ASCII commands, thus driving a visualization of the program.

2.3.2 Polka’s functionality
Polka provides the user with a colour GUI, real-time animations in 2D, as well as a reasonably easy to use system. It also provides users with its own high-level abstractions to make the creation of animations easier and faster than with many other systems.

In terms of sorting algorithms Polka is a useful tool. In the examples section of the program there is code for Shell Sort, Quick Sort, Heap Sort and Bubble Sort. Although this is by no means an exhaustive list of sorting algorithms, it gives a useful basis for the user to build on. This is because the real purpose of Polka is to allow people to build their own structures and animate them. There are also numerous other example animations relating to mathematical problems. For example, the Towers of Hanoi problem, Prim’s minimum spanning tree and various breadth and depth first searches.

The control panel of Polka looks dated but is functional. It consists of a pause/run button, a step through button that takes the user through the animation one step at a time, and a scroll bar that determines the speed of the animation.

2.3.3 A conclusion of Polka
Polka appears to have been a good system in its time, and still serves a purpose. However, it only contains a limited number of sorting algorithms. If a user wishes to look at any more then they must implement these new algorithms themselves by writing the appropriate code. The GUI is extremely dated by today’s standards.

2.3.4 Screenshots of Polka
Figure 2.1: The Polka console where a user specifies the number of elements in an array and what the values are of these elements. The console prints the ordered elements to the console screen.
2.3.5 **An overview of online sorting algorithm animation systems**

There are numerous online systems that provide an animation of sorting algorithms. These systems are generally provided by universities, or other academic institutes as a means for their students to better understand sorting algorithms. Generally, they are provided in applets that require nothing more than an Internet browser to view. This approach means that anyone anywhere can view these animations. So far, the majority of online systems are in 2D, but some 3D systems are starting to appear on the Internet, although these tend to require a Java download before they will function.

2.3.6 **Online systems’ typical functionality**

Such systems have limited functionality. They merely have the ability to show the user an animation of the majority of the sorting algorithms. These almost always include Quick Sort, Bubble Sort, Merge Sort, Heap Sort and Selection Sort. Some systems allow the user to specify the speed at which the algorithm will animate, while a few can show several algorithms sorting simultaneously. However, the majority do not allow the user to specify how many elements are to be sorted, and none of the researched systems have allowed the user to specify the actual values of the elements. There is no support for users to implement their own algorithms in web-based systems but this is not a fundamental problem as it is not the intention of such systems.
2.3.7 A conclusion of online systems
Although simple compared to standalone systems online systems generally do exactly what they are designed for successfully. They allow users to see how the various sorting algorithms work, so in terms of being a teaching aid they are successful. However, these systems normally offer no description of what is currently happening in the animation so are of little use unless the user already has some knowledge of sorting algorithms. These systems are simple to use though and accessible from anywhere at anytime, so complement a lecturers teaching.

2.3.8 Screenshots of online systems

Figure 2.4: On some of the more complex online systems the user is allowed to specify the type of input data (e.g. random, ascending, or descending), as well as the number of elements they wish to see sorted.

![Image of online system interface]

Figure 2.5: Insertion Sort, before the sort process has begun.

![Image of Insertion Sort before sorting]

Figure 2.6: Insertion Sort midway through the sorting process. The red line shows how much of the data has been sorted so far, while the blue line is scanning through the data looking for the correct insertion point.

![Image of Insertion Sort during sorting]

Figure 2.7: Insertion Sort once the sorting process has been completed. With some online systems the user is told how long has elapsed in milliseconds since the sorting started. However, in many systems they are not.

![Image of Insertion Sort after sorting]
Figure 2.8: Some of the more complex online systems allow the user to specify the number of elements to be sorted, as well as the type of data to sort and how long the sort process has taken in milliseconds. However, none allow the user to specify the values of the elements, and very few allow two sorts simultaneously.

2.4 A critical evaluation of existing systems

Although there are numerous existing systems none has ever been accepted by academics across the globe as the definitive solution for teaching sorting algorithms. Stasko believes that this is because although many people believe that algorithm animation is useful, no empirical evidence has ever been presented supporting this belief [32]. Stasko and his colleagues conducted a study designed to ascertain whether algorithm animation was beneficial to students in the learning of sorting algorithms. This study used the XTango system, and while it did support the view that algorithm animation systems helped students learn, the results showed the benefits were only marginal. They concluded by stating that algorithm animations will not benefit ‘novice’ students just learning the topic as much as more advanced students. These ‘experienced’ students may use the animation to refine their understanding of a particular algorithm, whereas ‘novice’ students would benefit more by actually constructing an algorithmic animation rather than viewing a predefined one. This paper was published in 1993 and still seems accurate.

Despite these limitations algorithm animation seems here to stay. When it comes to choosing what sort of system to use it is a case of what the user requires. If the user simply wishes to learn how sorting algorithms work then online systems seem the best approach as they are simpler to use, and are accessible to anyone from anywhere. These are only suitable though in addition to teaching, as they do not provide sufficient descriptions on how each sorting algorithm works. If the user wishes to implement any of their own algorithms or animations then they have to use a standalone system such as Polka. Such a system provides a GUI allowing the user to specify the number of elements to be sorted, the values of each element, and then allows the user to specify the speed of the animation, as well as providing facilities such as step–through so that the user can see what happens at each stage of the sort process.

Such standalone systems do not appear to have taken the user into sufficient consideration whilst being designed and this has led to the majority having poor usability. If such a system was developed properly it could offer the user a number of benefits over online systems. For example, experienced users could try out new sorting algorithms and demonstrate them to other users of a similar experience level to receive feedback about the new algorithm. This would be an excellent feature if the user wished to show the new algorithm at a conference. Novice users could also
learn more from such a system. As stated previously, users would learn more by actually building
the sorting algorithms they are learning about, rather than just watching them.

However, although existing standalone systems do offer additional functionality to online systems
they generally do not benefit the user significantly in terms of learning about sorting algorithms.
This is because the majority of standalone systems lack a textual description of what is happening
in the animation and also have animation builders that are extremely complex and difficult to use.

2.5  Graphical User Interfaces

2.5.1 What are Graphical User Interfaces?
The term GUI refers to the techniques involved in using window graphics, along with the
keyboard and the mouse, to provide an easy to use interface to computer programs [33].
Rudiments of a GUI include such things as windows, pull-down menus, text components,
buttons, scroll bars, and iconic images. A system's GUI, along with its input devices, is
sometimes referred to as its ‘look and feel’ [34].

2.5.2 Why do we need Graphical User Interfaces?
GUIs are widely used today because they have brought computing to the masses. Previously
computing was the domain of people with specialized knowledge of syntax that forms commands
to drive command line interfaces. Clearly, this required a user to understand lots of unintuitive
commands before they could get tasks done. The introduction of GUIs has changed this, now a
user simply needs to know which icon to press to perform a task.

GUIs offer the following attractive qualities [35]:

- An easy means for data entry and modification.
- An attractive and easy-to-use interface between humans and machines.
- Good screen management when multiple applications are running.

They also provide a better means of communicating with a machine by [36]:

- Extensive use of visual control items.
- Allowing data to be manipulated intuitively on screen.
- Interface consistency E.g. Microsoft Windows.

2.5.3 Human Computer Interaction
HCI can be defined as ‘the discipline concerned with the design, evaluation, and implementation
of interactive computing systems for human use and with the study of major phenomena
surrounding them’ [36]. Therefore, HCI is concerned with the joint performance of tasks by
humans and computers.

As a professional community, HCI dates from about 1982, the date of the Gaithersburg
conference on human factors in computing and about the time of the commercial emergence of
the personal computer. During the ensuing two decades, there has been an increase in research
and development activities surrounding computer interfaces and the use of computers. The
development of the area has reached the point where the first steps are being taken to codify the
field.

2.5.4 Human Computer Interaction in Graphical User Interfaces
Numerous guidelines, standards and techniques have been developed concerning HCI issues in all
aspects of computing, but they have been particularly abundant in the GUI area. This appears to
be due to the fact that GUIs are now the dominant interaction medium between humans and computers hence they require the most research [37]. Some of the best-known guidelines for GUI have come from luminaries in the field as Jakob Nielsen. Nielsen has advocated numerous ideas that allegedly make a GUI easier for users to interact with. Although some of the ideas advocated are universally accepted (e.g. keep colours such as red and blue apart on the GUI as the human eye cannot differentiate easily between them) and easy to judge, some are far more obscure and contentious. For example, what is the best way to judge the usability of a GUI? Is it best to break the problem down into a number of tasks and time how long it takes the user to complete each section, or is talking to the user as they go through the task a more accurate way to discern how they feel about the usability of the system? Therefore, despite the numerous guidelines there are many that are not universally accepted so care must be taken to use only appropriate guidelines that fit in with the context of this project.

2.6 A conclusion of the research
The research conducted in the two major areas stated in the introduction has confirmed a number of ideas. Firstly, the investigation into sorting algorithms has confirmed them to be central to the discipline of computing, indeed it has been ascertained that sorting is one of the most common operations that programs deal with in some shape or form. This is confirmed by numerous sources, including Donald Knuth, one of the major luminaries in the field of Computer Science.

Therefore it is clearly crucial that to design programs efficiently an individual must have a solid understanding of the fundamentals of sorting algorithms. Since the efficiency of programs is hugely important as it effects how quickly they run, this point is above argument.

The other key point provided one of the most controversial questions thrown up by this investigation - ‘Do algorithm animation systems actually improve students understanding of sorting algorithms?’ A number of research projects are currently ongoing into this topic, and so far the results show that animating sorting algorithms does not benefit the students significantly. A Ph.D. published by Colleen Kehoe [38] looked into the effects of animating algorithms on students understanding of the algorithms. Her results were not what many had expected.

‘...results showed that neither the addition of static graphics, nor animated graphics had any effect on learning as measured by a multiple-choice post test.’

She goes on to conclude:

‘So can animations be useful tools for teaching algorithms? The answer so far seems to be “Yes”, “No”, “Sometimes”, and “You’ll never find out using that method”. A question that draws on so many fields (computer science, visualization, education, psychology, cognitive science, sociology, etc.) is bound to have many different answers and different reasons behind those answers. One point seems to be clear, however: that simply throwing an animation into the mix of resources available to students does not magically improve their understanding of an algorithm.’

However, some investigations have revealed positive results and one such investigation was conducted by Mayer [39]. In a series of experiments between 1989 and 1992, Mayer demonstrated that illustrations (both static and animated) can have a dramatic positive effect on learning, but only if certain conditions are met. His conditions for meaningful learning are:

• ‘explanative text’- the text must present a cause-and-effect system that allows for qualitative reasoning.
• ‘sensitive tests’ - the performance measures must evaluate the learner's understanding and qualitative reasoning about the system.

• ‘explanative illustrations’ - the illustrations must help the learner build a runnable mental model of the system.

• ‘inexperienced learners’ - the students must not spontaneously engage in active learning processes such as the construction of a runnable mental model of the system.

At the moment opinions appear to be divided within the computing community, but even the most fervent of critics accept that given the correct design involving liaising with both students and lecturers, such a system would be a vital tool in the teaching of sorting algorithms to students. Therefore provided the results of the most recent investigations in this area are followed such a system will be beneficial to the students.

The research has also highlighted the importance of GUls in computing today. One only has to look at Operating Systems (OS’) to see that all popular OS’ have a GUI. One of the major reasons for this has been the explosion of users of computers, with the majority coming from non-technical backgrounds. Since computers are now used predominantly by users who do not understand the syntax required to use a command line driven interface, the designer of a program likely to be used by non-technical users has no alternative but to include a GUI as part of the system. Although it can be argued that technical people will use an algorithm animation system, a straw poll of students on the developer’s Computer Science course showed that the majority do not feel confident using the command line; therefore the inclusion of a GUI is vital.

Finally, the field of HCI was found to be far more ambiguous than was expected. Although a number of principles and guidelines have been found that are relevant to this project, an equal number have been found that either are inappropriate to the project, or are not agreed on unanimously by experts in the field of HCI. Those that are relevant and are universally accepted will be used, while others will be left out. In instances where there are no guidelines to follow research will be conducted to discover what the end users require in the given situation.

In conclusion, the literature review has validated the project proposal. The proposed system will consist of a GUI the user will use to launch the various tasks. The interface will adhere to standard HCI usability guidelines. These will include consistency between any icons uses, a simple and clear interface, help and support where necessary, short menu paths for the most common tasks, and the GUI components will be sufficiently spaced out so that there will be no feeling of crowdedness on the screen.

The sorting algorithms provided will consist of the Bubble Sort, Quick Sort, Insertion Sort, Selection Sort, and Heap Sort. The reason for including these sorting algorithms is because research has shown these to be among the most common sorting algorithms to be taught to Computer Science students. The system will be web-based. The reasoning behind this is mainly due to the increase of distance learning that is occurring at universities and other teaching institutes. Such a web-based system will mean that the students can access the system at any time making the system far more useful for revision purposes, as now a student will not have to have access to a specific machine where the program is installed, but will be able to access it at any time providing they have access to the Internet. Java is likely to be the programming language used to develop the system, as this is a language commonly used for similar systems and is also a language the developer is competent with.
3.0 Requirements Analysis

The aim of the requirements analysis section is to examine existing algorithm animation systems to discover what techniques work effectively in terms of animating sorting algorithms, as well as the HCI principles used to design the interface. This will enable desirable aspects to be identified for inclusion in the design of the final system, while also highlighting bad practices that need to be identified so that they can be avoided.

3.1 System assessment surveys

The system assessment survey has been designed to assess two main factors of existing systems. Firstly, it aims to determine which web-based systems provided the best experience for users in HCI terms and why. Secondly, it aims to discover which animation techniques for sorting algorithms are most effective in terms of the user understanding the sorting algorithm. If the survey can discover why certain techniques were more effective than others then the best approaches can be combined to provide a system that is more effective as a teaching aid than any of the existing systems, as well as a system that is equally good in HCI terms.

A total of eleven systems were tested, of which ten were web-based. The reason for this apparent discrepancy was that the research already undertaken has strongly suggested that the final system should be web-based due to its apparent benefits to students outside of lectures. Therefore, it was clear that more could be learned from examining current web-based systems than by looking at stand-alone systems. The single stand-alone system was assessed because it was felt that it could provide some useful insight in terms of algorithm animation techniques. Another important point was the actual sorting algorithm used to judge the effectiveness of the algorithm animations. In each instance Quick Sort was used, as this was deemed the most complicated of all the sorting algorithms to be implemented in the system. If a system could effectively show how this algorithm worked then it was felt it should be able to show simpler sorting algorithms equally successfully.

Full details of the eleven systems assessed can be found in appendix a. This section contains information on the location of each system on the Internet, the scores given to each system for each category, and screenshots of the important aspects of each system.

3.1.1 Human Computer Interaction factor results

Figure 3.1: Results from the HCI section of the system assessment surveys.
The results for HCI factors show that the average system does not meet the users HCI needs to an acceptable standard. This can be shown looking at the best average score for any of the categories, ‘Consistency’. This scored only 2.4 on average, approximately halfway between ‘average’ and ‘good’ on the scale used. This adds weight to the argument that the average system has not provided the users with a system that they find really easy to use. Three categories fall between 2.8 and 3, these being ‘Simplicity’, ‘Clarity’, and ‘Menu Paths’. These results strengthen the view that the systems are not as intuitive to use as could be expected. The two remaining categories were poorly implemented in the majority of systems. In the ‘Layout’ category, the average score was only 3.5, halfway between ‘average’ and ‘poor’, while the worst scoring category was ‘Help and Support’. This averaged 4.5, with a best score of only 3. This seems especially poor, as a first time user of a system tends to rely on the systems help to some extent to help them learn how to use the system. If the help section is poor, or non-existent, then the user will quickly lose confidence in the system and become frustrated with it. It seems clear from these scores that HCI issues have been at least partially ignored during system development for the majority of the existing systems. Looking at the average score for the ‘Overall’ category, only 3.2, further enforces this opinion.

Despite the average scores being low in usability terms, two systems were clearly better than the others. These were the ‘Java Sorting Animation’, which averaged 2.4 per category, and the SAAS, which averaged 1.6 per category. These systems clearly have considered the users needs during development and as a result have far better usability that the rest of the systems assessed.

3.1.2 An analysis of the Human Computer Interaction results

The results show that the majority of systems score only average or worse in the majority of HCI areas. However, it is more useful to analyse what has been done well, so that these aspects can be incorporated into the design if appropriate, and what has been done badly, so that these techniques can be avoided, or remedied, in the design.

For the first three points it appears that only SAAS followed any HCI principles during development as it scored ‘excellent’ on each point. From this it appears that following relevant HCI principles should ensure that the system meets a user’s usability needs. The next point, ‘Help and Support’, was one in which no system scored better than ‘average’. From the research conducted it appeared that this area was included as an after thought, if at all. There were no instances of step-through guides, FAQs, or trouble-shooters, all of which would have made the systems much more accessible to the intermediate user targeted by these systems. The next criteria was ‘Menu Paths’. This was an anomaly as only one system had menus. This system did not exploit menus to there full potential as using them did not make the system feel any easier to navigate. However, in theory menus should be more intuitive for the user to use. Therefore, menus will be included in at least one of the prototype layouts developed. If they can be shown to be advantageous to the system as a whole then they will be included in the final design of the system. The final criteria was ‘Layout’. This returns to the first three criteria spoken of, in that a system should score well for ‘Layout’ if it simply follows relevant HCI principles. Clearly, the average score for a system was made up of the individual factors, so if the individual factors score well in HCI terms, then the system as a whole will score well.
3.1.3 Algorithm animation technique results

Figure 3.2: Results of the algorithm animation section of the system assessment survey.

In Figure 3.2 the first set of three bars represents stack view, the second set of bars represents block view, the third set represents dot view, and the fourth set of bars represents a technique encountered called colour stack view.

The results show clearly that the block view approach to animating algorithms was the most effective technique in all three categories. Despite this it only score above ‘average’ in one category. This was ‘How confident would the user be at describing Quick Sort after watching the animation once?’ For the remaining two categories block view was significantly better than the other approaches assessed but still scored poorly. The second most effective technique appears to be colour stack view. However, there was only one instance of this technique so these results may not reflect the effectiveness of this technique in general. The third most effective technique was stack view. This technique scored poorly, averaging worse than ‘poor’ on each category. The least effective technique was dot view. This approach averaged 5 (very poor) in each category.

Although the results reveal problems with the majority of systems there were two systems that used innovative techniques that made understanding the sorting algorithm animation far easier. The ‘Interactive Tutorial for Quick Sort’ was one of these. It provided step-by-step details of what was happening, enabling the user to have a much better understanding of the process. ‘Data Structures and Algorithms: Quick Sort’ was the other system that gave the user more than just an animation. This system highlighted which section of code was currently being executed. Although this technique requires knowledge of Java (the programming language used in the system), it gives the user a greater understanding of what was going on than systems that had clearer animations but no feedback. This is because at any point in the execution the user is able to see which section of the code is being executed. Therefore, as long as they have some knowledge of Java they should be able to form at least a rough idea of what is happening.

3.1.4 An analysis of the algorithm animation technique results

Of the techniques looked at none can be said to have been effective in teaching Quick Sort to the user. The reason for this could be partly due to the complexity of Quick Sort, but it is certain that the majority of systems were poor due to the lack of a detailed explanation being provided while
the animation was executing. Some systems simply did not have any explanations, while some had step through buttons that still did not make the animation any easier to understand. The most successful system succeeded because it told the user what was happening in words in addition to the animation. Another effective technique was highlighting where in the code execution has reached. Although this is less useful that a description of the current stage, together they should complement each other (and the animation), and should provide a system that helps the user understand how the various sorting algorithms work.

Of the four animation techniques investigated one appears to be of no use to the user whatsoever. This is dot view. This approach leaves the user confused about what was happening; in fact it was not even clear anything was being sorted in some instances. Of the other techniques stack view and block view appear to offer the most to the user, while the colour stack view appears to be a novel technique that has only been implemented in one system, with limited success in terms of user understanding of the sorting algorithm.

3.1.5 A conclusion of the system assessment survey results
The research into existing systems has unearthed a number of interesting points in regards to what will be included and excluded in the prototypes and the final system.

• Detailed help and support, including faqs, walkthroughs and a trouble-shooting guide need to be included to give the user assistance if they come across a section of the system they are unsure of.

• Menu-paths should be considered in one of the prototypes layouts developed, as they are associated with good HCI design for GUIs. If the feedback is positive then they should be included in the final design.

• Information should be provided to the user while the animation is showing the sorting algorithm executing. This should be in the form of descriptive text explaining how the algorithm works.

• Stack view and block view will be implemented, but dot view will not be as it offers the user nothing in terms of understanding sorting algorithms.

• The colour stack view technique will be left out, as there is a lack of evidence suggesting that it is useful for teaching sorting algorithms to students.

3.2 Human Computer Interaction design principles for algorithm animation systems
Good user interface design is critical to the success of a system, as highlighted by the research into existing systems. An interface that is difficult to use will, at best, result in a high level of user errors, at worst, users will simply refuse to use the system irrespective of its functionality. Due to the importance of the GUI further research was required into the field of HCI to ensure that the system met all relevant HCI principles, thus ensuring that the final system would provide the desired functionality in an easily usable system. This research focused on two major areas of HCI – interface layout and the use of colour in a GUI. These areas incorporate the seven HCI factors considered in the existing system assessment survey, but go into much more detail. This was because the survey highlighted the importance of good HCI design in terms of the systems success. It is crucial to understand what a user requires in terms of these factors as a failure to implement suitable measures for each point could lead to an interface that is unusable.
Although there are other areas that are often advocated as being crucial to the success of an interface, such as effective error messages, these were not incorporated. This was because it was not felt that they were hugely significant for a relatively simple interface such as this, so were left out due to time constraints. If more time were available then these areas would have been considered as they have the potential to improve the usability of the system slightly. Research was also undertaken into usability heuristics and principles, as advocated by Jakob Nielsen, one of the major authorities in HCI. These are generalizable abstractions intended to orient designers towards thinking about different aspects of their designs.

3.2.1 The interface layout
The layout of a user interface plays a major role in determining how successful a system is. This is especially true for certain types of system, such as safety critical systems where operators are faced with controls that affect a safety critical process so they simply cannot afford to select the wrong option. Although this system is not safety critical the same principles apply.

- Controls and displays that are functionally related should be grouped together. This is especially relevant for the animation controls of this system. For example, if the user is manipulating the animation they may be trying to find an acceptable speed for the algorithm to execute at. Therefore it is crucial that the buttons that speed up and slow down the animation are close to each other.

- Controls and displays frequently used by the user should be easily accessible. This is especially relevant in menu paths as the user will clearly see the items at the top of a menu before the items at the bottom, and taking the concept a stage further, the user will see things on a main menu before things on a sub-menu. This is relevant as menus are being considered as a navigation method for the system. Therefore, if menus are used then screens that are likely to be accessed frequently (e.g. the algorithm race setup screen that allows the user to set up an algorithm race) should be on the main menu, while screens that are likely to be used infrequently (e.g. an about screen stating details of the developer) should be placed in a sub-menu if there is not sufficient space on the main menu.

3.2.2 Colour
Colour is used in almost all algorithm animation systems as different colours are used to highlight different aspects of the animation. For example, in Bubble sort a coloured element often represents how far through the array the sorting algorithm has processed. Different colours are also used to highlight elements that are sorted, elements that have not yet been considered, and elements that are currently being considered. Clearly it is crucial that a colour maintains its’ meaning throughout else the users of the system will easily become confused.

- The number of colours used should be limited, and they should be used conservatively. No more than five separate colours should be used in a single screen, and no more than seven separate colours should be used in the whole system. Colour should be used selectively and consistently and should not be used to simply brighten up the interface. Although this point is extremely general it is applicable to this system as following it should ensure no sections of the interface are overshadowed by other sections. It is also applicable in terms of the colours used for individual elements within the interface. For example, a text box should be white with black text as this is what a user expects from a text box.

- Colour change should be used to show a change in system state. If a display changes colour, this should indicate that a significant event has occurred. For example, the
background colour of the animation panel could change colour to show that the sorting algorithm has finished.

- Colour pairings should be carefully considered. Because of the physiology of the eye, people cannot focus on red and blue simultaneously. Eyestrain is a likely consequence of a red and blue display. This is applicable when it comes to deciding which colours to use for highlighting elements that are being sorted. Existing systems frequently use red and blue together and this is clearly not a good idea given the physiology of the eye.

- A colour scheme for colour-blind people should be provided, and should be the initial scheme used. This is important, as approximately 8% of males are colour blind, meaning that if this aspect is not considered then it will make the system unusable to these people.

### 3.2.3 Design heuristics

There are a number of design heuristics that are relevant for a sorting algorithm animation system.

- Consistency – user interfaces should not surprise the user. For example, an action performed in one section of the system should give the same result as an action performed in another part of the system if the action is the same. This is relevant in this system as the user would expect the same thing to happen after an algorithm had finished sorting, regardless of whether it was a single animation running on its’ own, or an algorithm running as part of an algorithm race.

- Responsiveness – this factor measures the rate of communication between the system and the user. Generally, an instantaneous response is required. This is relevant in this system, as the animation controls should clearly have an immediate effect once they have been used or the user may not think they have worked correctly.

- Recoverability – the ability for users to recover from a mistake they have made whilst using the system. An example of this is asking the user if they really wish to quit the system when they have pressed the quit button. This is important, as the user may have selected this button by accident.

### 3.2.4 Usability principles

In addition to the design heuristics mentioned there are a number of usability principles that are relevant to the system. Usability principles were developed by Jakob Nielsen and can be applied to interface design.

- Match between the system and the real world - the system should speak the users’ language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. In the case of this system the language used should be as concise as possible while still relevant to the field of sorting algorithms.

- Error prevention - even better than good error messages is a careful design that prevents a problem from occurring in the first place. This is critical to the success of the system as if the user starts having errors through no fault of their own they will lose faith in the correctness of the system, and may begin to doubt the accuracy of the information it is trying to teach them.
• Aesthetic and minimalist design - dialogues should not contain information that is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility. This is especially important for this system as it aims to be a teaching aid, so any unnecessary information detracts from the clarity of the relevant information, thus making the learning process more difficult.

• Help and documentation - even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large. This is especially important for this system, as one of the major problems existing systems appear to have is a lack of decent help facilities.

3.3 Prototypes
A number of prototype screen layouts have been created with the aim of providing a final design that meets all of the usability principles identified in the previous section. Each prototype was given to ten Computer Science students of intermediate experience, as they have similar experience to the target users of the system. These users were asked to assess the prototypes against the following criteria:

• Controls and displays functionally related are grouped together.
• Frequently used controls are easily accessible.
• Use of colour is consistent and limited to no more than five different colours.
• No instances of colour pairings that are difficult for the eye to distinguish between.
• The layout is aesthetically pleasing and minimalist.

These HCI criteria were used, as they were the criteria identified in the HCI design principles for algorithm animations systems that were applicable to static prototypes. The remaining principles will only be able to be tested once an interactive system has been developed.

The most successful aspects of the initial prototypes were identified, and included in a new prototype. This was again assessed against the principles stated previously. More modifications were made which ultimately led to a final prototype that the testers unanimously accepted. This final prototype will be the basis of the design of the final system for the single animation screen.

3.3.1 The aims of prototype 1
Prototype 1 aims to allow the effectiveness of a vertically partitioned interface to be assessed by splitting the majority of the screen into two vertical sections. It aims to see how the users feel about the grouping of the animation panel and its related controls on the right-hand three-quarters of the screen, and the statistics, description, and colour key being grouped together in the left-hand quarter of the screen. This prototype also aims to see how effective the users feel the animation control buttons are in their current locations. Does the user feel it is appropriate to locate the animation speed bar above the animation panel separate from the other control buttons? Or does the user feel all the animation controls should be located together?

3.3.2 A description of prototype 1
Prototype 1 uses a menu bar to avoid wasting space on the interface. Below the menu bar the screen is split into two sections. The section on the left gives the user feedback on the status of the sorting algorithm being executed, as well as the statistics for the current sorting algorithm. There is also a text panel that contains a description of the sorting algorithm being executed. On the right hand side of the screen there is a label stating the name of the current algorithm being executed. Below this is a scroll bar that determines the speed at which the algorithm is executed.
Then there is the actual panel itself where the algorithm animation will appear. This is situated in the centre of the screen. Below this panel there are three buttons that allow the user to control the execution of the algorithm, with a panel showing the colour scheme in use appearing at the very bottom of the screen.

Figure 3.3: Screen layout for prototype 1.

![Figure 3.3: Screen layout for prototype 1.]

### 3.3.3 The aims of prototype 2
The main goal of prototype 2 is to see how the users respond to having a significant area of the screen devoted to buttons that will allow them to navigate the interface, rather than having a menu bar to allow them to do the same task. The buttons have been placed on the left of the screen therefore allowing the rest of the interface to be a more regular shape. Another goal of this prototype is to see how the users feel about a horizontal segmentation for the remainder of the interface. In this prototype the animation panel and its controls have been situated in the upper section, with the remaining components (description, colour key, and generated statistics) being located in the lower section. The prototype aims to see if the users feel this grouping is more intuitive than that used in prototype 1. Finally, it aims to see if the user feels more comfortable with the animation controls being situated to the side of the animation panel rather than above or below it.

### 3.3.4 A description of prototype 2
Prototype 2 differs from the first in that it has no menu bar. Instead it has a number of buttons down the left hand side of the screen. Each of these corresponds to a menu item from prototype 1. As with the first prototype there is a title along the top of the screen. Below this is a label stating which sorting algorithm is currently being executed, and below this is a space in the very centre of the screen where the animation is shown. The control buttons are positioned to the right of the animation, with the speed bar positioned directly beneath it. The statistics, algorithm description and colour key are placed into the remaining space at the bottom of the screen.
3.3.5 The aims of prototype 3
The principal goal of prototype 3 is to assess how effective the user feels the interface is when split into three sections, with the upper three-quarters of the interface being split equally into two vertical sections, and the bottom quarter covering the entire width of the screen. This approach gives the animation control components more space and clarity but they are now further from the animation panel. So this prototype aims to judge whether the user feels this is acceptable or not. This approach also gives the statistics, the algorithm description, and the colour keys more space and clarity so another aim of this prototype is to see if the user feels if this is more effective than either of the groupings provided in prototype 1 or prototype 2.

3.3.6 A description of prototype 3
Prototype 3 attempts to make the layout as uncluttered as possible. A title sits at the top of the page, with a menu bar below it. This menu bar again attempts to minimize the space that is used by buttons, and allows the user to reach any of the other facilities within the system. Below the menu bar the system is split into two. On the left is the panel where the animation will be shown; while on the right are the statistics, the colour key, and the status of the execution. This approach differs slightly from the other two in that it reduces the information that the user is provided, in an attempt to minimize the perceived clutter on the interface. The idea with this layout is that the user has the most important pieces of information immediately available to them, but if they require more information they can find it by selecting a menu item. Therefore some of the labels seen in the previous prototypes have been left out. Finally, at the bottom of this interface is the control bar. Although not immediately adjacent to the animation pane, hopefully it is close enough for the user to automatically associate its functionalities with the animation.
3.4 An evaluation of the prototypes

Prototype 1:
- The menu bar saves space and is as easy to read as the button system shown in prototype 2.
- The vertical partitioning worked well as a whole, although the users felt that it would provide each component with more space if the each section were of an equal size.
- Lots of space is provided for the animation panel. This should make the animation clearer as it can be shown in more detail in a larger space.
- Lots of space is provided for the description panel. This is useful as the testers feel this is an important area that they will use while the animation is executing.
- The control bar is situated in the wrong place. The users felt it would be more natural for the control bar to appear at the bottom of the panel the animation was shown in.
- The users felt it was not natural to associate the status, statistics, and description on the left of the screen with the animation on the right of the screen.

Prototype 2:
- The buttons on the left-hand side of the screen worked effectively but took up more room than the menu bar.
- The buttons on the right-hand side of the screen were far bigger than necessary, and appeared to be wasting space.
- The animation panel seemed crammed in.
- The information at the bottom of the screen was hard to take in as it is all so close together.
- The information at the bottom of the screen was obviously linked to the animation pane.
- The users felt the vertical segmentation was easier to relate to than the horizontal segmentation.
- The users felt that having the animation controls on the side of the animation panel was a bad idea. Although it was still obvious that they controlled the animation panel it was felt it was more intuitive to place the controls directly under the animation panel.

Prototype 3:
- It feels spacious – everything seems to have lots of room.
- No overhead label is associated with animation panel. The users feel this could cause problems in terms of them identifying what was being animated in the panel.
- The bar at the bottom containing the controls for the animation is not immediately identifiable with the animation panel.
- A description needs to be included to enhance user learning.
- Users feel the three way split does not work as well as a two way vertical split.
- Users feel that having all of the information about the algorithm being animated grouped together on the right-hand side works very well.

3.5 A revision of the prototypes

After analyzing the responses to the initial three screen prototypes a new prototype was created that attempted to combine the various positive comments regarding the first three prototypes. The aim of this revised prototype was to see if the users would find an interface acceptable that incorporated all the good points they had previously made regarding the first three prototypes. The revised prototype aims to test the following features:
- Vertical segmentation into two equal sized segments.
- All statistics and details in the right-hand segment.
- Animation panel in the left-hand segment, with the animation controls being placed directly below it.

Figure 3.6: The screen layout for the combined prototype.

The original ten testers were asked to give their opinions on the new layout, and again assess it in terms of the HCI criteria used previously. As before, they made a number of points.

- The vertical segmentation works well and leads to a clear structure to the layout.
- It is not obvious what the clear button in the menu bar is actually meant to do. If it is linked to the animation panel the users felt that it should be placed in the grouping of buttons below the animation panel.
- The label for the speed bar seems to imply that the whole area of controls below it is to do with speed. The users felt it would be appropriate to name this label ‘controls’, and to replace the speed label with two labels indicating ‘speed up’ and ‘slow down’.
- There is no information regarding what type of data is being processed in terms of its order.
- The system does not appear to provide a clear link to a section that will explain the algorithms in more detail.
- There is no information regarding what the type of animation being used to show the sorting algorithm is.

The negative points are all perfectly valid and were dealt with in a new prototype layout by addressing the concerns raised. The point made regarding a detailed explanation of the algorithm was deemed a major flaw in the system, as this was an aspect that many of the existing systems looked at had failed on. Therefore it was decided to add another item to the menu bar linking directly to an algorithm analysis screen.

3.6 The final prototype
Once modifications had been made to the combined prototype it was shown to the testers. The testers universally praised the prototype, having no concerns with its layout. Therefore although
this is only a static layout prototype, it will be used as the basis for the single animation screen as it has been successfully tested with ten users. This has proved that it has met a number of the HCI issues specified earlier. In terms of the interface layout this final prototype has met both points specified. These were:

- Controls and displays that are functionally related should be grouped together.
- Controls and displays frequently accessed by the user should be easily accessible.

It is clear from looking at the final prototype that all of the controls are grouped together based on functionality. The menu bar groups all of the navigation controls together, while the animation controls panel (at the bottom left of the screen) groups all of the animation manipulation controls together. The final prototype also places the options that are likely to be frequently accessed in easy to reach places. For example, the menu bar will allow the user to jump to any part of the system by simply pressing the mouse once, with no sub-menus needing to be navigated.

Another important factor discussed previously was colour in the interface. Due to the fact that this was a static prototype some features specified could not be tested, such as a change of colour to show a system state change, and the consistent use of colour. However, some have been met by this prototype. These are:

- Limiting the number of colours.
- Conservative use of colours.
- Providing a colour scheme for colour-blind people.
- Avoidance of colour pairings that the eye is unable to easily differentiate between.

This can easily be seen by looking at the prototype. There are only three different colours used (black, white and grey) and all of these are visible to colour blind users. There is clearly no place in the layout where an element is hard to distinguish due to the colour of a neighbouring component, and the use of colours is definitely conservative.

In terms of the design heuristics it has been impossible for a static prototype to prove that these points have been met simply because these are only factors that can be proven once a dynamic system has been implemented. This is also the case for the majority of the usability principles specified. However, the design does meet the aesthetic and minimalist design point, as there is nothing on the interface that the user felt should not be there, or detracted from the clarity of the interface.

Figure 3.7: The screen layout for the final prototype.
4.0 Requirements Specification

The requirements listed below have been ascertained after performing a review into existing work in the fields of sorting algorithms, algorithm animation, GUls and HCI design principles. The requirements analysis section has also highlighted good aspects of existing systems that appear beneficial to an algorithm animation system in terms of usability.

4.1 Gathering the requirements

4.1.1 What has been learnt from the requirements elicitation process?
A number of important points were discovered at each stage of the requirements analysis process. These are covered in detail below.

4.1.2 The system assessment surveys
The system assessment surveys highlighted a number of aspects that appeared to be central to the success of some of the existing systems, as well as highlighting areas where the existing systems let the users down. The aspects that were identified for inclusion in the final system from this area of requirements analysis were:

- Detailed help and support.
- Inclusion of the stack and block techniques for animating.
- Inclusion of information that states what is happening at each point while the algorithm is animating.
- HCI factors followed in the design process to ensure a usable system is developed.

Although some of these points appear obvious many of the existing systems neglected at least one of them causing the systems usability to suffer significantly. Detailed help and support is clearly critical to such a system as the main aim of the system is to aid in teaching how sorting algorithms work to students. If there is any aspect of the system that the user does not understand then they should be able to query this in a help facility. If a user is able to do this it will enhance their confidence in the system. This should make the process of understanding sorting algorithms easier as the user will feel the challenge is less daunting as the system is helping them.

The block technique for showing a sorting algorithm at work has been chosen for inclusion as it scored highest in the system assessment survey in terms of user learning. Therefore its inclusion is clearly important, as it appears to offer the best method for a user to learn how the selected sorting algorithms work. The stack view has also been chosen for inclusion and this decision is more contentious. Although it scored little better than the dot view (the worst scoring technique) it was decided to include this method as it was felt many of the existing systems simply did not support it appropriately with detailed enough descriptions stating what was happening. Therefore if sufficiently detailed descriptions are provided to work in conjunction with the animation, stack view will offer a valid alternative to block view. These two approaches should cover all of the potential users preferences in learning styles for the sorting algorithms included in this system.

Information regarding what is happening at each step in an animation is also to be included. This is because one of the major flaws of the existing systems appears to be that while they may show the animation very clearly, the user often simply has no idea what the logic is behind the algorithm. Therefore it was felt that a description that states clearly how the sorting algorithm being animated works would provide a valuable source of information to the user and allow them to understand what is happening in each animation much more easily.
The system assessment surveys highlighted the importance of following relevant HCI principles in the system. This was because many of the existing systems had their usability compromised significantly because there were fundamental flaws in the interface. For example, some interfaces had buttons that were difficult to select, and several had text that was difficult to distinguish against its background. Although these points seem insignificant, collectively they are seriously detrimental to a system’s usability, and reduce the users confidence in the system.

The system assessment survey also identified the following animation techniques that will not be considered for the final design.

- Dot view.
- Colour stack view.

Dot view will not be included because in none of the systems assessed was there any evidence that the dot view allowed the user to understand how a sorting algorithm worked. It was felt that even with the aid of a step-by-step description dot view was simply to abstract to be of any significant use. Colour stack view on the other hand potentially has some use. However, this approach was only used in one current system so it was difficult to determine how useful it could really be. Since it scored much worse than the block view approach it was decided that it would not be implemented, due to time constraints and the fact that it appeared to offer no real benefits.

Finally, the system assessment survey aimed to see how affective menu bars were in sorting algorithm animation systems. The results for this were inconclusive, as only one of the eleven systems tested contained menus, and they were poorly implemented in this system. Therefore the system assessment surveys could have no influence on whether or not menus were included in the system design.

4.1.3 Human Computer Interaction design principles for sorting algorithm animation systems

After looking at a wide range of HCI design principles a number of areas were identified that were relevant to algorithm animation systems. These were the following:

- Interface layout.
- Use of colour.
- Design heuristics.
- Usability principles.

These areas contain several principles each and were covered in the HCI design principles for algorithm animation systems section. As mentioned briefly before, it is critical that an interface lives up to a users expectations on every single point, as if it fails in even one seemingly insignificant place then the user will start to lose faith in the system. Therefore it is important that these areas and the principles within them are given sufficient consideration within this system. Following the identified principles should ensure that the system is easily usable by the system’s target users.

4.1.4 Prototyping

The prototyping section was invaluable as it allowed proposed interface layouts to be shown to potential users who then judged them against the HCI principles. This process was iterative, as there were three rounds of prototyping before the users unanimously agreed that the latest prototype was acceptable. The prototype design was aided by the HCI design principles for algorithm animation systems spoken of previously. They provided a basis from which all of the prototypes were developed. This basis included things such as consistent use of colour, sufficient
space for all items in the interface to be clearly identified, etc. Once all these aspects had been successfully included in the interface it was a case of modifying the layout and adding or removing items until the users were happy that it was intuitive and met their requirements. More information can be found in the prototype sub-section of the requirements analysis section on how the final prototype meets the HCI design principles specified.

4.2 User requirements

4.2.1 Functional requirements

The requirements below state the functionality that the final system will provide. They were derived by the literature review that looked into existing systems and ascertained what the lecturers and students required from an algorithm animation system. They were also further refined by the system assessment survey, as well as the prototypes. The feedback from these two requirements analysis processes enabled the aims of the system mentioned in the project proposal to be refined into system requirements that were as close to what the end user wanted from the system as possible given the time constraints.

- The system will be web based, so will be coded in a language that can be deployed on the Internet.

- The system will provide the animations of the following algorithms:
  - Bubble Sort.
  - Quick Sort.
  - Insertion Sort.
  - Selection Sort.
  - Heap Sort.

If time permits then further algorithms will be added to the system.

- The system will be able to show multiple animations simultaneously, in the form of algorithm races. Once all algorithms have finished they will be ranked in order of completion.

- The user will have the ability to choose different types of data for the animation of each algorithm. The available types of data will be:
  - Sorted.
  - Random.
  - Pre sorted.
  - Partially sorted.

- The system will provide a concise description for each sorting algorithm, stating what it does in simple English, as well as describing the technique used by the algorithm to sort the data, and the efficiency of the algorithm.

- The system will provide online help. This will tell the user how to perform simple tasks by providing step-through guides.

- The system will provide statistics for each animation as it sorts. This will include time, number of swaps, and number of comparisons.

- The user will be able to select the number of items that the algorithm has to sort.
- The system will include a control mechanism that will allow the user to pause the animation of an algorithm at any point. There will also be a function that will allow the user to adjust the speed of the animation, and a function that allows the user to step through the animation one operation at a time.

- The system will use colour to enhance learning. This will be done by assigning a colour to elements in the array depending on what category it falls into:
  - Sorted.
  - Not sorted.
  - Currently being considered.
A key will be provided to inform the user of what the colours mean.

- The system will give feedback on the current status of any algorithm that is currently being animated.

- The system will describe what is happening at each stage in the animation. For example, if the sorting algorithm is searching for the correct place to insert an element then this will be displayed in a textual format, as this has been shown to enhance learning by the research conducted.

### 4.2.2 Non-functional requirements

- The final system should be an effective teaching tool that will easily allow students to understand all they need to know about the sorting algorithms implemented.

- The systems’ front end will be a GUI. This will adhere to all of the guidelines described previously on the subject of HCI. This will ensure the system meets the users usability requirements.

- The system will have a logical and intuitive layout of controls.

- Colour will be used in the GUI to benefit the user. This will also follow the guidelines described above about the use of colour in interfaces. In addition, the default colour scheme will be developed for colour-blind users.

- The system will be reliable. No unexpected events will startle the user, and the system will not crash more than an acceptable percentage of the time.

- The system will provide user help that follows HCI principles. This will enable users to find solutions to problems they may encounter, giving them greater confidence in the system.

### 4.2.3 Additional requirements

For a user to access the system and use it to the maximum of its capabilities they must have the following:

- A computer with mouse, keyboard and monitor.
- Access to the Internet.
4.2.4 Justification for the requirements

Although the requirements analysis section justified why a number of the requirements were included in the system, there are several points that require explanation.

Showing a single animation is obviously necessary as this makes the fundamental goal of the system (teaching sorting algorithms to students) possible. Multiple animations are equally obvious. For a user to understand which is the most efficient algorithm to use in a given circumstance it is easiest to show this by making the sorting algorithms execute simultaneously in a race against each other. This is why the ability to choose different types and quantities of data is also important, as they allow the user to create different circumstances where algorithms may perform differently to what the user expects. Clearly, if multiple sorting algorithms are not implemented the user will have no way of judging which is the best to use in a given circumstance.

The five sorting algorithms were chosen for implementation because they were the most widely taught algorithms according to research. Therefore these five are most likely to be able to assist a user in learning how sorting algorithms work as they are the algorithms they are most likely to be taught.

Statistics were included because they allow the user to determine which algorithm finished first if they finish very close together, while feedback will state what the algorithm is currently doing. This may not always be clear to users. For example, if the algorithm is working out where to insert an element and the user does not realise this they may simply think the animation has crashed. By giving feedback this will ensure that this does not happen.

It was decided the system would be web based because the system aims to assist the students in their learning outside as well as inside lectures. Therefore, by placing the finished implementation on the Internet users would be able to access it at any time providing they have access to a computer with Internet access.

Animation controls were included so that the user could run the animation at the speed they felt most comfortable at. This also allows the users to move rapidly through sections they understand, and then slow the animation down when they reach a section they are unsure of. It was felt that this would enhance their learning as they could take their time on sections of the animation they did not understand fully. This was also the case for including a detailed description as the algorithm executed.

Online help is to be included as this will help to ensure that the users find the system as easy to use as possible. This will include a section describing how each algorithm works in detail, as well as stating the best, worst, and average performances of the sorting algorithms included. This section is aimed at helping the user to work out how to use the system, as well as allowing them to learn the more detailed nuances of the sorting algorithms once they have grasped the basics.

The non-functional system requirements are based around the HCI design principles spoken of earlier. The system developed must meet all of the specified criteria for it to be considered usable. This is as important as the functional requirements, as if the user feels the system is not usable then they simply will not use it, so will never get as far as using the functionality of the system. Therefore requirements such as the two listed below are clearly needed.

- Reliability.
- Logical and intuitive system layout.
Finally, the additional requirements were specified as if the user cannot meet these they will be unable to access the system.

## 5.0 Design

This section provides a design based on the requirements specified in the previous section. The design has been reached by performing a requirements analysis on existing algorithm animation systems, as well as by creating prototype screen layouts. This has enabled the highlighting of good and bad facets within these designs to be identified. These have then been included in to the design where appropriate.

### 5.1 Design considerations

A number of things had to be considered before the design could be started. The first of these was to decide whether the system was to be implemented using an object-oriented or a procedural technique. After studying the requirements of the system, as well as considering the skills possessed by the developer, it was decided to use an object-oriented approach.

The next decision came when deciding whether to implement the system using a bottom-up or a top-down approach. Due to the complexity of the final system it seemed logical to use the standard development practice for complex systems, which is the bottom-up approach. This technique works by breaking the system down into a number of independent sub-systems. These sub-systems are then further broken down into individual modules.

*Figure 5.1: The breakdown of the overall system into smaller, more manageable parts.*

The main benefits of this technique are twofold. Firstly, it allows the validity of each module within the system to be proven easily. This in turn makes proving the validity of the sub-systems simpler and ultimately makes the finished system easier to test. It also has the advantage of making error detection and highlighting easier. This is due to the tester having knowledge regarding whether the individual modules function as required before attempting testing on the sub-systems. The other main benefit of this approach is that it allows the functionality of the system to be extended in a relatively simple manner if planned correctly. This is because the system will be developed using an object-oriented approach. When new sub-systems (and modules) are required the new parts can be developed, tested, and then plugged in to the existing system.
5.2 System architecture design
Large systems are typically decomposed into smaller, more manageable sub-systems that provide the user with some related set of services. This section will deal with the initial design process of identifying these sub-systems, and then establishing a framework for the control and communication of the identified sub-systems. This will make the creation of an accurate description of the software architecture possible.

The most important step in this stage is the identification of the major underlying component of the system, and this is extremely easy to spot. Since the purpose of the system is to show the animation of algorithms, then the sorting algorithms must be the principal element of the system. This point is strengthened when it is realised that the other key components of the system are ascertained by the examination of the sorting algorithms that have been processed. These are the statistics that are generated when the sorting algorithms are run, the animation techniques that are associated with the various sorting algorithms, the descriptions of the sorting algorithms, and the components of the GUI that are geared towards manipulating the algorithm animation.

From this analysis it is clear that the sorting algorithms are the fundamental concept underpinning the entire system. It is also easy to see the three major sub-systems:

(i) The different views associated with the different sorting algorithms.
(ii) The processes provided to manipulate and display data generated by the sorting algorithms
(iii) The production of statistics and descriptions of the sorting algorithms.

This can be modelled as shown in the following figure.

*Figure 5.2: The high-level system design.*

Now that the basic system framework has been identified for the system architecture detail can be added to each of the sub-systems identified. Details of how each of these sub-systems has been implemented can be found in the detailed design and implementation section.

5.3 Animation design
The animation design section discusses the animation techniques that have been chosen for the system. These were decided by conducting research into which animation techniques were best for users in terms of understanding the sorting algorithms that were being animated.
5.3.1 The block animation
The block view is one of the original approaches to animating algorithms, and was used in the original ‘sorting out sorting’ [22] in 1981. It works by using an x-axis where the different elements are located at a position in the index, and a y-axis that shows the actual value of the element. Once the sorting process has been completed the user will be left with the shortest bar on the left (representing the element with the smallest value), and the longest bar on the right (representing the element with the largest value). The final, sorted set of elements looks as follows:

*Figure 5.3: The sorted elements in block view.*

5.3.2 The stack animation
The stack view consists of a stack of horizontal bars placed on top of each other. In this view the y-axis corresponds to the bars index, while the x-axis corresponds to the bars value. Once the sorting algorithm has placed each element in the correct position the user is left with the widest bar at the very bottom of the stack, with the thinnest bar at the very top. The final, sorted set of elements looks as follows:

*Figure 5.4: The sorted elements in stack view.*

5.4 User interface design
The user interface has been partially designed already from the prototyping conducted to determine an acceptable layout for the single animation screen. This has obviously determined the layout of the single animation screen, but has also provided feedback on what the user feels are acceptable colour schemes, as well as what is acceptable in terms of overall screen layouts. This prototype was deemed an appropriate basis for the final system because users from the target group have tested it, helped refine it, and then judged the final prototype to be acceptable on all the static HCI criteria specified.
Clearly there are numerous other screens within the system that need designing. At this stage in the design users have tested only the overall layout of the single animation screen. Therefore all other screens within the system could potentially change after they have been tested. It was felt that the most important screen in the system was the single animation screen as this is likely to be used most frequently; hence this has been prototyped while the others have not been due to time constraints.

5.4.1 The applet
All of the screens within the system will be located within the applet as this is the single heavyweight Swing component used in the system. The applet can be divided in to two major sections.

- Menu bar.
- Screen.

Figure 5.5: Applet layout with sub-components.

5.4.2 The menu bar
The menu bar will sit at the very top of the applet and will always be visible. It will allow the user to navigate the system, and will look as follows.

Figure 5.6: The menu bar container.

<table>
<thead>
<tr>
<th>Single Animation</th>
<th>Algorithm Race</th>
<th>Help</th>
<th>Algorithm Analysis</th>
<th>About</th>
<th>Quit</th>
</tr>
</thead>
</table>

It was decided to include a menu bar in the final design because the prototyping conducted discovered that if buttons were used for navigation they would take up far more space. This would lead to the interface being more cluttered with each element being harder to distinguish as it would have less space. Although only one of the eleven systems assessed in the system assessment survey used a menu virtually all modern applications use a menu for system navigation. This is due to the reasons stated previously, and also because it allows the system to group similar features together. For example, ‘cut’, ‘copy’ and ‘paste’ are all a similar type of operation, and are always in the same menu in Microsoft applications. Due to these reasons menus have been included in the final design.
5.4.3 Single animation setup screen

Figure 5.7: The single animation setup screen.

The single animation setup screen allows the user to define their desired properties for a single animation. A label at the top of the page asks the user to select the properties for the algorithm, and is included to give the user guidance about the purpose of this screen. Below this are four labels, each with a corresponding combo box. The user uses these to select their desired properties for the single animation. The contents of each combo box are determined by the requirements, with there being a combo box for each possible input parameter to the single animation screen. For example, the algorithm type combo box has the options ‘Bubble Sort, Quick Sort, Heap Sort, Selection Sort, and Insertion Sort’. At the bottom of this screen are two buttons. The first one is titled ‘start animation’. This will load the selected parameters into the single animation screen, as well as making the single animation screen visible and the current screen invisible. The other button is titled ‘cancel’. This button will return the user to the welcome screen. All of the parameters that may have been specified will be forgotten by the system if ‘cancel’ is pressed.

The single animation screen has been developed in this way for a number of reasons. Firstly, the label at the top and instructions below it keep the interface consistent as they give the user feedback on the section of the system they are in. This meets the feedback HCI principle stated in the HCI design principles section. The colour scheme used in this screen is also consistent with that used in the prototype. The reasoning behind this was that if the testers thought that the colour scheme worked in the prototype then there is no reason why users should not accept it as the colour scheme for the final system. Further, it is entirely visible to visually impaired users, thus meeting another of the requirements. The colour scheme itself is used system wide to ensure consistency. Finally, the combo boxes have been used because they are the best method of selecting between a choice of options without cluttering up the interface. They therefore allow the system to meet the requirement of interface clarity as well as the usability principle of minimalist design.
5.4.4 Algorithm race setup screen

Figure 5.8: The algorithm race setup screen.

The algorithm race setup screen is very similar to the single animation setup screen. As before, it has a title and descriptive text explaining the purpose of the screen to the user. It also uses combo boxes to allow the user to specify the parameters for the algorithm race. However, rather than having one label and combo box for the user to specify the algorithm type there are five.

This allows the user to choose five different sorting algorithms to watch simultaneously in an algorithm race. The sorting algorithms that can be chosen are specified in the requirements specification section. However, there is also the additional choice of none. This gives the user the chance to select less than five algorithms for a race if they desire.

The algorithm race setup screen is intentionally very similar to the previous screen. This was to ensure that the screen remains consistent with other screens in the system.

5.4.5 Algorithm analysis screen

Figure 5.9: The algorithm analysis screen.
The algorithm analysis screen enables the user to view a detailed description of each of the sorting algorithms. This description includes how the algorithm works as well as the average, worst, and best case performances of the algorithm. The user will also be able to see how the various algorithms perform over varying quantities of data by looking at a graph where the performance of each algorithm will be plotted. The screen itself has a label at the top informing the user of its purpose. Below this there is a combo box and button that allows the user to select the algorithm they wish to view. At the bottom of the screen on the left hand side is the ‘ok’ button. When the user presses this button they are taken back to the welcome screen. On the right hand side of the screen is the graph panel. The user can decide which algorithms they want plotting at any time by using the toggle boxes below the graph panel.

The algorithm analysis screen meets the requirements of consistent colouring and feedback. Its layout is different to the previous screens and draws heavily from the prototype. It has been split into two equal vertical sections because this approach proved a successful method in the prototype for grouping different types of information into different areas.

The information on the sorting algorithms is stored in strings, with there being a string containing the relevant text for each sorting algorithm. When the ‘display topic’ button is pressed the getSelectedIndex method is used upon the combo box to determine which item had been selected. An ‘if’ statement is then used to display the relevant string of text in the text area. When a new item is chosen the same mechanism works, which leads to the previous item being overwritten in the text area.

5.4.6 Help screen

Figure 5.10: The help screen.

The help screen enables the user to view help topics, which should answer any queries they have regarding the system. It also provides some information about sorting algorithms and the appropriate circumstances when they should, or should not be used. As with the other screens, it has a label at the top informing the user about the purpose of the screen. There is also a combo box that enables the user to select there desired help topic. Once the user has selected a topic the contents will be displayed in the text area. There is also a button at the bottom right hand side of the screen that takes the user back to the welcome screen. The help screen has been designed in this manner to ensure it is consistent with the rest of the system in terms of colouring, feedback and layout.
As with the algorithm analysis screen the actual topics that can be displayed are stored in strings, with there being a string containing the relevant text for each help topic. When the ‘display topic’ button is pressed the getSelectedIndex method is used upon the combo box to determine which item had been selected. An ‘if’ statement is then used to display the relevant string of text in the text area. When a new topic is chosen the same mechanism works, which leads to the previous help topic being overwritten in the text area.

5.4.7 About screen

The about screen is extremely simple. It provides information about the developer and the dates of development. The aim of this screen is to give any users who have queries that cannot be solved using the help facility the chance to contact the developer regarding these queries. The only section of the about screen that the user can interact with is the cancel button. This returns the user to the welcome screen. Finally, the about screen maintains the consistency of the system for the same reasons as the previous screens.

5.4.8 Welcome screen

The welcome screen is the first screen the user sees in the system. The screen welcomes the user to the system, and directs them to the menu bar at the top of the screen. The purpose of this screen is to try to guide the user towards the facilities that they wish to use. Although this will not be important for any user who has previous experience, it will be invaluable for a first-time user. This screen also acts as a portal to all of the other screens. It is the screen from which the user can access all other sub-sections of the system, and is the screen the user is taken to whenever they exit any of the other screens. As this is the first screen the user sees in the system it is especially important that it meets the HCI guidelines discussed previously. Following the design approach that was used for the previous screens has ensured this is the case.
5.4.9 Single animation screen

Figure 5.13: The single animation screen.

The single animation screen is identical to the final prototype developed. At the top is a label that states the purpose of the screen. Below this the screen is split vertically in two. On the left there is a label stating the algorithm currently being animated as well as the type of data being operating on. Below this is the animation panel, which is where the animations will be shown. Below this are the animation controls. These will allow the user to change the speed of animation, or to pause, or step-through the animation if they wish. Once pressed the run button will change its text to pause. If this button is pressed again the animation will be paused, and the text will change to run. Pressing this again will then resume the animation.

On the right hand side there is a text area at the top of the screen, which will correspond to the algorithm being animated. This text area will hold a description of how the sorting algorithm works, as well as its best, worst, and average case performances. Below the text area are a number of text boxes that will show statistics relating to the algorithm as it executes. Finally, at the very bottom is the colour key. This informs the user of what the colour of each item in the animation means.

The single animation screen has been kept identical to the prototype design because of the positive response the final prototype received from the testers. Due to this it was felt that no modifications were required to this screen. More information on this screen can be found in the prototype section of this document.
5.4.10 Algorithm race screen

Figure 5.14: The algorithm race screen.

The algorithm race screen is very similar to the single animation screen. However, this time the animation panel may be divided up into as many as five sections, depending on how many algorithms were specified in the animation race setup screen. There are also up to five sets of text box, again depending on the users selections. The only addition from the single animation screen (with the obvious exception of the additional text boxes and animation panels) is the rank label and corresponding text boxes. These remain blank until the corresponding animation has finished. Then they change to a rank value. This value will depend on how quickly the sorting algorithm finishes in comparison with the other sorting algorithms. The first to complete sorting will have a rank value of 1, with the last receiving a value of 5.

As stated above, the algorithm race screen is very similar to the single animation screen. The purpose for this is that it is known that the users have accepted the layout of the single animation screen, so by keeping the layout as close as possible to this it is expected that the users will also accept this layout. As with all the other screens, the colour and layout is consistent with the other screens in the system, which will ensure the user remains comfortable using the system.

5.5 The sorting components

The visual aspects of the system have been described in-depth, but for the system to function as desired a number of items have to be implemented to allow the animations to be displayed and manipulated.

5.5.1 The data

The logical point to start at in this section is the data that is to be sorted. The data needs to be generated based on the user parameters specified on the single animation setup screen or the animation race setup screen. The data itself will be stored in an array, with the array being filled using Java’s random number generating facility. However, firstly the getSelectedIndex method spoken of in the help and algorithm analysis screens will be used to determine which item has been selected on the combo box that governs the quantity of data to be used. An ‘if’ statement will then be used to ensure that the correct piece of code is ran to create the correct amount of elements for the array. The piece of code actually generating these elements will be in a ‘do’ loop.
that will ensure that random elements are continuously generated until the array is filled with the correct amount of elements.

Once the random data array(s) has been created it may need to be sorted into a particular order prior to the actual starting of the sorting algorithm(s). Whether this is necessary will be determined by using the getSelectedIndex method once more, but this time it will be used on the combo box that allows the user to specify the order of the data prior to sorting. An ‘if’ statement will again be used to determine what piece of code is to be ran based on the users input. If the data order selected is random then no actions will be performed on the data. If the data order is anything else then clearly the data will need to be sorted in to the relevant order. Three segments of code do this and can be found in appendix d.

5.5.2 Sorting
The sorting of the data in the arrays is clearly enough critical part of the system. Before sorting begins though the sorting technique(s) to be used must be ascertained. This will again be done using the getSelectedIndex method. This will look at the combo box that the user uses to specify the sorting technique to be used. An ‘if’ statement will then determine which piece of code is to be ran. Once this has been decided on nothing will happen until the user selects the ‘run’ button that indicates the start of sorting. When this happens the appropriate sorting technique will be applied to the data to sort it in to the correct order. Obviously there are five such techniques with each using a different method to sort the data. Once sorting has been completed a setText method will be used to change the contents of the status text box (that displays the current state of the sorting process) to ‘sorting completed’.

5.5.3 Algorithm Race
Java is well suited to supporting an algorithm race as it provides threads, which support parallel processes. Therefore in the algorithm race each sorting algorithm will be allocated its own thread. This will allow the sorting algorithms to race each other, as each should have an equal share of the processor. However, threads can never be assigned perfectly equal amounts of processor time unless there are multiple processors in the users machine, and since this is not going to be the case for the vast majority of occasions a mechanism needs to be put in place to deal with this. The technique decided upon for the system will be to allocate each thread a set amount of processor time to perform operations before the it has to pause and let the next thread perform some operations. Due to the speed of processors a time of 5 milliseconds will be allocated to each thread while it is operating. It must then yield and let the next thread have its allotted time. The order in which the threads will have access to the processor will be randomly decided at the start of sorting, and the threads will then stay in this order until they have finished sorting. Another approach that was considered was to allow each thread to perform one operation, before pausing to let the next thread access the processor. However, this approach was deemed to be less controlled as different algorithms would take different amounts of time to perform a single operation. Therefore this approach was discounted.

5.5.4 Animation Controls
The animation controls are an important part of the system as they allow the user to manipulate the animation. The ‘run’ button will start the animation. It will work by calling the relevant sorting method, which will then sort the data in to the correct order and display it on the screen. The ‘cancel’ button takes the user back to the welcome screen. This will work by setting all the current screens components to invisible using the setVisible method. It will then call a method, which will draw all of the welcome screens components to the screen. The ‘faster’ and ‘slower’ buttons will work by incrementing or decrementing a pause variables value. The pause variable will be placed in each sorting algorithm after the point in the code where a swap occurs. Modifying this value will clearly change the speed at which the sorting algorithm executes. The speed bar will also allow the user to modify the value of the pause variable. However, it will
allow the user to change the speed much more rapidly than using the faster or slower buttons. Finally, the ‘walkthrough’ button will work by executing the sorting algorithm until a swap occurs. When a swap does occur it will stop executing until the user presses the ‘walkthrough’ button again.

5.6 A summary of the design
This section has presented the high-level system design. Each section of the GUI has been presented and discussed. The developer has attempted to make all the screens that make up the system as similar as possible in terms of layout and colour usage. This was because this should lead to enhanced usability for the final system. The animation techniques that will be used in the system have also been described in detail. These techniques were chosen by assessing the effectiveness of animation techniques in existing systems. The next section in this document will describe the implementation in detail, focusing on the more complicated parts of the system.

6.0 Detailed Design and Implementation

6.1 Detailed design considerations
As stated previously it was necessary for a programming language to be used that could support Internet deployment for the system, as well as providing the developer with the ability to build a GUI. These reasons, coupled with the developer’s personal experience, made Java the logical choice. Once this decision had been made the developer had to decide which development kit to use to construct the GUI. The two options were AWT or Swing. AWT was the original toolkit for Java GUI programming but suffered from a number of usability issues. It does not have the ability to render graphics on the physical display device of any platform. AWT components also use a considerable amount of memory combined with their native peer objects. Swing extended AWT and addresses these concerns. It has only four ‘heavyweight’ components with native peers, with all other Swing components being known as ‘lightweight’. These ‘lightweight’ components are rendered by one the ‘heavyweight’ components. This is of direct contrast with the AWT toolkit where all components are ‘heavyweight’, and are rendered by the native windowing system. These ‘lightweight’ components are an advantage as they use far less memory than their ‘heavyweight’ counterparts. Swing also has a pluggable ‘look and feel’. This allows the programmer to use different look and feels as they see fit. For example, this would allow a Java program running in Unix to use the standard Windows colour scheme.

6.2 The Graphical User Interface
The GUI consists of the following screens:

- Welcome screen.
- Quit screen.
- About screen.
- Algorithm analysis screen.
- Help screen.
- Algorithm race setup screen.
- Single animation setup screen.
- Algorithm race screen.
- Single animation screen.

The screens of the GUI were the first items of the system developed. This was because they needed to be in place before more complex sections of the system (such as the single animation
screen animation) could be developed. For example, the single animation setup screen had to be developed before the single animation screen because the single animation screen required input from the single animation setup screen for the animations to be setup correctly.

### 6.2.1 NetBeans IDE

The screens making up the GUI were developed using NetBeans IDE, V3.51. This was an excellent program, as it allowed the developer to build the screens extremely quickly and efficiently. It allowed the developer to specify the ‘heavyweight’ top-level container (in this case an applet), and then allowed ‘lightweight’ components to be added to this by selecting the desired component, and dragging the desired outline of the component on to the applet. Although there were numerous instances where the components did not appear on the interface exactly where the developer desired, a fine-tuning layout facility allowed the majority of these issues to be corrected.

*Figure 6.1: NetBeans ‘form editor’ view showing the single animation screen at an early stage in the development process.*

![NetBeans form editor](image)

*Figure 6.2: The layout designer within NetBeans.*

### 6.2.2 Events and event handling

For the system to evolve from being a static collection of screens to a system consisting of screens the user could move between using the mouse, events and event-handling had to be implemented. Events are caused by actions that occur in a system and can range from a mouse button being clicked, to a specific button being pressed on the keyboard. Event ‘listeners’ can be
placed on components to detect when a certain event has occurred. For example, if a listener is placed upon a button to listen for a mouse click it will detect every time that button is clicked. When this action, or event is detected a piece of code can be ran to do some task. This is the mechanism that was used to create a dynamic system. All buttons and menu items had event listeners placed on them, and then the code associated with these listeners’ moves the users between screens as appropriate.

Figure 6.3: How a mouse listener is added to a menu item. Whenever this menu item is clicked by the mouse the listener will detect it.

```java
welcomeMenu7.addActionListener(new java.awt.event.ActionEvent(){
    public void actionPerformed(java.awt.event.ActionEvent e) {
        welcomeMenu7.setVisible(false);
        welcomeMenu7.setEnabled(false);
        welcomeMenu2.setVisible(false);
        welcomeMenu2.setEnabled(false);
        welcomeMenu3.setVisible(false);
        welcomeMenu3.setEnabled(false);
        welcomeMenu4.setVisible(false);
        welcomeMenu4.setEnabled(false);
        welcomeMenu5.setVisible(false);
        welcomeMenu5.setEnabled(false);
    }
});
```

Figure 6.4: Code that is executed when a mouse click is detected. This hides the welcome screen and displays the algorithm race setup screen.

6.3 The setup screens

6.3.1 The single animation setup screen

As with the other screens event handlers were crucial for this screen. Event handlers were placed on the ‘start animation’ and ‘cancel’ buttons to allow the user to move to the desired screen. However, for this screen the event handlers were not just for navigation purposes. This time they also had to record the users chosen parameters from the combo boxes in instance variables. These instance variables could then be used in the single animation screen to specify how the screen was to be setup. Each combo box represented a parameter the user could specify, and each had a corresponding instance variable.

Figure 6.5: The code for the data type combo box. This allows any of the required types of data to be selected by the user.

```java
scrComboBox3.setModel(new javax.swing.DefaultComboBoxModel(new String[]{
    "Random",
    "Reverse sorted",
    "Partially sorted",
    "Sorted"}));
```
Figure 6.6: The start button event handler. This records the value of each combo box so that it can be used in the single animation screen.

```java
private void sranBtnButto1MouseClicked(java.awt.event.MouseEvent evt) {
    // Code implementation...
}
```

6.3.2 The algorithm race setup screen
The algorithm race setup screen was very similar to the single animation setup screen in terms of its layout and the way it was developed. It used an identical technique to transfer the values specified in the algorithm race setup screen to the algorithm race screen, the only difference being that more values had to be recorded, as there were more combo boxes on the algorithm race setup screen. A decision was also taken during implementation that the minimum number of algorithms that could be selected for the algorithm race was two. This decision was made because it was felt that if the user wished to run a single animation they could use the single animation screen. Therefore the first two combo boxes (for selecting the algorithms to be animated) had no ‘none’ option, whereas the remaining three did have the ‘none’ option. This allowed the user to specify any quantity from two to five for the number of algorithms to be ran in a race.

6.4 The sorting screens
The sorting screens are the key components within the system. Both have identical core functionality, while the algorithm race screen offers additional functionality to support the algorithm race.

6.4.1 The single animation screen
The main underlying component of the single animation screen is the data being sorted. However, the data has to be set up to meet the users requirements (specified in the single animation setup screen) before it can be sorted. This is done by looking at the values of the instance variables that were set by the single animation setup screen. The system then constructs an array of data with the required amount of elements with each elements value being generated randomly. This array is then sorted to the users desired order if necessary (e.g. reverse sorted, pre-sorted, or partially sorted). If the user has selected ‘random’ no sorting is done on the data before the animation is started.

Figure 6.7: Filling the array with random values.

```java
elements = new int[size];
Random generator = new Random();
int counter = 0;
do {
    elements[counter] = generator.nextInt(100);
    counter ++;
}while (counter < size);
```
Once the array is in the correct order it can be rendered to the screen. Due to the developer being unfamiliar with the graphical drawing package in Swing an unconventional approach was used. Firstly, a panel was placed in the desired area of the screen where the animation was to be shown. This was the animation panel. Next, a layout manager was specified for the panel. The layout manager used was always the grid layout manager, but the layout varied depending on the view type and the number of elements in the array. For example, if the view was of type stack then the grid would always be one element wide, and as many elements high as the number of elements in the array. If the view were of type block then these dimensions would be swapped. Once the panel was in place a sub-panel was placed in each grid location. A text area component was then placed in to each of the sub-panels to represent an element in the array. These text areas were all non-editable, so that they appeared simply as white blocks in a stack or block of data. The height or width of these text areas could be modified to represent different data values. For example, if there were five elements in the array and the view was of type stack, then using the setColumns method on the text areas would allow the different values of the elements to be represented. Although this approach to rendering is not ideal it does meet the user requirements so is acceptable.
Figure 6.9: This shows how the initial data is rendered before sorting begins when block view is being used. First of all the layout is set, then the setRows method is used to set the desired value of the element. Due to size constraints all elements had to be divided by five to be sure they would fit into the panel. However, this led to the possibility that an element could have a value of zero. For example, if the elements value were 1/5 it would equal 0.2, which would be rounded down to zero. This was not desirable so an 'if' statement was used to catch any values of zero and set them to a value of one. Finally, the two lines at the bottom of the screenshot show how the text area is added to the sub panel, which is then added to the animation panel.

```java
GridBagConstraints grid = g;
GridLayout layout = new GridLayout(1,5);
twojPanelNew.setLayout(layout);

int count = 0;
int value = (elements[count]) / 5;
if (value < 1) {
    value = 1;
}
twojTextAreaNew1.setRow(value);
twojTextAreaNew1.setColumns(1);
twojPanelstack1.add(twojTextAreaNew1);
twojPanelNew.add(twojPanelstack1);
```

At this point elements within the array are normally not in the correct order so the corresponding text areas have their background colour set to white to indicate this. However, any data elements that are in the correct position (e.g. if the data is pre-sorted, or partially sorted then at least some of the elements are sorted in to the correct location) have their corresponding text areas background colours set to black, to indicate this.

Figure 6.10: Setting the background colour of text areas to black.

```java
twojTextAreaNew1.setBackground(new java.awt.Color(0, 0, 0));
twojTextAreaNew2.setBackground(new java.awt.Color(0, 0, 0));
twojTextAreaNew3.setBackground(new java.awt.Color(0, 0, 0));
twojTextAreaNew4.setBackground(new java.awt.Color(0, 0, 0));
twojTextAreaNew5.setBackground(new java.awt.Color(0, 0, 0));
```

Once the array is correctly rendered to screen the animation can be started. When this happens the animation panel must be updated each time items change location in the array. This is achieved by calling the relevant method that will render the current state of the array to the animation panel. This is done by placing a call statement in the sorting algorithm. This calls the relevant rendering method when items change location in the array. The background colour of the element that has moved in to the correct location is changed to black at this point.

The other elements in the single animation screen that require updating once the animation has finished are the text areas that show how many comparisons and swaps have taken place, the current status of the sorting algorithm, and the length of time that sorting took. The values for swaps and comparisons are recorded by setting counters for each to zero before sorting begins. Whenever a swap or comparisons occurs during sorting the relevant counter is incremented. When the screen is updated the latest values are placed in the relevant text areas. The time for the algorithm to be sorted is calculated by taking the time in milliseconds before the algorithms starts, and then taking it after the algorithm finishes. By subtracting the first from the second the total sorting time can be calculated. This value is placed in the relevant text area upon completion of
the animation. Finally, the animation status is set to complete when the sorting algorithm is completed by changing the text in the relevant text area.

Figure 6.11: Code that calculates the number of swaps and comparisons for bubble sort, as well as showing the calculating of the start and finish times for the sorting.

```java
long startTime = new GregorianCalendar().getTimeInMillis();
while (doSort) {
    doSort = false;
    for (int bub = 0; bub < elements.length - 1; bub++) {
        comparisons ++;
        if (elements[bub] > elements[bub + 1]) {
            bubble = elements[bub];
            elements[bub] = elements[bub + 1];
            elements[bub + 1] = bubble;
            doSort = true;
            swaps ++;
        }
    }
}
long finishTime = new GregorianCalendar().getTimeInMillis();
```

6.4.2 The algorithm race screen
The algorithm race operates similarly to the single animation screen in terms of setting up the arrays of data in to the desired order, and rendering them to the screen. The only real difference is that the developer had to ensure that the arrays were each rendered to a different panel within the animation panel so that all had sufficient space to be clearly seen. This was done by having as many animation panels as there were number of animations. This meant that the methodology described for the single animation screen would work for the algorithm race screen for rendering and updating the arrays during sorting.

One of the other differences between the two screens was the ranking text boxes. These were used to display the order the sorting algorithms had finished in. As was seen previously the time taken to complete the sorting was calculated for each algorithm. These values were all compared, with rank values being determined based on time comparisons. These ranking were then placed in the relevant text boxes. If any sorting algorithms finished in identical times then they were given the same ranking.

The final difference was that up to five algorithms had to be run simultaneously in an algorithm race. The obvious way to do this was by using threads, but due to the developer’s inexperience with threads as well as time constraints this has not yet been achieved. Instead, the system runs the sorting algorithms consecutively. For example, the first sorting algorithm will be run by calling the relevant method, and upon its completion the second will be run. This will continue until all of the sorting algorithms have been run. Therefore the system does not run the sorting algorithms in parallel as described in the design section, meaning that the algorithm race provided actually shows the sorting algorithms one by one. Clearly this is not ideal, as it does not meet the user requirements specified.

6.5 The sorting algorithms
The requirements section states that five sorting algorithms have to be implemented. These were the following:

- Bubble Sort.
- Quick Sort.
- Heap Sort.
6.5.1 Bubble Sort

The Bubble Sort code calculates the number of swaps and comparisons that occur by incrementing counters in the relevant places. Clearly it also sorts the data.

Figure 6.12: Bubble Sort code.

```java
while (doMore) {
    doMore = false;
    for (int i = 0; i < elements.length - 1; i++) {
        comparisons++;
        if (elements[i] > elements[i + 1]) {
            int temp = elements[i];
            elements[i] = elements[i + 1];
            elements[i + 1] = temp;
            doMore = true;
            swaps++;
        }
    }
}
```

6.5.2 Insertion Sort

As with Bubble Sort the Insertion Sort code includes the calculation of the number of swaps and comparisons that have occurred during the sort.

Figure 6.13: Insertion Sort.

```java
for (int i = 1; i < elements.length; i++) {
    int B = elements[i];
    comparisons++;
    while (i > 0 && elements[i-1] > B) {
        comparisons++;
        elements[j] = elements[j-1];
        j--;
    }
    elements[j] = B;
    swaps++;
}
```
6.5.3 Selection Sort

Figure 6.14: As with the two previous sorting algorithms Selection Sort has counters placed in the sorting code to calculate the number of swaps and comparisons that have taken place.

```c
for ( seli = 0; seli < elements.length; seli ++ ) {
    for ( selj = seli + 1; selj < elements.length; selj ++ ) {
        comparisons ++;
        if (elements[seli] > elements[selj]) {
            temp = elements[seli];
            elements[seli] = elements[selj];
            elements[selj] = temp;
            swaps ++;
        }
    }
}
```

6.5.4 Quick Sort

Figure 6.15: Quick Sort is more complex than the previous algorithms. It calls itself recursively until it is fully sorted. Again it calculates the number of swaps and comparisons that have taken place.

```c
if (lo > hi) {
    return;
}
else if (lo == hi - 1) {
    comparisons ++;
    if (elements[lo] > elements[hi]) {
        int temp = elements[lo];
        elements[lo] = elements[hi];
        elements[hi] = temp;
        swaps ++;
    }
    return;
}
int pivot = elements[(lo + hi) / 2];
elements[(lo + hi) / 2] = elements[hi];
elements[hi] = pivot;
comparisons ++;
while(lo < hi) {
    comparisons ++;
    while(elements[lo] <= pivot) && (lo < hi)) {
        lo ++;
        comparisons ++;
    }
    while((pivot <= elements[hi]) && (lo < hi)) {
        hi --;
        comparisons ++;
    }
if (lo < hi) {
    int temp = elements[lo];
    elements[lo] = elements[hi];
    elements[hi] = temp;
    swaps ++;
    comparisons ++;
}
elements[lo0] = elements[hi];
elements[hi] = pivot;
qsort(elements, lo0, lo - lo0);
qsort(elements, hi + 1, hi0);
```
6.5.5 Heap Sort

Figure 6.16: Heap Sort is another of the more complex sorting algorithms. It keeps track of the number of swaps and comparisons that have occurred in the same manner as the previous algorithms.

```java
for (int k = length/2; k > 0; k--) {
    downheap(elements, k, length);
}

do {
    int temp = elements[0];
    elements[0] = elements[length - 1];
    elements[length - 1] = temp;
    length = length - 1;
    downheap(elements, 1, length);
} while (length > 1);

public void downheap(int []elements, int k, int length) {
    int temp = elements[k-1];
    comparisons++;
    while (2 * k <= length/2) {
        comparisons++;
        int j = 2 * k;
        if ((j < length) && (elements[j-1] < elements[j])) {
            j ++;
        }
        if (temp >= elements[j-1]) {
            break;
        } else {
            elements[k-1] = elements[j-1];
            k = j;
            swaps++;
        }
    }
    elements[k-1] = temp;
}
```

6.5.6 Sorting

The sorting algorithms methods were called depending on what had been selected by the user. For example, if the user had selected Bubble and Quick Sort for the algorithm race then these two techniques would be called, one for each array of data. Upon completion of sorting the technique would call the relevant display method that would then render the data to the screen. A number of different methods were included to display data. For the single animation screen there were methods that would handle rendering the data for five, ten, twenty and fifty elements of data in block and stack view. An example of several of these can be found in appendix d.

6.6 The data array

The data arrays were implemented as described in the design section. This meant that the random number generator provided by Java was used to fill the array with the required number of elements. The random number generator used was that provided by the ‘Random’ class provided by Java. Although the numbers it produced were only pseudo-random they were sufficiently random for this system. Had entirely random numbers been required the developer would have used ‘SecureRandom’ class provided by Java, and then performed operations on the values generated to ensure they really were random.
The data arrays were pre-sorted when required in the manner described in the design section. This meant that the getSelectedIndex method was used in collaboration with an ‘if’ statement to determine which piece of code would be ran to pre-sort the data if it was necessary. Information on this can be found in appendix d.

6.7 Deploying the system on the Internet
For the system to be usable via the Internet as specified in the requirements it had to be deployed on the Internet. This was achieved by placing the completed applet inside a web page on the users university web site. This can be found at [41].

6.8 Functionality removed from initial design
A number of features were removed from the design as it either became apparent that they would not enhance the usability of the system, time constraints meant there was not time to implement them, or they simply were not relevant. Firstly, the initial design for the welcome screen contained an ok button. When it was decided that the welcome screen was to act as the main portal to the system, rather than simply a pop up screen that would only appear the first time the user loaded the system up, the need for this button was removed. The colour scheme selection screen was also removed. This was because this feature offered nothing in terms of usability to the system, as the colour scheme used were deemed acceptable from the prototyping feedback, and the colour scheme was acceptable to visually impaired users. An aspect that affected usability slightly was the removal of the graphs on the algorithm analysis page. The developer simply did not have sufficient time to implement this feature, so was forced to leave it out of the final system.

6.9 The implementation strategy
Before any of the more complex aspects of the system could be developed the basic system framework had to be put in place. This meant that the screens with all of the necessary event handlers were developed first. Once this was done the more complex sections of the system could be worked on. Although this approach was not ideal (it led to less time being allotted to the development of the core functionality of the system) it was the only realistic approach as this core functionality had to be built on top of the screens, as these screens had to feed user data in to the sorting screens before they could function.

Whilst the system was being implemented it was continuously tested. Once a component had been added to the system the component was checked to see that it worked as desired, and appeared where the developer wished on the screen. When components relied on each other to function correctly they were both checked whenever either was changed. This was to check that all of the components still worked. This approach meant that by the time the system reached the testing section of the project the majority of tests had already been passed, as the majority of errors had been discovered and fixed during implementation.

6.10 Summary of the detailed design and implementation process
The system was implemented using Java and the Swing windowing toolkit. The first section of the system to be developed was the GUI and then the event handling mechanism that allowed the users to move between different sections of the system. The more complex sections were implemented on top of this framework. Between them the screens allowed the user to not only watch animations of sorting algorithms (either singularly or in an algorithm race), but also to examine detailed descriptions of the methodologies and efficiencies of the five sorting algorithms included in the system. Help topics are also available that explain how to use the system. Upon completion the system was deployed on the users university web space, thus meeting the requirement that the system be web-based.
7.0 Testing

7.1 Testing introduction
Systems should not be tested as a single, monolithic unit. This is because the majority of systems are built out of sub-systems that are built out of modules, which in turn are composed of procedures and functions. The testing process should therefore proceed in stages where testing is carried out incrementally in conjunction with system implementation.

For these reasons a testing process composed of five stages has been adopted. This tests system components first, then the integrated system, and finally the system is tested against real users. Ideally, component defects are discovered early in the process and interface problems are discovered when the sub-systems are integrated. However, as defects with the system are discovered the program must be debugged and this may require other stages in the testing process to be repeated. For example, errors in system components may become apparent during integration testing. This means the process is an iterative one, with information being fed back from later stages of the process to earlier stages.

The testing process itself consists of the following five stages, which were conducted in the order shown below:

1. Unit testing – individual components are tested to ensure that they operate correctly. Each component is tested independently, without other system components. For example, the ‘cancel’ button on the algorithm analysis screen should take the user back to the welcome screen, and the quit menu item should take the user to the quit confirmation screen.

2. Module testing – a module is a collection of dependent components such as an object class or an abstract data type. A module encapsulates related components, so can be tested without other system modules. An example of this is the animation controls. In this instance module testing should ensure that the speed bar modifies the speed of the animation.

3. Sub-system testing – this phase involves testing collections of modules which have been integrated into sub-systems. Since the most common problems that arise in large software systems are interface mismatches the sub-system test process should concentrate on the detection of module interface errors by rigorously exercising these interfaces. An example of this is the single animation screen. All the components on this screen should be synchronized to work in tandem to support the users learning of the sorting algorithm being animated.

4. System testing – the sub-systems are integrated to make up the system. This process is concerned with finding errors that result from unanticipated interactions between sub-systems and sub-system interface problems. It is also concerned with validating that the system meets its functional and non-functional requirements and testing the emergent system properties.

5. Acceptance testing – This is the final stage in the testing process before the system is accepted for operational use. The system is tested by actual users rather than by the developer. Acceptance testing may reveal errors and omissions in the system requirements definition because the real users are likely to exercise the system in different ways to the developer. Acceptance testing may reveal requirements problems where the system’s facilities do not really meet the users needs or the systems performance is unacceptable.
Whenever a defect was discovered in the process it was corrected if possible. All sections of the test process that were influenced by the faulty component then had to be re-tested to ensure that they still worked correctly after the modifications had been made. Although this was a time consuming process it did help to remove numerous errors from the system and ensure that no new errors were added.

It is possible that some unknown defects remain in the system despite the comprehensive testing conducted. This is due to the time constraints placed on the developer. Any unknown defects remaining in the system are likely to be insignificant though as the system has passed acceptance testing and is therefore considered usable by end users.

Due to time constraints the animations controls that will allow the user to speed up, slow down, and walk through animations were excluded from the final system. If this proves to lead to significant usability issues with the system an attempt will be made to implement these features.

7.2 Emergent properties

Emergent properties of a system are attributes of the system as a whole. They can only be measured once the sub-systems have been integrated to form the complete system. In this system the one major emergent property that requires testing is usability. As stated previously in the requirements specification section usability can be broken down into the following sub-sections:

- The final system should be an effective teaching tool that will easily allow students to understand all they need to know about the sorting algorithms implemented.
- The system will have a logical and intuitive layout of controls.
- Colour will be used in the GUI to benefit the user. This will also follow the guidelines described above about the use of colour in interfaces. In addition, the default colour scheme will be developed for colour-blind users.
- The System will be reliable. No unexpected events will startle the user, and the system will not crash more than an acceptable percentage of the time.

The system will be tested against these factors as much as possible. However, such is the nature of emergent properties that only prolonged use by a large number of users can be guaranteed to find all the emergent property defects in the system. Therefore it is feasible that some defects that have a low probability of occurring will be missed.

7.3 Black box and white box testing

White box testing is concerned only with testing the software product; it cannot guarantee that the complete specification has been implemented. Black box testing is concerned only with testing the specification; it cannot guarantee that all parts of the implementation have been tested. Thus black box testing is testing against the specification and will discover faults of omission, indicating that part of the specification has not been fulfilled. White box testing is testing against the implementation and will discover faults of commission, indicating that part of the implementation is faulty. In order to fully test a software product both black and white box testing are required.

However, white box testing is much more expensive than black box testing. It requires the source code to be produced before the tests can be planned and is much more laborious in the determination of suitable input data and the determination of whether the software is correct. Black box testing can commence as soon as a component has been completed, while white box testing cannot start until all of the black box tests have been successfully passed. White box
testing should commence with the production of flow graphs and determination of paths. The paths should then be checked against the black box test plan and any additional required test runs determined and applied.

Due to time constraints only black box testing will be conducted on the system. Although this is not ideal, testing the entire system with the black box method should minimize the number of faults that will be missed as a result of not testing comprehensively using the white box technique.

7.4 Acceptance testing
Although the testing spoken of previously is thorough it only covers one aspect of the system – the system functionality, through black box testing. As stated numerous times earlier in this document the usability of the system is equally important to the functionality in creating a successful system. Therefore comprehensive usability testing must also be conducted to ensure the system is acceptable to the target user group.

The actual method used will be a query based empirical approach. The system assessment surveys used earlier in the requirements analysis section will be given to people in the target user group to evaluate the system against all of the usability issues previously raised. This will allow each usability aspect of the GUI to be given a rating. If the average ratings are sufficiently good the system will have passed acceptance testing. If the scores are not sufficiently good the system will have to be modified to improve these unacceptable aspects. Therefore this process is clearly iterative. However, due to the time constraints there will be a limited number of iterations that can occur. A total of twenty users will be asked to perform the evaluation. It is felt that this is a sufficiently large sample to give a true reflection on how usable this system is to the average user in the identified target user group. The twenty testers will all come from the target user group.

If more time had been available additional methods would have been considered such as experimental, or observational approaches to empirical testing. However, these techniques typically cost much more than the query technique in terms of time so are therefore not feasible for this project.

7.5 The test machine
All of the tests were conducted on the developer’s computer of the following specification:

- 2.4 GHz AMD Athlon processor.
- 512 MB RAM.
- 1024 by 768 pixels screen resolution, with 32-bit colour.
- Java 2 SDK (Standard Development Kit) 1.4.2.
- Sun JRE (Java Runtime Environment) 1.4.2.

7.6 The test plan
The test plan includes a program specification, a list of the particular inputs that will be tested, the outputs expected, and the actual outputs from the system. Details of the test plan itself can be found in appendix b.
7.7 Test results

7.7.1 Black box testing

The welcome screen
The welcome screen has the sole purpose of acting as a portal that allows the users to access all other sections of the system from a single familiar point. The tests conducted on this screen were based around the links in the menu bar at the top of this screen that allowed the user to move to other screens within the system. The system passed on all the specified tests as the user could move to all the required sections of the system by pressing the relevant menu item. The test results for this section are displayed in appendix b.

The quit screen
The quit screen has the purpose of allowing the user to exit the system in a controlled fashion. Two tests were conducted on this screen. The first checked that the user could exit the system successfully, while the second checked that the user could return to the welcome screen if they decided they did not wish to quit the system. The system passed both of these tests successfully. The test results for this section are displayed in appendix b.

The about screen
The about screen gives the user brief information on the system. The only test performed on this screen was to check that the user could successfully return to the welcome screen. The system successful passed this test. The test results for this section are displayed in appendix b.

The algorithm analysis screen
The algorithm analysis screen allows the user to see detailed descriptions of the algorithms included within the system. The tests conducted on this screen checked that the relevant information was shown in the text area when an element of the combo box was selected. The system passed all of these tests successfully, with all of the appropriate details being displayed in the text area once the display button was pressed. As with the other screens the user also had to be able to return to the welcome screen from this screen. The system passed this test successfully. The test results for this section are displayed in appendix b.

The help screen
The help screen allows the user to access instructions on how to use the system. As with the algorithm analysis screen the tests checked that the appropriate information was shown in the text area based on the item selected in the combo box. The system passed this test successfully, and also passed the test that checked it linked successfully to the welcome screen. The test results for this section are displayed in appendix b.

The single animation setup screen
The single animation setup screen is important as it allows the user to specify the parameters they wish the single animation to use when it is set up. This screen was tested to check that all the parameters specified by the user were successfully shown in the single animation screen. This included having the data represented in the view desired, having the desired number of elements to be sorted, having the data being in the desired format, and the desired sorting algorithm working on the data once the algorithm is started. The description text area in the single animation screen was also tested to check that it displayed details of how the selected sorting algorithm works. The system passed all of these tests successfully, meaning that the single animation setup screen could be relied on to set up the user’s desired algorithm animation using thee specified parameters. The system also passed the standard test of being able to successfully
navigate back to the welcome screen if the user decided they did not wish to set up a single animation. The test results for this section are displayed in appendix b.

The single animation screen
The single animation screen allows the user to watch an animation of a single array of data being sorted. This screen was tested to check that the data was sorted correctly by all sorting algorithms, that the text boxes updated appropriately to indicate the correct number of swaps and comparisons, that the time elapsed text box showed the duration sorting had taken, that the sorting status text box changed to indicate when sorting had been completed, and that the colour of the data elements changed once they were in the correct positions in the array to show that they had been sorted. All of these tests were passed successfully. However, there was a slight problem with the time elapsed text box. Due to the speed of the test machine the tests almost always took 0 milliseconds, hence this text box displayed ‘0’ as the time taken. This was not a fault with the system, it was caused due to the speed of the test machine. When a pause statement was added to check the validity of the code the value of the text box increased proving that the text box correctly calculates the time taken to sort the data. The ‘run’ button was also tested, as this was the button that actually started the animation race. Finally, the screen was also tested to check that the user could successfully reach the welcome screen. The test results for this section are displayed in appendix b.

The animation race setup screen
The animation race setup screen does an almost identical job to that of the single animation setup screen, except that it allows the user to specify between two and five sorting algorithms to operate simultaneously on different arrays of data. As with the tests for the single animation screen the tests focused principally around ensuring that the parameters the user specifies are correctly set up in the animation race screen. As before, tests were conducted to check that the data was set up in the desired view, that the number of elements to be sorted corresponded to the number specified and that the data type corresponded to the type specified. In addition, the system was tested to check that the desired quantity of sets of data was displayed, as determined by the user. The description text area was also tested to ensure that descriptions for all of the sorting algorithms that were going to operate were shown. All of these tests were passed, as was the standard test of checking that the user could return to the welcome screen from the current screen. The test results for this section are displayed in appendix b.

The animation race screen
Several of the tests conducted on this screen were the same as for the single animation screen. These included checking that the data was sorted correctly by each of the sorting algorithms, that the number of swaps and comparisons text boxes were updates correctly to the correct values, that the elements changed colour once they had been correctly ordered to indicate that they were in the correct place. In addition, the rank text box was tested to check that it correctly placed the animations in chronological order once they had finished sorting. The ‘run’ button was also tested, as this was what actually started the animation race. Finally, the ‘cancel’ button was tested to ensure that it took the user back to the welcome screen once pressed. All of these tests were successfully passed. The test results for this section are displayed in appendix b.

The animation controls
The animation controls were designed to allow the user to control the speed of the animations on execution, as well as allowing a walkthrough feature that slowed the animation speed down so that the user could step through each individual step of the sorting algorithm. These animation controls were designed to work on both the single and algorithm race screens. However, due to time constraints these features have not been successfully implemented, thus these tests have failed.
7.7.2 Black box testing screenshots

Figure 7.1: The first screenshot shows the algorithm analysis screen before a topic has been selected for display. The second screenshot shows that the correct topic has been displayed in the text area after a topic has been selected and the ‘display’ button has been pressed, with no other changes to the page occurring.

![Algorithm Analysis](image1)

Figure 7.2: This shows the help screen before (left) and after (right) a topic has been selected for display.

![Help Selection](image2)

Figure 7.3: The algorithm race screen before (left) and after (right) the ‘run’ button has been pressed to start the race itself. The data shown is in the stack view.

![Algorithm Analysis](image3)
Figure 7.4: Shows an algorithm race before (left) and after (right), with the data being viewed in the block format, and the data being set up in reverse order.

Figure 7.5: Shows data set up in reverse order in the block view of a single animation (left). The sorted data is also displayed (right).

Figure 7.6: This shows the single animation screen in stack view with partially sorted data before the sorting algorithm has run (left) and after (right).
7.7.3 Black box testing conclusion
With the exception of the animation controls, the system has been proved to meet the requirements specified previously. The user can navigate between all pages of the system, detailed information can be viewed on all the sorting algorithms contained in the system, and step through guides can be accessed to tell the user how to go about some of the most commonly performed tasks within the system. More importantly the system was proven to allow the user to specify the parameters for both a single animation, and an animation race, before running these animations. When the animations were executed the data was successfully sorted in to order using the specified sorting technique(s), and the various text boxes were updated to show the statistics generated by the sorting algorithms. The only area in which the system has not met its functional requirements is that of allowing the user to control the speed of the animation. How much of a hindrance to the usability of the system this will be, will be determined by the users in acceptance testing.

7.7.4 Acceptance testing results
The system was tested against the same criteria that were used to assess existing systems for the requirements analysis section. This meant that two aspects of the system were focused on. Firstly, the actual usability of the system was tested, and secondly, the ease of learning.

Acceptance testing – Human Computer Interaction factors
As can be seen in the graph below the system scores well on all aspects of usability. The scores range from ‘menu paths’ (1.2 average) to ‘layout’ (2.05 average). These scores prove that the system is easy to use for users within the system’s target group. The main benchmark is the average system score. The developed system scored excellently in this respect with an average score for the usability components of 1.7. This means that on average the users felt the system was better than good for each of the usability factors. It is also much better than the average usability score of the existing systems, which scored 3.2 on average. These factors combine to show that the system developed is an improvement on the majority of existing systems in usability terms.

Figure 7.7: The results from the HCI section of the system assessment survey conducted on the developed system.

<table>
<thead>
<tr>
<th>Quality of HCl Factors in Final System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consistency</td>
</tr>
<tr>
<td>Simplicity</td>
</tr>
<tr>
<td>Clarity</td>
</tr>
<tr>
<td>Help and Support</td>
</tr>
<tr>
<td>Menu Paths</td>
</tr>
<tr>
<td>Layout</td>
</tr>
<tr>
<td>Overall (this system)</td>
</tr>
<tr>
<td>Overall (other systems)</td>
</tr>
</tbody>
</table>

Acceptance testing – algorithm animation techniques
The results displayed in the graph below are not as positive as the usability results. Although the results are an improvement on the majority of existing systems, users are still not finding sorting algorithms easy to learn. The results have an average score of 3.15 for point 1 (How easy to understand is Quick Sort based on the animation and additional information provided in the system?). Point 2 (How confident would you be at accurately describing Quick Sort after
watching the animation once and looking at the additional information provided in the system?) has an average score of 3.7, while point 3 (How confident would you be at accurately describing Quick Sort after watching the animation five times and looking at the additional information provided by the system?) has an average score of 3.05. Although these scores are not as good as had been hoped for, they were better than the average existing system score for points 1 and 2, and only marginally worse for point 3. The reasons for this appear to be obvious. Due to the quantity and accuracy of descriptive text the system offers to the users describing how the various algorithms work the user always has detailed information at hand which helps them to understand how the sorting algorithms work. This appears to be extremely beneficial initially, but then the users need the animation controls to help cement the concepts they have learnt. Due to these controls not functioning, and the system functioning on an extremely fast machine the user cannot see the actual animation clearly, thus are unable to learn anything extra from it. The only real benefit they get is from looking at the statistics generated by the algorithm that help to inform them how efficient the algorithm has been.

Figure 7.8: The results from the effectiveness of animation techniques section of the system assessment survey conducted on the developed system.

Key for figure 7.8:
1: How easy to understand is Quick Sort based on the animation and additional information provided in the system?
2: How confident would you be at accurately describing Quick Sort after watching the animation once and looking at the additional information provided in the system?
3: How confident would you be at accurately describing Quick Sort after watching the animation five times and looking at the additional information provided by the system?

The first three blocks represent the stack view, while the remaining three blocks represent the block view.

7.8 Testing conclusions
The black box testing conducted proved that the system met the vast majority of the requirements stated in the requirements specification section. The navigation between all the screens within the system worked smoothly, all information was easily viewable on the help and algorithm analysis screens, the single animation and algorithm race setup screens worked correctly, and the algorithms ran correctly generating data that was placed in the correct places. The only area where the system did not meet the requirements as expected was in the animation controls. These did not work; hence the user was unable to adjust the speed of the animation. This was an issue because due to the speed of the test computers the animations finished so quickly that the user was unable to see what was happening clearly.
System assessment surveys were used to assess the usability of the developed system. The results from these surveys showed that users found the system very easy to use, with the average score for usability being much better than the average score for usability in existing systems. The results also showed the system as better than the average existing system in two of the three points for the effectiveness of animation techniques, while only slightly worse than the average system for the third point. The effectiveness of the animations was cause for concern though. Due to reasons stated previously the animations offered the user no real benefit in terms of learning, leaving the user to rely heavily on the text descriptions provided to learn how the sorting algorithms worked.

8.0 Conclusion

8.1 Overview
This section provides a critical analysis of the system developed as well as the development process as a whole. It looks in detail at the areas where the project has been successful as well as the areas where the project could have been improved. It also discusses where the developed system fits within the existing range of systems in the sorting algorithm animation field. Is the developed system an improvement on existing systems? What has been learnt that would be useful to future developers in this area? Finally, it discusses how the system could be enhanced in the future, and specifies a number of logical extensions that would increase the systems appeal to a broader range of users.

8.2 How successful was the project?
To judge the success of the project the process used to develop the final system has to be looked at in detail along with the system developed. This is because a system that meets all of the specified requirements is of no use if the requirements elicitation process has gathered inappropriate requirements.

The literature review was the first section of the project and provided an excellent foundation for the project. It gave the developer a detailed understanding of the field of sorting algorithm animation systems, and clearly showed how the field has developed in the past 30 years as well as the reasons behind these developments. It also provided the developer with a rough set of requirements that the final system would need to meet to be beneficial to the field.

The requirements analysis section built on the work of the literature review by looking at the existing systems in far more detail. Surveys were conducted upon existing systems to see which features were beneficial to the users and which features were hindrances. It enabled the identification of areas that let the majority of existing systems down (such as help and text descriptions). This process helped extend and refine the requirements identified by the literature review, and the prototype section further refined these. The prototype section presented a number of proposed designs to users within the target user group for assessment. After feedback these design were modified several times until the users felt the design met their usability requirements. Although these prototypes were only static they provided an invaluable source of information as they enabled a layout that the users felt comfortable with to be designed. Dynamic prototyping would have been even more beneficial but unfortunately time constraints made this impossible. These techniques enabled accurate and relevant requirements to be identified. Although not all of these requirements were achieved, they were undoubtedly accurate and suitable for such a project, and provided justification for the design.
The design section was based upon the final design from the prototype section. Although only the single animation screen had been prototyped the rest of the system followed the style and colour scheme used in the prototype, as the target user group had deemed these acceptable. The designs were detailed and showed how each screen would look upon completion. It also described how the various screens would interact, and specified the data that would pass between them. Although it did not deal with specifying how the code was laid out in any great detail, the developer feels that if the design were given to a peer they would be able to develop a similar system.

The implementation process of the system was predominantly successful, although it did suffer slightly due to some sections taking longer to implement than expected, which affected the overall functionality of the system. The approach used for development was an evolutionary technique, with sections being implemented and tested before further sections could be started on as these later sections relied on earlier components working before they could be implemented. The GUI framework was developed first and to schedule, meeting all the requirements specified for it. Navigation was straightforward throughout the entire system and the acceptance testing showed that the system scored highly in terms of usability. The algorithm analysis and help screens were also developed to schedule and enabled the user to easily access information on how to do tasks within the system if they required. It also enabled them to access detailed descriptions of each sorting algorithm implemented in the system. The setup screens for both the single animation and the algorithm race functioned as required which meant that the users could specify the exact parameters of the animation(s) they wished to see in operation. Problems were encountered on the implementation of the single animation and animation race screens. The problems that arose were to do with the developer discovering the best way in which to layout the screen so that all of the components were displayed clearly to the user. Such are the apparent inconsistencies in the Java layout managers that this took far longer than had been anticipated. Although ultimately these setbacks were overcome they had caused a significant delay that influenced the implementation of other sections of the system. For example, the animation controls were not successfully implemented due to the time constraints that these development problems had caused.

The final system was a success despite the problems described. Although exhaustive testing was impossible due to the quantity of code produced the tests conducted were sufficient to show that the vast majority of requirements were met. The system assessment surveys proved that the system was significantly more usable than the majority of existing systems. The only serious problem with the system was that the animation controls did not function properly, meaning that the speed of the animations could not be modified. Despite this setback the system was a slight improvement on the average existing system in terms of ease of learning, as it scored better than the average system in two of the three areas tested against. This was predominantly due to the text descriptions provided which made up for the lack of animation controls in the majority of cases.

8.3 Were the requirements appropriate and achievable?
As stated previously the system requirements were reached by an extensive requirements elicitation and analysis process. A number of techniques were used which included dialogue with the users from the target group. The extensive nature of the requirements elicitation process, coupled with the numerous iterations the requirements went through to refine them, leads the developer to believe that the requirements were entirely appropriate for the system. This opinion is enforced by the system assessment surveys conducted upon the final system. They gave the system good usability marks, thus justifying the usability-based requirements. Although they only gave the system average marks in ease of learning terms the developer feels this is due to the previously mentioned functionality problems with the final system. Had all of the functional requirements been met the developer has no doubts that the system would have scored well for this aspect of the system, thus justifying the requirements specified for this part of the system.
The only requirements that were not achieved were those relating to the ability for the user to control the speed of the animation. These were perfectly reasonable requirements to include, as they would have enhanced the users learning had they functioned as desired. Unfortunately, these requirements were not met due to reasons mentioned earlier.

### 8.4 Is the system an improvement on existing systems?

In terms of usability the system was undoubtedly an improvement on the vast majority of existing systems. Comparing the results of the system assessment surveys proved this. The system developed scored significantly better on all areas than the average existing systems. Although a few of the existing systems were better than the developed system in terms of usability, the system developed is certainly one of the better existing systems in usability terms.

When judged by its effectiveness as a teaching aid the system developed scored very similarly to the average existing system, being slightly better in two of the areas assessed, and marginally worse in the other. This means that although the system was certainly no worse than the average existing system in terms of ease of learning, it was not significantly better. This was disappointing, but when the areas of usability and effectiveness as a teaching aid are considered together the system developed is clearly a system that provide greater usability and functionality than the majority of existing systems.

### 8.5 How could the system be improved?

As with all systems there is room for improvement within the system developed. The major way in which the system could be improved is by implementing the animation controls successfully. This would enable the user to have more control over the animations, thus enabling them to fine tune their understanding of the various sorting algorithms within the system. Other ways in which the existing system could be enhanced are modifying the GUI in the single animation and animation race screens, so that the animations always occupy an equal amount of space, regardless of the number of elements being sorted. This would enhance the consistency of the system thus improve the usability somewhat.

More sorting algorithms could also be added to the system. Including algorithms such as Shell Sort and Bi-directional Quick Sort would make the system more comprehensive and would clearly allow the users to learn about these types of sorting algorithms. This would lead to a small problem with the animation race screens though, as this can currently only support five animations simultaneously. A decision would have to be made on whether it would be necessary to show more animations simultaneously, or to simply leave the system as it is and just add support for the new algorithms.

If more sorting algorithms were to be included then different animation techniques are likely to be required. For example, sorting techniques that use tree mechanisms for sorting are likely to be easier for a user to understand if they are animated using the technique shown in Figure 2.2, rather than the stack or block views used in this project. Techniques that use merging to sort data may also be easier to understand if shown in an alternate manner to the stack and block views. For example, in Merge Sort the two segments of the data that are sorted before the merging occurs could be displayed next to each other, with the actual merging process also being shown in some way. Clearly, this is a wide area that would need thorough investigation, to determine the best animation techniques for more complex sorting algorithms, if this extension were to be implemented.

Another enhancement for the system would be the addition of graphs showing the efficiency of each sorting algorithm against a specified amount of elements. Currently the complexity of each algorithm is described in terms of its best case, worst case, and average case, but displaying this
information in a graphical form would probably allow the user to learn the efficiency of each sorting algorithm more quickly and easily, as well as allowing the sorting algorithms to be compared at a glance. This feature was intended for inclusion in the system but had to be excluded due to time constraints.

The sizes of the data that can be sorted could also be increased. Currently the data is limited in size to fifty elements in the single animation screen, and twenty elements in the algorithm race screen. This undoubtedly means that the results are more likely to be influenced by fluke data when using random data. Some of the algorithms are much better comparatively when using much larger quantities of data, for example Quick Sort. The current system does not show this especially well so this would definitely be a future enhancement that would be strongly considered.

A much more general improvement that could be made to the functionality of the system would be turning it into a generic learning tool for Computer Science. Such a system could provide details on common topics in Computer Science such as ‘if else’ statements and ‘for’ loops, animating them where appropriate. Extending the system in this way would undoubtedly be hugely beneficial to Computer Science students, as nothing similar currently exists.

Finally, more comprehensive testing could be conducted. In particular, the usability testing could be more comprehensive. Although twenty users from the target user group were selected for acceptance testing a more comprehensive survey could have raised more issues on the usability of the system. It would be extremely useful to test the system on a number of lecturers of the topic, as they would undoubtedly be able to make more recommendations.

8.6 Guidelines for future developers in algorithm animation systems

Below is a list of guidelines that the developer would advise any person undertaking a similar project in the future to consider carefully before starting development.

- Usability of the system must be at the very core of the design, as poor usability will cause users to reject any system developed, regardless of its functionality.

- Text descriptions within the system are crucial to assist user learning. This appears to be because some of the sorting algorithms are very complex so the user is unlikely to understand them merely by watching an animation, regardless of how good the animation is. Text descriptions supplement the animations, improving the users understanding of the sorting algorithms. A lack of text descriptions was a clear flaw in many of the existing systems as it was severely detrimental to the ease of learning. This was discovered by the system assessment surveys conducted on the existing systems.

- Detailed help within the system is important. The system assessment surveys showed that the majority of the existing systems failed because they did not have detailed information on how to use the system, or if they did it was not in an obvious place. Providing detailed help in an easily accessible place makes the users feel comfortable with the system far more quickly.

- Stack view appears to be the best view to use in terms of learning Bubble sort, Quick sort, Heap sort, Insertion sort and Selection Sort, with block view being the second best view. Other views such as dot view simply offer the user nothing in terms of learning how the sorting algorithm works. This was discovered by the system assessment surveys.

- Animation controls are important to allow users to learn how sorting algorithms work. This point was proven by the acceptance testing on the developed system, where the
users commented on how effective the text descriptions were, but also stated that the lack of ability to execute the animation at their desired speed impeded their ease of learning of the sorting algorithms. They felt that both needed to be implemented successfully to provide a system that was really easy to learn from.

8.7 Skills the developer has learnt
The developer believes he has learnt a great deal during the course of the project, both technically and non-technically. The list below states the major skills that have been learnt or enhanced.

- The writing of a detailed report has improved the developers report writing skills.
- The literature review has allowed the developer to learn how to systematically examine existing sources on a given topic, in the form of academic journals, books, and web sites.
- The carrying out of a detailed design for the proposed system has improved the developers design skills dramatically.
- Creating the user interface using swing has enhanced the developer’s knowledge of creating GUIs using Java.
- Developing the project has enhanced the developer’s knowledge, confidence, and competence of Java.
- Creating and using the system assessment surveys has given the developer a far greater understanding on how to ascertain user opinion for a piece of software, as well as collecting these opinions in a quantitative form that is useful.

8.8 Summary
Despite not meeting all of the requirements specified the system produced was an improvement on the majority of existing systems, with the detailed text descriptions of the sorting algorithm proving key to this. The developer feels that with a few refinements the system could be of significant use as a teaching tool, and hopes to find time to make these refinements in the future. The developer also feels that ultimately for the user to really understand how sorting algorithms work the best method is for them to develop the algorithms themselves, as this will give them a far greater understanding than watching animations and reading descriptions. The project itself has been many things, challenging, frustrating and demanding, but ultimately extremely rewarding as the developer feels he has learnt an enormous amount from this piece of work.
Appendix A – System Assessment Surveys On Existing Systems

A.1 System assessment surveys
The first goal of this survey is to give a rating to each aspect of the specified sorting algorithm animation systems for the following six HCI factors:

- System consistency.
- Simplicity.
- Clarity.
- Help and support.
- Menu paths.
- Layout.

These factors were chosen as research conducted in the literature review identified these six aspects as key to the success of an interface in terms of its usability and its acceptance by users.

The other major goal of the survey was to judge how effective the different approaches to animating the sorting algorithms were. Once a sufficient sample had been taken the most usable system could be easily identifiable. The good parts of this system were then incorporated into the design of the developed system where appropriate, while the bad aspects were avoided. This survey also allowed the identification of the best and worst animation techniques. Clearly the best were included, while the worst were excluded, as they appeared to offer no benefits to the user whatsoever.

All questions asked within this assessment use the Likert scale. Each factor was measured from one to five, with one being excellent, three being average, and five being very poor.

A.2 System assessment 1

<table>
<thead>
<tr>
<th>System name:</th>
<th>PolkaW – (Polka for Windows)</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL:</td>
<td><a href="http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html">http://www.cc.gatech.edu/gvu/softviz/parviz/polka.html</a></td>
</tr>
</tbody>
</table>

A.2.1 HCI Factors

Consistency
How consistent are the icons within the system? 1

Simplicity
How easy to understand is the text and the icons used in the system? 4

Clarity
How clear is the purpose of each screen in the system? 4

Help and support
How detailed, relevant and accurate is the help provided by the system? 5
### Menu paths
Are the menu paths easy to navigate and intuitive?

### Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

### A.2.2 Algorithm animation

#### Stack view
How easy to understand is quick sort based on the animation?

- How confident would you be at accurately describing quick sort after watching the animation once? 5
- How confident would you be at accurately describing quick sort after watching the animation five times? 4

#### Block view
How easy to understand is quick sort based on the animation?

- How confident would you be at accurately describing quick sort after watching the animation once? 5
- How confident would you be at accurately describing quick sort after watching the animation five times? 4

#### Dot view
How easy to understand is quick sort based on the animation?

- How confident would you be at accurately describing quick sort after watching the animation once? 5
- How confident would you be at accurately describing quick sort after watching the animation five times? 5

### Other approaches

- No other animation techniques were provided.

How easy to understand is quick sort based on the animation? n/a

- How confident would you be at accurately describing quick sort after watching the animation once? n/a
- How confident would you be at accurately describing quick sort shown after watching the animation five times? n/a
A.2.3 Comments

Polka seems geared towards people who already know a lot about sorting algorithms as it has facilities allowing the user to implement their own sorting algorithms. It is also a difficult program to use with no obvious help facilities.

A.2.4 Screenshot

Figure A.1: The block view of quick sort in PolkaW.

A.3 System assessment 2

System name: Java Sorting Animation

URL: http://www.cs.brockport.edu/cs/javasort.html

A.3.1 HCI factors

Consistency
How consistent are the icons within the system? 1

Simplicity
How easy to understand is the text and the icons used in the system? 2

Clarity
How clear is the purpose of each screen in the system? 2
Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

A.3.2 Algorithm animation

Stack view
How easy to understand is the quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Block view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Dot view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Other approaches

No other animation techniques were provided.

How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

A.3.3 Comments

The animation is large and well spaced out, and all the elements are easy to distinguish. The controls offered are standard for such systems, but no description of what is happening is given.

A.3.4 Screenshot

*Figure A.2: Block view of quick sort before the animation has started.*

A.4 System assessment 3

System name: Sorting Algorithm Animation System

URL: www.mundayweb.com/neil/index.html

A.4.1 HCI factors
### Consistency
How consistent are the icons within the system?

| How consistent are the icons within the system? | 1 |

### Simplicity
How easy to understand is the text and the icons used in the system?

| How easy to understand is the text and the icons used in the system? | 1 |

### Clarity
How clear is the purpose of each screen in the system?

| How clear is the purpose of each screen in the system? | 1 |

### Help and support
How detailed, relevant and accurate is the help provided by the system?

| How detailed, relevant and accurate is the help provided by the system? | 3 |

### Menu paths
Are the menu paths easy to navigate and intuitive?

| Are the menu paths easy to navigate and intuitive? | n/a |

### Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

| Do the components in the system have sufficient space so that they can be easily identified and easily selected? | 2 |

### A.4.2 Algorithm animation

#### Stack view
How easy to understand is quick sort based on the animation?

| How easy to understand is quick sort based on the animation? | 4 |

How confident would you be at accurately describing quick sort after watching the animation once?

| How confident would you be at accurately describing quick sort after watching the animation once? | 5 |

How confident would you be at accurately describing quick sort after watching the animation five times?

| How confident would you be at accurately describing quick sort after watching the animation five times? | 4 |

#### Block view
How easy to understand is quick sort based on the animation?

| How easy to understand is quick sort based on the animation? | 4 |

How confident would you be at accurately describing quick sort after watching the animation once?

| How confident would you be at accurately describing quick sort after watching the animation once? | 5 |

How confident would you be at accurately describing quick sort after watching the animation five times?

| How confident would you be at accurately describing quick sort after watching the animation five times? | 4 |

#### Dot view
How easy to understand is quick sort based on the animation?

| How easy to understand is quick sort based on the animation? | 5 |

How confident would you be at accurately describing quick sort after watching the animation once?

| How confident would you be at accurately describing quick sort after watching the animation once? | 5 |

How confident would you be at accurately describing quick sort after watching the animation five times?

| How confident would you be at accurately describing quick sort after watching the animation five times? | 5 |
Other approaches

An approach called the ‘colour stack view’ was implemented. This can be seen below.

How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

A.4.3 Comments

The system is well laid out, with all aspects being easy to distinguish. As with the previous system, no descriptions are offered while the animation is running to enhance learning.

A.4.4 Screenshots

Figure A.3: The single animation view in SAAS.

Figure A.4: The unsorted and sorted data using colour stack view.
A.5 System assessment 4

System name: Interactive Tutorial for Quick Sort

URL: http://students.ceid.upatras.gr/~pirot/java/Quicksort/

A.5.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?

Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

A.5.2 Algorithm animation

Stack view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Block view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?
**Dot view**
How easy to understand is quick sort based on the animation?  
How confident would you be at accurately describing quick sort after watching the animation once?  
How confident would you be at accurately describing quick sort after watching the animation five times?

**Other approaches**
No other animation techniques were provided.

How easy to understand is quick sort based on the animation?  
How confident would you be at accurately describing quick sort after watching the animation once?  
How confident would you be at accurately describing quick sort after watching the animation five times?

**A.5.3 Comments**
This technique talks the user through the animation, saying what is happening at each point. This made it much easier for users to pick up what was happening in the sorting algorithm.

**A.5.4 Screenshot**

*Figure A.5: The description section of this system is clearly visible in the centre of the screen. This supplements the users understanding.*
### A.6 System assessment 5

<table>
<thead>
<tr>
<th>System name:</th>
<th>Quick Sort Animation</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL:</td>
<td><a href="http://www.cs.arizona.edu/people/peter/overview/demos/quicksort">http://www.cs.arizona.edu/people/peter/overview/demos/quicksort</a></td>
</tr>
</tbody>
</table>

#### A.6.1 HCI factors

<table>
<thead>
<tr>
<th>Consistency</th>
<th>How consistent are the icons within the system?</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>How easy to understand is the text and the icons used in the system?</td>
<td>3</td>
</tr>
<tr>
<td>Clarity</td>
<td>How clear is the purpose of each screen in the system?</td>
<td>3</td>
</tr>
<tr>
<td>Help and support</td>
<td>How detailed, relevant and accurate is the help provided by the system?</td>
<td>5</td>
</tr>
<tr>
<td>Menu paths</td>
<td>Are the menu paths easy to navigate and intuitive?</td>
<td>3</td>
</tr>
<tr>
<td>Layout</td>
<td>Do the components in the system have sufficient space so that they can be easily identified and easily selected?</td>
<td>4</td>
</tr>
</tbody>
</table>

#### A.6.2 Algorithm animation

<table>
<thead>
<tr>
<th>Stack view</th>
<th>How easy to understand is quick sort based on the animation?</th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>How confident would you be at accurately describing quick sort after watching the animation once?</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td>How confident would you be at accurately describing quick sort after watching the animation five times?</td>
<td>n/a</td>
</tr>
<tr>
<td>Block view</td>
<td>How easy to understand is quick sort based on the animation?</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>How confident would you be at accurately describing quick sort after watching the animation once?</td>
<td>4</td>
</tr>
</tbody>
</table>
How confident would you be at accurately describing quick sort after watching the animation five times?

Dot view
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

Other approaches

No other animation techniques were provided.

How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

A.6.3 Comments

This system was of no use whatsoever as it was impossible to understand what was happening due to the variation of the dot view that was used.

A.6.4 Screenshot

Figure A.6: The modified dot view is difficult to understand.
A.7 System assessment 6

System name: Sorting Algorithm Demos

URL: http://www.cs.ubc.ca/spider/harrison/Java/sorting-demo.html

A.7.1 HCI factors

Consistency
How consistent are the icons within the system? n/a

Simplicity
How easy to understand is the text and the icons used in the system? 3

Clarity
How clear is the purpose of each screen in the system? 3

Help and support
How detailed, relevant and accurate is the help provided by the system? 5

Menu paths
Are the menu paths easy to navigate and intuitive? n/a

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 3

A.7.2 Algorithm animation

Stack view
How easy to understand is quick sort based on the animation? 5

How confident would you be at accurately describing quick sort after watching the animation once? 5

How confident would you be at accurately describing quick sort after watching the animation five times? 5

Block view
How easy to understand is quick sort based on the animation? n/a

How confident would you be at accurately describing quick sort after watching the animation once? n/a
How confident would you be at accurately describing quick sort after watching the animation five times?

**Dot view**

How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Other approaches**

<table>
<thead>
<tr>
<th></th>
<th>n/a</th>
</tr>
</thead>
<tbody>
<tr>
<td>No other animation techniques were provided.</td>
<td></td>
</tr>
</tbody>
</table>

How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**A.7.3 Comments**

This system is extremely simple and the user learns almost nothing from observing it.

**A.7.4 Screenshot**

*Figure A.7: The simple but commonplace approach to animating sorting algorithms on the Internet. This approach offers little to the user.*

**A.8 System assessment 7**

System name: Sorting Algorithms

URL:

A.8.1 HCI factors

**Consistency**
How consistent are the icons within the system?  

**Simplicity**
How easy to understand is the text and the icons used in the system?  

**Clarity**
How clear is the purpose of each screen in the system?  

**Help and support**
How detailed, relevant and accurate is the help provided by the system?  

**Menu paths**
Are the menu paths easy to navigate and intuitive?  

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected?  

A.8.2 Algorithm animation

**Stack view**
How easy to understand is quick sort based on the animation?  

How confident would you be at accurately describing quick sort after watching the animation once?  

How confident would you be at accurately describing quick sort after watching the animation five times?  

**Block view**
How easy to understand is quick sort based on the animation?  

How confident would you be at accurately describing quick sort after watching the animation once?  

How confident would you be at accurately describing quick sort after watching the animation five times?  

**Dot view**
How easy to understand is quick sort based on the animation?  

How confident would you be at accurately describing quick sort after watching the animation once?  

How confident would you be at accurately describing quick sort
after watching the animation five times?

**Other approaches**

| No other animation techniques were provided. |

How easy to understand is quick sort based on the animation? n/a

How confident would you be at accurately describing quick sort after watching the animation once? n/a

How confident would you be at accurately describing quick sort after watching the animation five times? n/a

**A.8.3 Comments**

This is another of the typical approaches to animating sorting algorithms on the Internet. Again, it offers little to the user.

**A.8.4 Screenshot**

*Figure A.8: Quick sort using stack view.*

**A.9 System assessment 8**

<table>
<thead>
<tr>
<th>System name:</th>
<th>Algorithm Animations</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL:</td>
<td><a href="http://math.ucsd.edu/~fan/math188/bonus/park/sorting.htm">http://math.ucsd.edu/~fan/math188/bonus/park/sorting.htm</a></td>
</tr>
</tbody>
</table>

**A.9.1 HCI factors**

**Consistency** 3
How consistent are the icons within the system?

**Simplicity**
How easy to understand is the text and the icons used in the system?

**Clarity**
How clear is the purpose of each screen in the system?

**Help and support**
How detailed, relevant and accurate is the help provided by the system?

**Menu paths**
Are the menu paths easy to navigate and intuitive?

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

### A.9.2 Algorithm animation

**Stack view**
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Block view**
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Dot view**
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Other Approaches**

No other animation techniques were provided.
How easy to understand is quick sort based on the animation?  

How confident would you be at accurately describing quick sort after watching the animation once?  

How confident would you be at accurately describing quick sort after watching the animation five times?  

A.9.3 Comments

This approach shows which section of code is currently being executed. This helps the user understand the sorting algorithms slightly so long as they are proficient with the programming language being used.

A.10 System assessment 9

System name: Sorting Algorithm Animations

URL: http://www.cs.bell-labs.com/cm/cs/pearls/sortanim

A.10.1 HCI factors

Consistency  
How consistent are the icons within the system?  

Simplicity  
How easy to understand is the text and the icons used in the system?  

Clarity  
How clear is the purpose of each screen in the system?  

Help and Support  
How detailed, relevant and accurate is the help provided by the system?  

Menu Paths  
Are the menu paths easy to navigate and intuitive?  

Layout  
Do the components in the system have sufficient space so that they can be easily identified and easily selected?  

A.10.2 Algorithm animation

Stack view
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Block view**
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**Dot view**
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**Other approaches**

No other animation techniques were provided.

How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**A.10.3 Comments**

This system allows the user to specify the type of data to operate on and the amount of items to use. However, it only allows the user to view the animation in dot view so is therefore extremely difficult to understand.
A.10.4 Screenshot

Figure A.9: Quick sort being shown in the dot view.

A.11 System assessment 10

System name: Sorting Algorithms Demo

URL: http://java.sun.com/applets/jdk/1.0/demo/sortdemo/example1.html

A.11.1 HCI factors

**Consistency**
How consistent are the icons within the system? 3

**Simplicity**
How easy to understand is the text and the icons used in the system? 3

**Clarity**
How clear is the purpose of each screen in the system? 4

**Help and support**
How detailed, relevant and accurate is the help provided by the system? 5

**Menu paths**
Are the menu paths easy to navigate and intuitive? n/a
Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

A.11.2 Algorithm animation

Stack view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Block view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Dot view
How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

Other approaches
No other animation techniques were provided.

How easy to understand is quick sort based on the animation?

How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

A.11.3 Comments

This system is extremely rudimentary. It offers no real benefits to the user and seems to be merely an exercise for the benefit of the designer rather than the user.
A.11.4 Screenshots

*Figure A.10: Another example of the simple methods used to show sorting algorithms on the Internet.*

A.12 System assessment 11

**System name:** Data Structures and Algorithms: Quick Sort

**URL:** http://ciips.ee.uwa.edu.au/~morris/Year2/PLDS210/qsor.html

A.12.1 HCI factors

**Consistency**
How consistent are the icons within the system?  

**Simplicity**
How easy to understand is the text and the icons used in the system?  

**Clarity**
How clear is the purpose of each screen in the system?  

**Help and support**
How detailed, relevant and accurate is the help provided by the system?  

**Menu paths**
Are the menu paths easy to navigate and intuitive?  

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected?  

A.12.2 Algorithm animation

**Stack view**
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?

How confident would you be at accurately describing quick sort after watching the animation five times?

**Block view**
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**Dot view**
How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**Other approaches**

No other animation techniques were provided.

How easy to understand is quick sort based on the animation?
How confident would you be at accurately describing quick sort after watching the animation once?
How confident would you be at accurately describing quick sort after watching the animation five times?

**A.12.3 Comments**

Although the interface for this system wasn’t especially well designed the animations were the easiest to understand of any system, as there was an ongoing description of what was happening as the algorithm executed. This makes the sorting algorithm far easier to understand.
A.12.4 Screenshot

Figure A.11: This shows the description we are given as the array of elements is sorted.

Appendix B – Test Plan

B.1 Test plan
The test plan covers all the different types of functionality offered by the system. It is written in the style described in the testing section of this document.

B.2 Welcome screen tests

System Area: The welcome screen

Specification:

The welcome screen is the first part of the system that the user sees, and is the basis for all navigation within the system. Effectively it acts as a portal, allowing the user to access all other sections of the system from this one screen. Navigation is achieved by selecting one of the menu items in the menu bar at the top of the welcome screen using the mouse.

Input: Selecting one of the menu items using the mouse

Output: The user will be moved to the desired screen.

Test Cases:

Test 1 - Single animation menu item (left hand mouse button)
Input: Select the ‘single animation’ menu item with the left hand mouse button.
Test result: Test passed.
Test 2 – Algorithm race menu item (left hand mouse button)
Input: Select the ‘algorithm race’ menu item with the left hand mouse button.
Expected output: ‘Algorithm race setup’ screen displayed.
Actual output: ‘Algorithm race setup’ screen displayed.
Test result: Test passed.

Test 3 – Help menu item (left hand mouse button)
Input: Select the ‘help menu’ item with the left hand mouse button.
Expected output: ‘Help screen’ displayed.
Actual output: ‘Help screen’ displayed.
Test result: Test passed.

Test 4 – Algorithm analysis menu item (left hand mouse button)
Input: Select ‘algorithm analysis’ menu item with the left hand mouse button.
Expected output: ‘Algorithm analysis’ screen displayed.
Actual output: ‘Algorithm analysis’ screen displayed.
Test result: Test passed.

Test 5 – About menu item (left hand mouse button)
Input: Select ‘about’ menu item with the left hand mouse button.
Expected output: ‘About’ screen displayed.
Actual output: ‘About’ screen displayed.
Test result: Test passed.

Test 6 – Quit menu item (left hand mouse button)
Input: Select ‘quit’ menu item with the left hand mouse button.
Expected output: ‘Quit’ screen displayed.
Actual output: ‘Quit’ screen displayed.
Test result: Test passed.

Test 7 - Single animation menu item (right hand mouse button)
Input: Select ‘single animation’ menu item with the right hand mouse button.
Test result: Test passed.

Test 8 – Algorithm race menu item (right hand mouse button)
Input: Select ‘algorithm race’ menu item with the right hand mouse button.
Expected output: ‘Algorithm race setup’ screen displayed.
Actual output: ‘Algorithm race setup’ screen displayed.
Test result: Test passed.

Test 9 – Help menu item (right hand mouse button)
Input: Select ‘help menu’ item with the right hand mouse button.
Expected output: ‘Help screen’ displayed.
Actual output: ‘Help screen’ displayed.
Test result: Test passed.
**Test 10 – Algorithm analysis menu item (right hand mouse button)**
Input: Select ‘algorithm analysis’ menu item with the right hand mouse button.
Expected output: ‘Algorithm analysis’ screen displayed.
Actual output: ‘Algorithm analysis’ screen displayed.
Test result: Test passed.

**Test 11 – About menu item (right hand mouse button)**
Input: Select ‘about’ menu item with the right hand mouse button.
Expected output: ‘About’ screen displayed.
Actual output: ‘About’ screen displayed.
Test result: Test passed.

**Test 12 – Quit menu item (right hand mouse button)**
Input: Select ‘quit’ menu item with the right hand mouse button.
Expected output: ‘Quit’ screen displayed.
Actual output: ‘Quit’ screen displayed.
Test result: Test passed.

**B.3 Quit screen tests**

**System Area:** The quit screen

**Specification:**

The quit screen allows the user to exit the system, if the user has selected the quit menu item by accident they can return to the welcome screen. Navigation is achieved by selecting the desired button.

Input: Selecting the yes or no buttons with the mouse

Output: The user will exit the system; the user will be moved to the welcome screen.

**Test Cases:**

**Test 1 – Yes button (left hand mouse button)**
Input: Select ‘yes’ button with the left hand mouse button.
Expected output: System exited.
Actual output: System exited.
Test result: Test passed.

**Test 2 – Yes button (right hand mouse button)**
Input: Select ‘yes’ button with the right hand mouse button.
Expected output: System exited.
Actual output: System exited.
Test result: Test passed.

**Test 3 – No button (left hand mouse button)**
Input: Select ‘no’ button with the left hand mouse button.
Expected output: Welcome screen displayed.
Actual output: Welcome screen displayed.
Test result: Test passed.

Test 4 – No button (right hand mouse button)
Input: Select ‘no’ button with the right hand mouse button.
Expected output: Welcome screen displayed.
Actual output: Welcome screen displayed.
Test result: Test passed.

B.4 About screen tests

System Area: About screen

Specification:

The about screen displays information on the developer of the system. The user can only navigate back to the welcome screen from the about screen.

Input: Selecting the ‘cancel’ button.

Output: The user will be moved to the welcome screen.

Test Cases:

Test 1 – Cancel button (left hand mouse button)
Input: Select ‘cancel’ button with the left hand mouse button.
Expected output: Welcome screen displayed.
Actual output: Welcome screen displayed.
Test result: Test passed.

Test 2 – Cancel button (right hand mouse button)
Input: Select ‘cancel’ button with the right hand mouse button.
Expected output: Welcome screen displayed.
Actual output: ‘Welcome’ screen displayed.
Test result: Test passed.

B.5 Algorithm analysis screen tests

System Area: Algorithm analysis screen

Specification:

The algorithm analysis screen allows the user to look at detailed descriptions of the sorting algorithms included in the system. It also allows the user to return the welcome screen.

Input: Selecting topics from the combo box, pressing the display topic button, pressing the cancel button.
Output: The user will be moved to the welcome screen, desired topic will be shown.

Test Cases:

Test 1 – Cancel button (left hand mouse button)
Input: Select ‘cancel’ button with the left hand mouse button.
Expected output: ‘Welcome’ screen displayed.
Actual output: ‘Welcome’ screen displayed.
Test result: Test passed.

Test 2 – Cancel button (right hand mouse button)
Input: Select ‘cancel’ button with the right hand mouse button.
Expected output: ‘Welcome’ screen displayed.
Actual output: ‘Welcome’ screen displayed.
Test result: Test passed.

Test 3 – Bubble sort and display topic button
Input: Select ‘bubble sort’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected output: Bubble sort information displayed in the text area.
Actual output: Bubble sort information displayed in the text area.
Test result: Test passed.

Test 4 – Quick sort and display topic button
Input: Select ‘quick sort’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected output: Quick sort information displayed in the text area.
Actual output: Quick sort information displayed in the text area.
Test result: Test passed.

Test 5 – Insertion sort and display topic button
Input: Select ‘insertion sort’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected output: Insertion sort information displayed in the text area.
Actual output: Insertion sort information displayed in the text area.
Test result: Test passed.

Test 6 – Selection sort and display topic button
Input: Select ‘selection sort’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected output: Selection sort information displayed in the text area.
Actual output: Selection sort information displayed in the text area.
Test result: Test passed.

Test 7 – Heap sort and display topic button
Input: Select ‘heap sort’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected output: Heap sort information displayed in the text area.
Actual output: Heap sort information displayed in the text area.
Test result: Test passed.

B.6 Help screen tests

System Area: Help screen

Specification:

The help screen allows the user to look at detailed descriptions of how to do tasks in the system. It also provides answers to common questions that users may have. The descriptions provided are placed in a central text area. This screen also allows the user to return to the welcome screen.

Input: Selecting topics from the combo box, pressing the display topic button, pressing the cancel button.

Output: The user will be moved to the welcome screen, or the desired topic will be shown in the text area.

Test Cases:

Test 1 – Cancel button (left hand mouse button)
Input: Select ‘cancel’ button with the left hand mouse button.
Expected Output: ‘Welcome’ screen displayed.
Actual Output: ‘Welcome’ screen displayed.
Test Result: Test passed.

Test 2 – Cancel button (right hand mouse button)
Input: Select ‘cancel’ button with the right hand mouse button.
Expected Output: ‘Welcome’ screen displayed.
Actual Output: ‘Welcome’ screen displayed.
Test Result: Test passed.

Test 3 – FAQS and display topic button
Input: Select ‘FAQS’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: FAQS information displayed in the text area.
Actual Output: FAQS information displayed in the text area.
Test Result: Test passed.

Test 4 – How do I run a single animation? and display topic button
Input: Select ‘How do I run a single animation?’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: Steps on how to run a single animation displayed in the text area.
Actual Output: Steps on how to run a single animation displayed in the text area.
Test Result: Test passed.
Test 5 – How do I run an algorithm race? and display topic button
Input: Select ‘How do I run an algorithm race?’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: Steps on how to run an algorithm race displayed in the text area.
Actual Output: Steps on how to run an algorithm race displayed in the text area.
Test Result: Test passed.

Test 6 – How do I quit the System? and Display Topic Button
Input: Select ‘How do I quit the System?’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: Steps on how to quit the system displayed in the text area.
Actual Output: Steps on how to quit the system displayed in the text area.
Test Result: Test passed.

Test 7 – Walk-Through Guides and Display Topic Button
Input: Select ‘Walk-Through guides’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: Information displayed telling the user to select a specific walk-through guide from the list.
Actual Output: Information displayed telling the user to select a specific walk-through guide from the list.
Test Result: Test passed.

Test 8 – Please Select a Help Topic and Display Topic Button
Input: Select ‘Please select a help topic’ from the combo box, and then the ‘display topic’ button using the mouse.
Expected Output: Information displayed telling the user to select a specific topic from the list in the text area.
Actual Output: Information displayed telling the user to select a specific topic from the list in the text area.
Test Result: Test passed.

B.7 Menu bar item tests

System Area: Menu bar items

Specification:

Unless the user is on the welcome screen the menu bar items should be greyed out and should not be operable.

Input: Selecting a menu item.

Output: User should remain on current screen with no changes made to the system.

Test Cases:
Test 1 – ‘Single Animation’ menu item
Input: Select ‘Single Animation’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

Test 2 – ‘Algorithm Race’ menu item
Input: Select ‘Algorithm Race’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

Test 3 – ‘Help’ menu item
Input: Select ‘Help’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

Test 4 – ‘Algorithm Analysis’ menu item
Input: Select ‘Algorithm Analysis’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

Test 5 – ‘About’ menu item
Input: Select ‘About’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

Test 6 – ‘Quit’ menu item
Input: Select ‘Quit’ menu item whilst on any screen except the welcome screen.
Expected Output: User remains on current screen with no changes made to the system.
Actual Output: User remains on current screen with no changes made to the system.
Test Result: Test passed.

B.8 Single animation setup screen tests

System Area: Functionality of the ‘single animation setup’ screen.

Specification:

The single animation setup screen allows the user to specify the characteristics of the animation they are setting up by using the various combo boxes on this screen.
Input: Selecting an option from any of the combo boxes, then selecting the ‘start’ button.

Output: The selected information from the combo boxes will be displayed in the relevant places on the ‘main screen’ in single animation format.

Test Cases:

**Test 1 – ‘Animation type’ combo box (stack)**
Input: ‘Stack’ selected from the ‘Animation type’ combo box, then press ‘start animation’ button.
Expected Output: The data is displayed in the stack view in the ‘single animation screen’. The user is moved to the ‘single animation screen’.
Actual Output: The data is displayed in the stack view in the ‘single animation screen’. The user is moved to the ‘single animation screen’.
Test Result: Test passed.

**Test 2 – ‘Animation type’ combo box (block)**
Input: ‘Block’ selected from the ‘Animation type’ combo box, then press ‘start animation’ button.
Expected Output: The data is displayed in the block view in the ‘single animation screen’. The user is moved to the ‘single animation screen’.
Actual Output: The data is displayed in the block view in the ‘single animation screen’. The user is moved to the ‘single animation screen’.
Test Result: Test passed.

**Test 3 – ‘Algorithm type’ combo box (Bubble Sort)**
Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. Details of Bubble Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Bubble Sort.
Actual Output: The user is moved to the single animation screen. Details of Bubble Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Bubble Sort.
Test Result: Test passed.

**Test 4 – ‘Algorithm type’ combo box (Quick Sort)**
Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. Details of Quick Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Quick Sort.
Actual Output: The user is moved to the single animation screen. Details of Quick Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Quick Sort.
Test Result: Test passed.
Test 5 – ‘Algorithm type’ combo box (Insertion Sort)
Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. Details of Insertion Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Insertion Sort.
Actual Output: The user is moved to the single animation screen. Details of Insertion Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Insertion Sort.
Test Result: Test passed.

Test 6 – ‘Algorithm type’ combo box (Selection Sort)
Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. Details of Selection Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Selection Sort.
Actual Output: The user is moved to the single animation screen. Details of Selection Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Selection Sort.
Test Result: Test passed.

Test 7 – ‘Algorithm type’ combo box (Heap Sort)
Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. Details of Heap Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Heap Sort.
Actual Output: The user is moved to the single animation screen. Details of Heap Sort are placed in the description box in the single animation screen. When the animation is started in the single animation view the data is sorted using Heap Sort.
Test Result: Test passed.

Test 8 – ‘Data type’ combo box (Random)
Input: Select ‘random’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. The data is order randomly in the animation screen.
Actual Output: The user is moved to the single animation screen. The data is order randomly in the animation screen.
Test Result: Test passed.

Test 9 – ‘Data type’ combo box (Reverse Sorted)
Input: Select ‘reverse sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. The data is reverse sorted in the animation screen.
Actual Output: The user is moved to the single animation screen. The data is reverse sorted in the animation screen.
Test Result: Test passed.

Test 10 – ‘Data type’ combo box (Part Sorted)
Input: Select ‘part sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. The first half of the data is sorted, while the remainder is random data, in the animation screen.
Actual Output: The user is moved to the single animation screen. The first half of the data is sorted, while the remainder is random data, in the animation screen.
Test Result: Test passed.

Test 11 – ‘Data type’ combo box (Sorted)
Input: Select ‘sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. The data is pre-sorted in the animation screen.
Actual Output: The user is moved to the single animation screen. The data is pre-sorted in the animation screen.
Test Result: Test passed.

Test 12 – ‘No. elements’ combo box (5)
Input: Select ‘5’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. There are 5 elements of data in the animation panel.
Actual Output: The user is moved to the single animation screen. There are 5 elements of data in the animation panel.
Test Result: Test passed

Test 13 – ‘No. elements’ combo box (10)
Input: Select ‘10’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. There are 10 elements of data in the animation panel.
Actual Output: The user is moved to the single animation screen. There are 10 elements of data in the animation panel.
Test Result: Test passed

Test 14 – ‘No. elements’ combo box (20)
Input: Select ‘20’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. There are 20 elements of data in the animation panel.
Actual Output: The user is moved to the single animation screen. There are 20 elements of data in the animation panel.
Test Result: Test passed

Test 15 – ‘No. elements’ combo box (50)
Input: Select ‘50’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the single animation screen. There are 50 elements of data in the animation panel.
Actual Output: The user is moved to the single animation screen. There are 50 elements of data in the animation panel.
Test Result: Test passed

Test 16 – ‘Start Animation’ button
Input: Select ‘start animation’ button
Expected Output: The user is moved to the single animation screen.
Actual Output: The user is moved to the single animation screen.
Test Result: Test passed

Test 17 – ‘Cancel’ button
Input: Select ‘cancel’ button.
Expected Output: The user is moved to the welcome screen.
Actual Output: The user is moved to the welcome screen.
Test Result: Test passed

B.9 Algorithm race setup screen tests

System Area: Functionality of the ‘algorithm race setup’ screen.

Specification:

The algorithm race setup screen allows the user to specify the characteristics of the animations they are setting up by using the combo boxes.

Input: Selecting an option from any of the combo boxes, then selecting the ‘start’ button.

Output: The selected information from the combo boxes will be displayed in the relevant places on the ‘main screen’ in algorithm race format.

Test Cases:

Test 1 – ‘Animation type’ combo box (stack)
Input: ‘Stack’ selected from the ‘Animation type’ combo box, then press ‘start animation’ button.
Expected Output: All data is displayed in the stack view in the ‘algorithm race screen’.
The user is moved to the ‘algorithm race screen’.
Actual Output: All data is displayed in the stack view in the ‘algorithm race screen’. The user is moved to the ‘algorithm race screen’.
Test Result: Test passed.

Test 2 – ‘Animation type’ combo box (block)
Input: ‘Block’ selected from the ‘Animation type’ combo box, then press ‘start animation’ button.
Expected Output: All data is displayed in the block view in the ‘algorithm race screen’.
The user is moved to the ‘algorithm race screen’.
Actual Output: All data is displayed in the block view in the ‘algorithm race screen’. The user is moved to the ‘algorithm race screen’.
Test Result: Test passed.

Test 3 – ‘Algorithm type 1’ combo box (Bubble Sort)
Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Bubble Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Bubble Sort.
Test Result: Test passed.

Test 4 – ‘Algorithm type 1’ combo box (Quick Sort)
Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Quick Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Quick Sort.
Test Result: Test passed.

Test 5 – ‘Algorithm type 1’ combo box (Insertion Sort)
Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Insertion Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Insertion Sort.
Test Result: Test passed.

Test 6 – ‘Algorithm type 1’ combo box (Selection Sort)
Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Selection Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Selection Sort.
Test Result: Test passed.

Test 7 – ‘Algorithm type 1’ combo box (Heap Sort)
Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Heap Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel1 is sorted using Heap Sort.

Test Result: Test passed.

Test 8 – ‘Algorithm type 2’ combo box (Bubble Sort)
Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Bubble Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Bubble Sort.

Test Result: Test passed.

Test 9 – ‘Algorithm type 2’ combo box (Quick Sort)
Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Quick Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Quick Sort.

Test Result: Test passed.

Test 10 – ‘Algorithm type 2’ combo box (Insertion Sort)
Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Insertion Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Insertion Sort.

Test Result: Test passed.

Test 11 – ‘Algorithm type 2’ combo box (Selection Sort)
Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Selection Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Selection Sort.
Test Result: Test passed.

Test 12 – ‘Algorithm type 2’ combo box (Heap Sort)
Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Heap Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel2 is sorted using Heap Sort.
Test Result: Test passed.

Test 13 – ‘Algorithm type 3’ combo box (Bubble Sort)
Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Bubble Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Bubble Sort.
Test Result: Test passed.

Test 14 – ‘Algorithm type 3’ combo box (Quick Sort)
Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Quick Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Quick Sort.
Test Result: Test passed.

Test 15 – ‘Algorithm type 3’ combo box (Insertion Sort)
Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Insertion Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Insertion Sort.
Test Result: Test passed.
Test 16 – ‘Algorithm type 3’ combo box (Selection Sort)
Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Selection Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Selection Sort.
Test Result: Test passed.

Test 17 – ‘Algorithm type 3’ combo box (Heap Sort)
Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Heap Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel3 is sorted using Heap Sort.
Test Result: Test passed.

Test 18 – ‘Algorithm type 4’ combo box (Bubble Sort)
Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Bubble Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Bubble Sort.
Test Result: Test passed.

Test 19 – ‘Algorithm type 4’ combo box (Quick Sort)
Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Quick Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Quick Sort.
Test Result: Test passed.

Test 20 – ‘Algorithm type 4’ combo box (Insertion Sort)
Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Insertion Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Insertion Sort.

Test Result: Test passed.

**Test 21 – ‘Algorithm type 4’ combo box (Selection Sort)**

Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Selection Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Selection Sort.

Test Result: Test passed.

**Test 22 – ‘Algorithm type 4’ combo box (Heap Sort)**

Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Heap Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel4 is sorted using Heap Sort.

Test Result: Test passed.

**Test 23 – ‘Algorithm type 5’ combo box (Bubble Sort)**

Input: Select ‘Bubble Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Bubble Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Bubble Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Bubble Sort.

Test Result: Test passed.

**Test 24 – ‘Algorithm type 5’ combo box (Quick Sort)**

Input: Select ‘Quick Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Quick Sort.
Actual Output: The user is moved to the algorithm race screen. Details of Quick Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Quick Sort.
Test Result: Test passed.

**Test 25 – ‘Algorithm type 5’ combo box (Insertion Sort)**

Input: Select ‘Insertion Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Insertion Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Insertion Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Insertion Sort.
Test Result: Test passed.

**Test 26 – ‘Algorithm type 5’ combo box (Selection Sort)**

Input: Select ‘Selection Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Selection Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Selection Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Selection Sort.
Test Result: Test passed.

**Test 27 – ‘Algorithm type 5’ combo box (Heap Sort)**

Input: Select ‘Heap Sort’ from the ‘Algorithm type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Heap Sort.

Actual Output: The user is moved to the algorithm race screen. Details of Heap Sort are placed in the description box in the algorithm race screen. When the animation is started in the algorithm race view the data in panel5 is sorted using Heap Sort.
Test Result: Test passed.

**Test 28 – ‘Data type’ combo box (Random)**

Input: Select ‘random’ from the ‘Data type’ combo box, then press ‘start animation’ button.

Expected Output: The user is moved to the algorithm race screen. The data is order randomly in the animation screen.

Actual Output: The user is moved to the algorithm race screen. The data is order randomly in the animation screen.
Test Result: Test passed.
Test 29 – ‘Data type’ combo box (Reverse Sorted)
Input: Select ‘reverse sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. The data is reverse sorted in the animation screen.
Actual Output: The user is moved to the algorithm race screen. The data is reverse sorted in the animation screen.
Test Result: Test passed.

Test 30 – ‘Data type’ combo box (Part Sorted)
Input: Select ‘part sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. The first half of the data is sorted, while the remainder is random data, in the animation screen.
Actual Output: The user is moved to the algorithm race screen. The first half of the data is sorted, while the remainder is random data, in the animation screen.
Test Result: Test passed.

Test 31 – ‘Data type’ combo box (Sorted)
Input: Select ‘sorted’ from the ‘Data type’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. The data is pre-sorted in the animation screen.
Actual Output: The user is moved to the algorithm race screen. The data is pre-sorted in the animation screen.
Test Result: Test passed.

Test 32 – ‘No. elements’ combo box (5)
Input: Select ‘5’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. There are 5 elements of data in the animation panel.
Actual Output: The user is moved to the algorithm race screen. There are 5 elements of data in the animation panel.
Test Result: Test passed.

Test 33 – ‘No. elements’ combo box (10)
Input: Select ‘10’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. There are 10 elements of data in the animation panel.
Actual Output: The user is moved to the algorithm race screen. There are 10 elements of data in the animation panel.
Test Result: Test passed.

Test 34 – ‘No. elements’ combo box (20)
Input: Select ‘20’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. There are 20 elements of data in the animation panel.
Actual Output: The user is moved to the algorithm race screen. There are 20 elements of data in the animation panel.
Test Result: Test passed

Test 35 – ‘No. elements’ combo box (50)
Input: Select ‘50’ from the ‘No. elements’ combo box, then press ‘start animation’ button.
Expected Output: The user is moved to the algorithm race screen. There are 50 elements of data in the animation panel.
Actual Output: The user is moved to the algorithm race screen. There are 50 elements of data in the animation panel.
Test Result: Test passed

Test 36 – ‘Start Animation’ button
Input: Select ‘start animation’ button
Expected Output: The user is moved to the algorithm race screen.
Actual Output: The user is moved to the algorithm race screen.
Test Result: Test passed

Test 37 – ‘Cancel’ button
Input: Select ‘cancel’ button.
Expected Output: The user is moved to the welcome screen.
Actual Output: The user is moved to the welcome screen.
Test Result: Test passed

B.10 Single animation screen tests

System Area: Functionality of the single animation screen

Specification:

The main screen can show a single sorting algorithm animating. Once the animation has finished the elements of the array will be in ascending order. The user can also return to the welcome screen using the cancel button.

Input: Selecting run will start the animation, selecting cancel should take user to the welcome screen.

Output: The data will be sorted, with the relevant text boxes being updated to show how long the animation has taken, how many swaps have occurred and how many comparisons have occurred. If user selects ‘cancel’ they should be taken to the ‘welcome screen’.

Test Cases:

Test 1 – ‘Run’ button.
Input: User selects ‘run’ with the mouse.
Expected Output: Algorithm animation starts and sorts the data in to ascending order. The text boxes are updated to display how many swaps and comparisons have occurred, how
long the sorting has taken, and that sorting is completed. The elements that are sorted should also have changed colour to indicate that they have been sorted.
Actual Output: Algorithm animation starts and sorts the data into ascending order. The text boxes are updated to display how many swaps and comparisons have occurred, how long the sorting has taken, and that sorting is completed. The elements that are sorted should also have changed colour to indicate that they have been sorted.
Test Result: Test passed

Test 2 – ‘Cancel button.
Input: User selects ‘cancel’ with the mouse.
Expected Output: User moved to the welcome screen.
Actual Output: User moved to the welcome screen.
Test Result: Test passed

B.11 Animation race screen tests

System Area: Functionality of the algorithm race screen

Specification:

The main screen can show multiple sorting algorithms animating simultaneously. Once the animations have finished the elements of the arrays will be in ascending order. The user can also return to the welcome screen using the cancel button.

Input: Selecting run will start the animations, selecting cancel should take user to the welcome screen.

Output: The data will be sorted, with the relevant text boxes being updated to show how long the animations have taken, how many swaps have occurred and how many comparisons have occurred. The rank text box will indicate the order in which the sorting algorithm finished. If user selects ‘cancel’ they should be taken to the ‘welcome screen’.

Test Cases:

Test 1 – ‘Run’ button.
Input: User selects ‘run’ with the mouse.
Expected Output: Algorithm animations start and sort the data into ascending order. The text boxes are updated to display how many swaps and comparisons have occurred, how long the sorting has taken, and that sorting is completed. The rank text boxes are also updated to display the order in which the sorting algorithms finished. The elements that are sorted have changed colour to indicate that they have been sorted.
Actual Output: Algorithm animations start and sort the data into ascending order. The text boxes are updated to display how many swaps and comparisons have occurred, how long the sorting has taken, and that sorting is completed. The rank text boxes are also updated to display the order in which the sorting algorithms finished. The elements that are sorted have changed colour to indicate that they have been sorted.
Test Result: Test passed
Test 2 – ‘Cancel button.
Input: User selects ‘cancel’ with the mouse.
Expected Output: User moved to the welcome screen.
Actual Output: User moved to the welcome screen.
Test Result: Test passed

B.12 Animation control tests

System Area: Functionality of the animation controls

Specification:

The animation controls allow the user to specify the speed of the animation, as well as allowing the animation to move through one step at a time in a walk-through mode.

Input: Selecting faster or slower will have the obvious effects. The speed bar can also be manipulated to adjust the speed. Pressing the walk-through button will cause the animation to enter walk-through mode.

Output: The speed of the animation will change depending on the button selected. Walk-through mode may be entered.

Test 1 – ‘Walkthrough’ button.
Input: User selects ‘walkthrough’ with the mouse.
Expected Output: Animation changes speed so that it moves through its steps one by one.
Actual Output: Nothing happens.
Test Result: Test failed.

Test 2 – ‘Speed’ scroll bar.
Input: User modifies the speed bar to indicate an increase, or decrease, in the speed of the animation.
Expected Output: Speed of animation increases or decreases as relevant.
Actual Output: Nothing happens.
Test Result: Test failed.

Test 3 – ‘Faster’ button.
Input: User selects ‘faster’ with the mouse.
Expected Output: Animation changes speed so that the animation executes faster.
Actual Output: Nothing happens.
Test Result: Test failed.

Test 4 – ‘Slower’ button.
Input: User selects ‘slower’ with the mouse.
Expected Output: Animation changes speed so that the animation executes slower.
Actual Output: Nothing happens.
Test Result: Test failed.
Appendix C – System Assessment Surveys On the Developed System

C.1 Acceptance testing
The usability testing of the system was conducted using a modified version of the system assessment surveys previously used to assess the usability of existing systems in the requirements analysis section. As before, the goal was to rate the system in terms of usability and ease of learning. The usability of the system was judged against the same six HCI factors:

- System consistency.
- Simplicity.
- Clarity.
- Help and support.
- Menu paths.
- Layout.

The other major goal of the survey was to assess how effective the system was in terms of aiding the users in their learning of the sorting algorithms. As before, a number of questions were asked that attempted to discover how successful the users felt the animations were in teaching them how the sorting algorithms work.

A total of twenty users were chosen to assess the system. All of these users were in the target group that the system is aimed at. The users were asked to judge each aspect of the system on the Likert scale, with each factor being measure between one and five, with one being excellent, three being average, and five being very poor.

C.2 System assessment 1

C.2.1 HCI factors

**Consistency**
How consistent are the icons within the system? 1

**Simplicity**
How easy to understand is the text and the icons used in the system? 1

**Clarity**
How clear is the purpose of each screen in the system? 1

**Help and support**
How detailed, relevant and accurate is the help provided by the system? 2

**Menu paths**
Are the menu paths easy to navigate and intuitive? 1

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 2
C.2.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the animation and additional information provided in the system?  
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?  
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?  
Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?  
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?  
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?  

C.2.3 Comments

The system was extremely easy to use, despite its dull colour scheme. The animations were impossible to see but I was able to learn about the sorting algorithms due to the detailed explanations of how each algorithm worked, as well as the statistics generated by the sorting algorithms.

C.3 System Assessment 2

C.3.1 HCI factors

Consistency
How consistent are the icons within the system?  
Simplicity
How easy to understand is the text and the icons used in the system?  
Clarity
How clear is the purpose of each screen in the system?  
Help and support
How detailed, relevant and accurate is the help provided by the system?
Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.3.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.3.3 Comments

The system was easy to use, although the layout didn’t look tidy on occasions. The different views seemed to offer no real advantage over each other, but the descriptions made it possible to understand how the sorting algorithms worked.

C.4 System Assessment 3

C.4.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?
Clarity
How clear is the purpose of each screen in the system? 1

Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 1

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 2

C.4.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

C.4.3 Comments

Sorting algorithms were understandable after reading the text thoroughly. System as a whole was quite usable.

C.5 System Assessment 4

C.5.1 HCI factors
Consistency
How consistent are the icons within the system? 1

Simplicity
How easy to understand is the text and the icons used in the system? 3

Clarity
How clear is the purpose of each screen in the system? 2

Help and support
How detailed, relevant and accurate is the help provided by the system? 3

Menu paths
Are the menu paths easy to navigate and intuitive? 1

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 3

C.5.2  Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

C.5.3  Comments

System was good although the layout sometimes seemed slightly inconsistent. Sorting algorithms relied heavily on the text to be understood.
C.6 System Assessment 5

C.6.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?

Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.6.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?
C.6.3 Comments

System was fine, and helped me build on my previous understanding of sorting algorithms.

C.7 System Assessment 6

C.7.1 HCI factors

Consistency
How consistent are the icons within the system? 1

Simplicity
How easy to understand is the text and the icons used in the system? 2

Clarity
How clear is the purpose of each screen in the system? 1

Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 1

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 2

C.7.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.7.3 Comments

System was good in usability terms, but needed more animation controls to make the animations of any real use to learning sorting algorithms.

C.8 System Assessment 7

C.8.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?

Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.8.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.8.3 Comments

Usability of the system was excellent, algorithms were learnable due to the detailed text descriptions.

C.9 System Assessment 8

C.9.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?

Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.9.2 Algorithm animation
Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 5

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 4

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 5

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 4

C.9.3 Comments
The usability of the system was generally very good. Quick sort and Heap sort was difficult to understand, even with the descriptions but the rest of the sorting algorithms were fine.

C.10 System Assessment 9

C.10.1 HCI factors
Consistency
How consistent are the icons within the system? 2

Simplicity
How easy to understand is the text and the icons used in the system? 1

Clarity
How clear is the purpose of each screen in the system? 2

Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 1
Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.10.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.10.3 Comments

The system needs to have more than text descriptions to allow users with no knowledge of sorting algorithms to learn about them.

C.11 System Assessment 10

C.11.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?
Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 1

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 2

C.11.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

C.11.3 Comments
Usability of the system was very good, and the algorithms were fairly easy to understand due to the text descriptions provided.

C.12 System Assessment 11

C.12.1 HCI factors

Consistency
How consistent are the icons within the system? 3
**Simplicity**
How easy to understand is the text and the icons used in the system?

**Clarity**
How clear is the purpose of each screen in the system?

**Help and support**
How detailed, relevant and accurate is the help provided by the system?

**Menu paths**
Are the menu paths easy to navigate and intuitive?

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

**C.12.2 Algorithm animation**

**Stack view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

**Block view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

**C.12.3 Comments**

Animation was impossible to view, which meant I had to rely on the text to understand how the sorting algorithms worked. Although this was good I had hoped that the animation would be the main method of learning.
## C.13 System Assessment 12

### C.13.1 HCI factors

<table>
<thead>
<tr>
<th>Consistency</th>
<th>How consistent are the icons within the system?</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simplicity</td>
<td>How easy to understand is the text and the icons used in the system?</td>
<td>2</td>
</tr>
<tr>
<td>Clarity</td>
<td>How clear is the purpose of each screen in the system?</td>
<td>3</td>
</tr>
<tr>
<td>Help and support</td>
<td>How detailed, relevant and accurate is the help provided by the system?</td>
<td>2</td>
</tr>
<tr>
<td>Menu paths</td>
<td>Are the menu paths easy to navigate and intuitive?</td>
<td>2</td>
</tr>
<tr>
<td>Layout</td>
<td>Do the components in the system have sufficient space so that they can be easily identified and easily selected?</td>
<td>2</td>
</tr>
</tbody>
</table>

### C.13.2 Algorithm animation

<table>
<thead>
<tr>
<th>Stack view</th>
<th>How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?</td>
<td>2</td>
</tr>
<tr>
<td>Block view</td>
<td>How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?</td>
<td>2</td>
</tr>
</tbody>
</table>
C.13.3 Comments

Using this system was simple, and since I already had some knowledge of the sorting algorithms the text reinforced what I had previously learnt.

C.14 System Assessment 13

C.14.1 HCI factors

**Consistency**
How consistent are the icons within the system?

2

**Simplicity**
How easy to understand is the text and the icons used in the system?

3

**Clarity**
How clear is the purpose of each screen in the system?

2

**Help and support**
How detailed, relevant and accurate is the help provided by the system?

2

**Menu paths**
Are the menu paths easy to navigate and intuitive?

1

**Layout**
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

2

C.14.2 Algorithm animation

**Stack view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

3

**Block view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

3
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?  

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.14.3 Comments

Very usable system. The learning of how the sorting algorithms work was done almost entirely by reading the accompanying text, this seems a very good way of learning in comparisons with other systems.

C.15 System Assessment 14

C.15.1 HCI factors

Consistency
How consistent are the icons within the system?  

Simplicity
How easy to understand is the text and the icons used in the system?  

Clarity
How clear is the purpose of each screen in the system?  

Help and support
How detailed, relevant and accurate is the help provided by the system?  

Menu paths
Are the menu paths easy to navigate and intuitive?  

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?  

C.15.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?  

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 4

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 4

C.15.3 Comments
Fairly simple to use system but as with many other systems, the actual learning of how the sorting algorithms work is difficult.

C.16 System Assessment 15

C.16.1 HCI factors

Consistency
How consistent are the icons within the system? 2

Simplicity
How easy to understand is the text and the icons used in the system? 2

Clarity
How clear is the purpose of each screen in the system? 2

Help and support
How detailed, relevant and accurate is the help provided by the system? 1

Menu paths
Are the menu paths easy to navigate and intuitive? 2

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 3

C.16.2 Algorithm animation
**Stack view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

**Block view**
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

**C.16.3 Comments**

Although the system is usable the lack of animation controls makes it difficult for a user with very little knowledge of sorting algorithms to understand what is happening.

**C.17 System Assessment 16**

**C.17.1 HCI factors**

**Consistency**
How consistent are the icons within the system?

**Simplicity**
How easy to understand is the text and the icons used in the system?

**Clarity**
How clear is the purpose of each screen in the system?

**Help and support**
How detailed, relevant and accurate is the help provided by the system?

**Menu paths**
Are the menu paths easy to navigate and intuitive?
Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.17.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.17.3 Comments

For the system to help the user it needs to allow more control over the execution of the animation.

C.18 System Assessment 17

C.18.1 HCI factors

Consistency
How consistent are the icons within the system?

Simplicity
How easy to understand is the text and the icons used in the system?

Clarity
How clear is the purpose of each screen in the system?
Help and support
How detailed, relevant and accurate is the help provided by the system?

Menu paths
Are the menu paths easy to navigate and intuitive?

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

C.18.2 Algorithm Animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.18.3 Comments

The system was symbol and straightforward, although on occasions the text seemed to be out of proportion to the page. The sorting algorithms were fairly easy to learn mainly due to the text. The animations didn’t really help.

C.19.1 HCI factors

Consistency
How consistent are the icons within the system?
Simplicity
How easy to understand is the text and the icons used in the system? 1

Clarity
How clear is the purpose of each screen in the system? 1

Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 2

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 1

C.19.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 4
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 4
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3
How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 3

C.19.3 Comments
The animation controls didn’t work which made the animations impossible to understand due to the speed of execution. This needs fixing for the system to be really useful
C.20 System Assessment 19

C.20.1 HCI factors

Consistency
How consistent are the icons within the system?

1

Simplicity
How easy to understand is the text and the icons used in the system?

1

Clarity
How clear is the purpose of each screen in the system?

1

Help and support
How detailed, relevant and accurate is the help provided by the system?

1

Menu paths
Are the menu paths easy to navigate and intuitive?

1

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected?

2

C.20.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

3

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system?

4

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system?

3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

3
C.20.3 Comments

The animations weren’t very helpful, but the text did give me some idea about how each sorting algorithm worked. The system was easy to use though.

C.21 System Assessment 20

C.21.1 HCI factors

Consistency
How consistent are the icons within the system? 2

Simplicity
How easy to understand is the text and the icons used in the system? 2

Clarity
How clear is the purpose of each screen in the system? 1

Help and support
How detailed, relevant and accurate is the help provided by the system? 2

Menu paths
Are the menu paths easy to navigate and intuitive? 2

Layout
Do the components in the system have sufficient space so that they can be easily identified and easily selected? 2

C.21.2 Algorithm animation

Stack view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 2

How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 3

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system? 2

Block view
How easy to understand is the sorting algorithm based on the Animation and additional information provided in the system? 2
How confident would you be at accurately describing the sorting algorithm shown after watching the animation once and looking at the additional information provided in the system? 

How confident would you be at accurately describing the sorting algorithm shown after watching the animation five times and looking at the additional information provided in the system?

C.21.3 Comments

The system made it obvious how to do tasks, and the help was useful, although it could have been more detailed. The animations weren’t especially useful, but the descriptions clearly stated how each sorting algorithm works.

Appendix D – Key Sections Of Code In The Developed System

D.1 Reverse sorting data

```java
// sort data
int x, y, index;
for (x=1; x < elements.length; x++) {
  index = elements[x];
  y = x;
  while ((y > 0) && (elements[y - 1] > index)) {
    elements[y] = elements[y-1];
    y = y-1;
  }
  elements[y] = index;
}
// now swap so its reverse sorted
int counta = 0;
int countb = elements.length - 1;
int holda;
int holdb;
```

D.2 Pre-sorting data

```java
// sort data
int x, y, index;
for (x=1; x < elements.length; x++) {
  index = elements[x];
  y = x;
  while ((y > 0) && (elements[y - 1] > index)) {
    elements[y] = elements[y-1];
    y = y-1;
  }
  elements[y] = index;
}
```
D.3 Partially sorting data

// part sort data
int x, y, index;
for (x=1; x < elements.length/2; x++) {
    index = elements[x];
    y = x;
    while ((y > 0) && (elements[y - 1] > index)) {
        elements[y] = elements[y-1];
        y = y-1;
    }
    elements[y] = index;
}

D.4 Bubble Sort method

public void BubSort() {
    java.awt.GridBagConstraints gridBagConstraints2;
    gridBagConstraints2 = new java.awt GridBagConstraints();
    gridBagConstraints2.gridx = 0;
    gridBagConstraints2.gridy = 1;

    comparisons = 0;
    swaps = 0;

    boolean doMore = true;
    int bubibubtemp;

    long starttime = new GregorianCalendar().getTimeInMillis();

    while (doMore) {
        doMore = false;
        for (bubi =0; bubibubi < elements.length - 1; bubibubi++) {
            comparisons ++;
            if (elements[bubi] > elements[bubi + 1]) {
                bubibubtemp = elements[bubi];
                elements[bubi] = elements[bubi + 1];
                elements[bubi + 1] = bubibubtemp;
                doMore = true;
                swaps ++;
            }
        }
    }

    long finishtime = new GregorianCalendar().getTimeInMillis();

    String swaps2 = Integer.toString(swaps);
    String comps2 = Integer.toString(comparisons);
    twojTextArea9.setText(swaps2);
    twojTextArea4.setText(comps2);
    long timetosort = finishtime - starttime;
    twojTextArea5.setText("" + timetosort);
    twojTextArea3.setText("Sorting completed");
}
D.5 Heap Sort method

public void HeapSort() {
    comparisons = 0;
    swaps = 0;

    java.awt.GridBagConstraints gridBagConstraints2 = new java.awt.GridBagConstraints();
    gridBagConstraints2.gridx = 0;
    gridBagConstraints2.gridy = 1;

    int length = elements.length;
    long starttime = new GregorianCalendar().getTimeInMillis();

    for (int k = length/2; k > 0; k--) {
        do {
            int temp = elements[0];
            elements[0] = elements[length - 1];
            elements[length - 1] = temp;
            length = length - 1;
            downheap(elements, 1, length);
        } while (length > 1);

        String swaps2 = Integer.toString(swaps);
        String comps2 = Integer.toString(comparisons);

        twojTextArea9.set_text(swaps2);
        twojTextArea9.set_text(comps2);

        long finishtime = new GregorianCalendar().getTimeInMillis();
        long timetosort = finishtime - starttime;

        twojTextArea5.set_text("" + timetosort);
        twojTextArea3.set_text("Sorting completed");
    }
}

D.6 Quick Sort method

public void QuickSort() {
    comparisons = 0;
    swaps = 0;

    java.awt.GridBagConstraints gridBagConstraints2 = new java.awt.GridBagConstraints();
    gridBagConstraints2.gridx = 0;
    gridBagConstraints2.gridy = 1;

    int length = elements.length;
    long starttime = new GregorianCalendar().getTimeInMillis();

    qsort(elements, 0, elements.length - 1);
long finishTime = new GregorianCalendar().getTimeInMillis();
String swaps2 = Integer.toString(swaps);
String comps2 = Integer.toString(comparisons);

twojTextArea9.setText(swaps2);
twojTextArea4.setText(comps2);

long timetosort = finishTime - startTime;
twojTextArea5.setText("" + timetosort);
twojTextArea3.setText("Sorting completed");

public void qsort(int[] elements, int lo0, int hi0) {
    int lo = lo0;
    int hi = hi0;

    comparisons ++;

    if (lo > hi) {
        return;
    }
    else if (lo == hi - 1) {
        comparisons ++;
        if (elements[lo] > elements[hi]) {
            int temp = elements[lo];
            elements[lo] = elements[hi];
            elements[hi] = temp;
            swaps ++;
        }
        return;
    }
    int pivot = elements[(lo + hi)/2];
    elements[(lo + hi) / 2] = elements[hi];
    elements[hi] = pivot;
    comparisons ++;
    while((lo < hi) {
        comparisons ++;
        while((elements[lo] <= pivot) && (lo < hi)) {
            lo ++;
            comparisons ++;
        }
        while((pivot <= elements[hi]) && (lo < hi)) {
            hi --;
            comparisons ++;
        }
        if (lo < hi) {
            int temp = elements[lo];
            elements[lo] = elements[hi];
            elements[hi] = temp;
            swaps ++;
            comparisons ++;
        }
        elements[hi0] = elements[hi];
        elements[hi] = pivot;
        qsort(elements, lo0, lo - 1);
        qsort(elements, hi + 1, hi0);
    }
D.7 Selection Sort method

```java
public void SelectionSort() {
    comparisons = 0;
    swaps = 0;

    java.awt.GridBagConstraints gridBagConstraints2 = new java.awt.GridBagConstraints();
    gridBagConstraints2.gridx = 0;
    gridBagConstraints2.gridy = 1;

    int seli, selj, temp;
    long starttime = new GregorianCalendar().getTimeInMillis();
    for (seli = 0; seli < elements.length; seli++) {
        for (selj = seli + 1; selj < elements.length; selj++) {
            comparisons++;
            if (elements[seli] > elements[selj]) {
                temp = elements[seli];
                elements[seli] = elements[selj];
                elements[selj] = temp;
                swaps++;
            }
        }
    }
    long finishtime = new GregorianCalendar().getTimeInMillis();
    long timetosort = finishtime - starttime;

    String swaps2 = Integer.toString(swaps);
    String comps2 = Integer.toString(comparisons);

    twojTextArea9.setText(swaps2);
    twojTextArea4.setText(comps2);
}
```

D.8 Insertion Sort method

```java
public void InsertionSort() {
    comparisons = 0;
    swaps = 0;

    java.awt.GridBagConstraints gridBagConstraints2 = new java.awt.GridBagConstraints();
    gridBagConstraints2.gridx = 0;
    gridBagConstraints2.gridy = 1;

    long starttime = new GregorianCalendar().getTimeInMillis();
    for (int i = 1; i < elements.length; i++) {
        int j = i;
        int B = elements[i];
        comparisons++;
        while ((j > 0) && (elements[j - 1] > B)) {
            comparisons++;
            elements[j] = elements[j - 1];
```
j--;
}
elements[j] = B;
swaps++;
}

long finishtime = new GregorianCalendar().getTimeInMillis();
long timetosort = finishtime - starttime;

String swaps2 = Integer.toString(swaps);
String comps2 = Integer.toString(comparisons);

twojTextArea9.setText(swaps2);
twojTextArea4.setText(comps2);

D.9 Draw stack method for five items in the single animation screen

public void draw5stack(GridBagConstraints g) {
    GridBagConstraints grid = g;
    GridLayout layout = new GridLayout(1,5);
    twojPanelNew.setLayout(layout);

    int count = 0;
    int value = (elements[count])/4;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew1.setRows(value);
    twojTextAreaNew1.setColumns(1);
    twojPanelstack1.add(twojTextAreaNew1);
    twojPanelNew.add(twojPanelstack1);

    count = 1;
    value = (elements[count])/4;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew2.setRows(value);
    twojTextAreaNew2.setColumns(1);
    twojPanelstack2.add(twojTextAreaNew2);
    twojPanelNew.add(twojPanelstack2);

    count = 2;
    value = (elements[count])/4;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew3.setRows(value);
    twojTextAreaNew3.setColumns(1);
    twojPanelstack3.add(twojTextAreaNew3);
    twojPanelNew.add(twojPanelstack3);

    count = 3;
    value = (elements[count])/4;
    if (value < 1) {
        value = 1;
    }
D.10 Draw block method for five items in the single animation screen

public void draw5(GridBagConstraints g) {
    GridBagConstraints grid = g;
    GridLayout layout = new GridLayout(5,1);
    twojPanelNew.setLayout(layout);

    int count = 0;
    int value = (elements[count])/5;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew1.setRows(1);
    twojTextAreaNew1.setColumns(value);
    twojPanelstack1.add(twojTextAreaNew1);
    twojPanelNew.add(twojPanelstack1);

    count = 1;
    value = (elements[count])/5;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew2.setRows(1);
    twojTextAreaNew2.setColumns(value);
    twojPanelstack2.add(twojTextAreaNew2);
    twojPanelNew.add(twojPanelstack2);

    count = 2;
    value = (elements[count])/5;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew3.setRows(1);
    twojTextAreaNew3.setColumns(value);
    twojPanelstack3.add(twojTextAreaNew3);
    twojPanelNew.add(twojPanelstack3);

    count = 3;
    value = (elements[count])/5;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew4.setRows(1);
    twojTextAreaNew4.setColumns(value);
    twojPanelstack4.add(twojTextAreaNew4);
    twojPanelNew.add(twojPanelstack4);

    count = 4;
    value = (elements[count])/4;
    if (value < 1) {
        value = 1;
    }
    twojTextAreaNew5.setRows(value);
    twojTextAreaNew5.setColumns(1);
    twojPanelstack5.add(twojTextAreaNew5);
    twojPanelNew.add(twojPanelstack5);

    twojPanel3.add(twojPanelNew, grid);
}
value = (elements[count])/5;
if (value < 1) {
    value = 1;
}
twojTextAreaNew4.setRows(1);
twojTextAreaNew4.setColumns(value);
twojPanelstack4.add(twojTextAreaNew4);
twojPanelNew.add(twojPanelstack4);

count = 4;
value = (elements[count])/5;
if (value < 1) {
    value = 1;
}
twojTextAreaNew5.setRows(1);
twojTextAreaNew5.setColumns(value);
twojPanelstack5.add(twojTextAreaNew5);
twojPanelNew.add(twojPanelstack5);

twojPanel3.add(twojPanelNew, grid);
}

D.11 Draw stack method for five items in the algorithm race screen with two algorithms racing

private void multistack52multi(int flag){
    int test = flag;
    if (test == 1) {
        // for first array
        GridLayout layout = new GridLayout(6,1);
        multiPanel2.removeAll();
        multiPanel2.setVisible(false);
        multiPanel2.setLayout(layout);
        stacktext1.add(stacktextlabel1);
        multiPanel2.add(stacktext1);

        int count = 0;
        int value = (multielements1[count])/5;
        if (value < 1) {
            value = 1;
        }
        multiArea1.setRows(1);
        multiArea1.setColumns(value);
        p11.add(multiArea1);
        multiPanel2.add(p11);

        count = 1;
        value = (multielements1[count])/5;
        if (value < 1) {
            value = 1;
        }
        multiArea2.setRows(1);
        multiArea2.setColumns(value);
        p12.add(multiArea2);
        multiPanel2.add(p12);
        count = 2;
value = (multielements1[count])/5;
if (value < 1) {
    value = 1;
}
multiArea3.setColumns(value);
p13.add(multiArea3);
multijapnel2.add(p13);

count = 3;
value = (multielements1[count])/5;
if (value < 1) {
    value = 1;
}
multiArea4.setColumns(value);
p14.add(multiArea4);
multijapnel2.add(p14);

count = 4;
value = (multielements1[count])/5;
if (value < 1) {
    value = 1;
}
multiArea5.setColumns(value);
p15.add(multiArea5);
multijapnel2.add(p15);

multiArea1.setBackground(new java.awt.Color(0, 0, 0));
multiArea2.setBackground(new java.awt.Color(0, 0, 0));
multiArea3.setBackground(new java.awt.Color(0, 0, 0));
multiArea4.setBackground(new java.awt.Color(0, 0, 0));
multiArea5.setBackground(new java.awt.Color(0, 0, 0));
multiArea1.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea2.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea3.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea4.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea5.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multijapnel2.setVisible(true);
}
else if (test == 2) {

    // for second array
    GridLayout layout1 = new GridLayout(6,1);
multijapnel3.setLayout(layout1);
multijapnel3.setVisible(false);
multijapnel3.removeAll();
    stacktext2.add(stacktextlabel2);
multijapnel3.add(stacktext2);

    int count = 0;
    int value = (multielements2[count])/5;
    if (value < 1) {
        value = 1;
    }
    multiArea21.setRows(1);
multiArea21.setColumns(value);
p21.add(multiArea21);
multijapnel3.add(p21);

count = 1;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea22.setRows(1);
multiArea22.setColumns(value);
p22.add(multiArea22);
multijapnel3.add(p22);

count = 2;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea23.setRows(1);
multiArea23.setColumns(value);
p23.add(multiArea23);
multijapnel3.add(p23);

count = 3;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea24.setRows(1);
multiArea24.setColumns(value);
p24.add(multiArea24);
multijapnel3.add(p24);

count = 4;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea25.setRows(1);
multiArea25.setColumns(value);
p25.add(multiArea25);
multijapnel3.add(p25);

mawjTextField21.setText("Sorting Completed");
multijapnel3.setVisible(true);
multiArea21.setBackground(new java.awt.Color(0, 0, 0));
multiArea22.setBackground(new java.awt.Color(0, 0, 0));
multiArea23.setBackground(new java.awt.Color(0, 0, 0));
multiArea24.setBackground(new java.awt.Color(0, 0, 0));
multiArea25.setBackground(new java.awt.Color(0, 0, 0));
multiArea22.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea23.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea24.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea25.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));

D.12 Draw block method for five items in the algorithm race screen with two algorithms racing

private void multiblock52multi(int flag){
    int test = flag;
    if (test == 1) {
        // for first array
        GridLayout layout = new GridLayout(1,5);
        multijapnel2.setLayout(layout);
        multijapnel2.removeAll();
        multijapnel2.setVisible(false);
        int count = 0;
        int value = (multielements1[count])/5;
        if (value < 1) {
            value = 1;
        }
        multiArea1.setRows(value);
        multiArea1.setColumns(1);
        p11.add(multiArea1);
        count = 1;
        value = (multielements1[count])/5;
        if (value < 1) {
            value = 1;
        }
        multiArea2.setRows(value);
        multiArea2.setColumns(1);
        p12.add(multiArea2);
        multijapnel2.add(p12);
        count = 2;
        value = (multielements1[count])/5;
        if (value < 1) {
        ...
        }
    }
}
value = 1;
}
multiArea3.setRows(value);
multiArea3.setColumns(1);
p13.add(multiArea3);

multiArea3.setRows(value);
multiArea3.setColumns(1);
p13.add(multiArea3);

count = 3;
value = (multielements1[count])/5;
if (value < 1) {
    value = 1;
}
multiArea4.setRows(value);
multiArea4.setColumns(1);
p14.add(multiArea4);

multiArea4.setRows(value);
multiArea4.setColumns(1);
p14.add(multiArea4);

count = 4;
value = (multielements1[count])/5;
if (value < 1) {
    value = 1;
}
multiArea5.setRows(value);
multiArea5.setColumns(1);
p15.add(multiArea5);

multiArea5.setRows(value);
multiArea5.setColumns(1);
p15.add(multiArea5);

multiArea1.setBackground(new java.awt.Color(0, 0, 0));
multiArea2.setBackground(new java.awt.Color(0, 0, 0));
multiArea3.setBackground(new java.awt.Color(0, 0, 0));
multiArea4.setBackground(new java.awt.Color(0, 0, 0));
multiArea5.setBackground(new java.awt.Color(0, 0, 0));
multiArea1.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea2.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea3.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea4.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea5.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea2.setVisible(true);
}
else if (test == 2) {
    // for second array
    GridLayout layout1 = new GridLayout(1,51);
multipanel3.setLayout(layout1);
multipanel3.setVisible(false);
multipanel3.removeAll();

    int count = 0;
    int value = (multielements2[count])/5;
    if (value < 1) {
        value = 1;
    }
multiArea21.setRows(value);
multiArea21.setColumns(1);
p21.add(multiArea21);
multipanel3.add(p21);

    count = 1;
    value = (multielements2[count])/5;
    if (value < 1) {
value = 1;
}
multiArea22.setColumns(1);
multiArea22.setColumns(1);
p22.add(multiArea22);
multipanel3.add(p22);

count = 2;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea23.setColumns(value);
multiArea23.setColumns(value);
p23.add(multiArea23);
multipanel3.add(p23);

count = 3;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea24.setColumns(value);
multiArea24.setColumns(value);
p24.add(multiArea24);
multipanel3.add(p24);

count = 4;
value = (multielements2[count])/5;
if (value < 1) {
    value = 1;
}
multiArea25.setColumns(value);
multiArea25.setColumns(value);
p25.add(multiArea25);
multipanel3.add(p25);
multipanel3.setVisible(true);

mawjTextField21.setText("Sorting Completed");
multiArea11.setBackground(new java.awt.Color(0, 0, 0));
multiArea21.setBackground(new java.awt.Color(0, 0, 0));
multiArea22.setBackground(new java.awt.Color(0, 0, 0));
multiArea23.setBackground(new java.awt.Color(0, 0, 0));
multiArea24.setBackground(new java.awt.Color(0, 0, 0));
multiArea25.setBackground(new java.awt.Color(0, 0, 0));
multiArea211.setBackground(new java.awt.Color(0, 0, 0));
multiArea22.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea23.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea24.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));
multiArea25.setBorder(new javax.swing.border.LineBorder(new java.awt.Color(255, 255, 255)));

int test1 = Integer.parseInt(mawjTextField9.getText());
int test2 = Integer.parseInt(mawjTextField10.getText());
if (test1 == test2) {
D.13 Welcome screen initialisation method

private void initWelcomeComponents() {
    java.awt.GridBagConstraints gridBagConstraints;
    welcomejPanel1 = new javax.swing.JPanel();
    welcomejPanel3 = new javax.swing.JPanel();
    welcomejLabel1 = new javax.swing.JLabel();
    welcomejPanel4 = new javax.swing.JPanel();
    welcomejLabel2 = new javax.swing.JLabel();
    welcomejLabel3 = new javax.swing.JLabel();
    welcomejMenuBar1 = new javax.swing.JMenuBar();
    welcomejMenu1 = new javax.swing.JMenu();
    welcomejMenu7 = new javax.swing.JMenu();
    welcomejMenu2 = new javax.swing.JMenu();
    welcomejMenu4 = new javax.swing.JMenu();
    welcomejMenu3 = new javax.swing.JMenu();
    welcomejMenu5 = new javax.swing.JMenu();
    welcomejMenu6 = new javax.swing.JMenu();

    getContentPane().setLayout(new java.awt.GridLayout(1, 1));
    welcomejPanel1.setLayout(new java.awt.GridBagLayout());
    welcomejPanel1.setBorder(new javax.swing.border.EtchedBorder());
    welcomejPanel3.setLayout(new java.awt.GridBagLayout());
    welcomejPanel3.setBorder(new javax.swing.border.EtchedBorder());
    welcomejPanel1.add(welcomejLabel1, gridBagConstraints);
    welcomejPanel3.add(welcomejLabel1, gridBagConstraints);
    welcomejPanel1.add(welcomejPanel4, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu1, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu7, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel2, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel3, gridBagConstraints);
    welcomejPanel1.setLayout(new java.awt.GridBagLayout());
    welcomejPanel4.setLayout(new java.awt.GridBagLayout());
    welcomejPanel1.add(welcomejPanel4, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu1, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu7, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel2, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu2, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu4, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu5, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu6, gridBagConstraints);
    welcomejPanel1.add(welcomejPanel4, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu1, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu7, gridBagConstraints);
    welcomejPanel1.add(welcomejPanel4, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu1, gridBagConstraints);
    welcomejPanel1.add(welcomejMenu7, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel2, gridBagConstraints);
    welcomejPanel4.add(welcomejLabel3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu2, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu4, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu5, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu6, gridBagConstraints);
    welcomejLabel1.setFont(new java.awt.Font("Dialog", 1, 30));
    welcomejLabel1.setText("Welcome");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.insets = new java.awt.Insets(5, 0, 5, 0);
    welcomejPanel3.add(welcomejLabel1, gridBagConstraints);
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.fill = java.awt.GridBagConstraints.HORIZONTAL;
    welcomejPanel1.add(welcomejLabel1, gridBagConstraints);
    welcomejPanel4.add(welcomejPanel3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu2, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu4, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu3, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu5, gridBagConstraints);
    welcomejPanel4.add(welcomejMenu6, gridBagConstraints);
    welcomejLabel1.setFont(new java.awt.Font("Dialog", 1, 30));
    welcomejLabel2.setFont(new java.awt.Font("Dialog", 1, 30));
    welcomejLabel4.setFont(new java.awt.Font("Dialog", 1, 30));
    welcomejLabel2.setText("Please make a selection");
    gridBagConstraints = new java.awt.GridBagConstraints();
    gridBagConstraints.gridx = 0;
    gridBagConstraints.gridy = 0;
    gridBagConstraints.insets = new java.awt.Insets(5, 5, 5);
welcomejPanel4.add(welcomejLabel2, gridBagConstraints);
welcomejLabel4.setText("the top of the screen");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 2;
gridBagConstraints.insets = new java.awt.Insets(0, 0, 5, 0);
welcomejPanel4.add(welcomejLabel4, gridBagConstraints);
welcomejPanel3.setText("from the menu bar at");
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
gridBagConstraints.insets = new java.awt.Insets(0, 0, 5, 0);
welcomejPanel4.add(welcomejLabel3, gridBagConstraints);
gridBagConstraints = new java.awt.GridBagConstraints();
gridBagConstraints.gridx = 0;
gridBagConstraints.gridy = 1;
welcomejPanel4.add(welcomejLabel3, gridBagConstraints);
welcomejPanel1.add(welcomejPanel4, gridBagConstraints);
welcomePanel1.setSize(400, 400);
getContentPanel().add(welcomejPanel1, new java.awt.GridBagConstraints());
welcomejMenu1.setText("Single Animation  ");
welcomejMenu1.setToolTipText("Single Animation");
welcomejMenu1.setFont(new java.awt.Font("Dialog", 1, 18));
public void mouseClicked(java.awt.event.MouseEvent evt) {
    welcomejMenu1MouseClicked(evt);
}
welcomejMenuBar1.add(welcomejMenu1);
welcomejMenu1.getAccessibleContext().setAccessibleName("Menu Bar");
welcomejMenu7.setText("Algorithm Race  ");
welcomejMenu7.setToolTipText("Algorithm Race");
welcomejMenu7.setFont(new java.awt.Font("Dialog", 1, 18));
public void mouseClicked(java.awt.event.MouseEvent evt) {
    welcomejMenu7MouseClicked(evt);
}
welcomejMenuBar1.add(welcomejMenu7);
welcomejMenu2.setText("Help  ");
welcomejMenu2.setToolTipText("Help");
welcomejMenu2.setFont(new java.awt.Font("Dialog", 1, 18));
public void mouseClicked(java.awt.event.MouseEvent evt) {
    welcomejMenu2MouseClicked(evt);
}
welcomejMenuBar1.add(welcomejMenu2);
welcomejMenu3.setText("Algorithm Analysis  ");
welcomejMenu3.setToolTipText("Algorithm Analysis");
welcomejMenu3.setFont(new java.awt.Font("Dialog", 1, 18));
public void mouseClicked(java.awt.event.MouseEvent evt) {
    welcomejMenu3MouseClicked(evt);
}
welcomejMenuBar1.add(welcomejMenu3);
welcomejMenu5.setText("About  ");
welcomejMenu5.setToolTipText("About");
welcomejMenu5.setFont(new java.awt.Font("Dialog", 1, 18));

welcomejMenu5.addMouseListerner(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {  
        welcomejMenu5MouseClickeed(evt);  
    }  
};  
welcomejMenuBar1.add(welcomejMenu5);  
welcomejMenu6.setText("Quit ");  
welcomejMenu6.setToolTipText("Quit");  
welcomejMenu6.setFont(new java.awt.Font("Dialog", 1, 18));  
welcomejMenu6.addMouseListerner(new java.awt.event.MouseAdapter() {  
    public void mouseClicked(java.awt.event.MouseEvent evt) {  
        welcomejMenu6MouseClickeed(evt);  
    }  
});  
welcomejMenuBar1.add(welcomejMenu6);  
setJMenuBar(welcomejMenuBar1);  
Toolkit toolkit = Toolkit.getDefaultToolkit();  
Dimension scrnsize = toolkit.getScreenSize();  
setSize(scrnsize);

D.14 ‘Run’ button method in the single animation screen

private void twojButton2MouseClicked(java.awt.event.MouseEvent evt) {  
    if (twojButton2.isEnabled() == true) {  
        // here we need to call the relevant method to sort the data in the desired way  
        if (anotheri == 0) {  
            // call bubblesort method  
            BubSort();  
        }  
        else if (anotheri == 1) {  
            // call quicksort method  
            QuickSort();  
        }  
        else if (anotheri == 2) {  
            // call insertion sort method  
            InsertionSort();  
        }  
        else if (anotheri == 3) {  
            // call selection sort method  
            SelectionSort();  
        }  
        else if (anotheri == 4) {  
            // call heap sort method  
            HeapSort();  
        }  
        twojButton2.setEnabled(false);  
    }

D.15 ‘Run’ button method in the algorithm race screen

private void mawjButton3MouseClicked(java.awt.event.MouseEvent evt) {  
    if (mawjButton3.isEnabled() == true) {  
        // start button
```c
int flag1 = 1;
int flag2 = 2;
int flag3 = 3;
int flag4 = 4;
int flag5 = 5;

if (multicounter == 2) {
    if (c1 == 0) {
        // call bub sort
        MultiBubSort(multielements1, flag1);
    } else if (c1 == 1) {
        // quick sort
        MultiQuickSort(multielements1, flag1);
    } else if (c1 == 2) {
        // insertion
        MultiInsertionSort(multielements1, flag1);
    } else if (c1 == 3) {
        // selection
        MultiSelectionSort(multielements1, flag1);
    } else if (c1 == 4) {
        // heap
        MultiHeapSort(multielements1, flag1);
    } if (c2 == 0) {
        // call bub sort
        MultiBubSort(multielements2, flag2);
    } else if (c2 == 1) {
        // quick sort
        MultiQuickSort(multielements2, flag2);
    } else if (c2 == 2) {
        // insertion
        MultiInsertionSort(multielements2, flag2);
    } else if (c2 == 3) {
        // selection
        MultiSelectionSort(multielements2, flag2);
    } else if (c2 == 4) {
        // heap
        MultiHeapSort(multielements2, flag2);
    } else if (multicounter == 3) {
        if (c1 == 0) {
            // call bub sort
            MultiBubSort(multielements1, flag1);
        } else if (c1 == 1) {
            // quick sort
            MultiQuickSort(multielements1, flag1);
        } else if (c1 == 2) {
            // insertion
```
MultiInsertionSort(multielements1, flag1);
}
else if (c1 == 3) {
    // selection
    MultiSelectionSort(multielements1, flag1);
}
else if (c1 == 4) {
    // heap
    MultiHeapSort(multielements1, flag1);
}
if (c2 == 0) {
    // call bub sort
    MultiBubSort(multielements2, flag2);
}
else if (c2 == 1) {
    // quick sort
    MultiQuickSort(multielements2, flag2);
}
else if (c2 == 2) {
    // insertion
    MultiInsertionSort(multielements2, flag2);
}
else if (c2 == 3) {
    // selection
    MultiSelectionSort(multielements2, flag2);
}
else if (c2 == 4) {
    // heap
    MultiHeapSort(multielements2, flag2);
}
if (c3 == 0) {
}
else if (c3 == 1) {
    // call bub sort
    MultiBubSort(multielements3, flag3);
}
else if (c3 == 2) {
    // quick sort
    MultiQuickSort(multielements3, flag3);
}
else if (c3 == 3) {
    // insertion
    MultiInsertionSort(multielements3, flag3);
}
else if (c3 == 4) {
    // selection
    MultiSelectionSort(multielements3, flag3);
}
else if (c3 == 5) {
    // heap
    MultiHeapSort(multielements3, flag3);
}
else if (multicounter == 4) {
    if (c1 == 0) {
        // call bub sort
        MultiBubSort(multielements1, flag1);
    }
    else if (c1 == 1) {

// quick sort
        MultiQuickSort(multielements1, flag1);
    } else if (c1 == 2) {
        // insertion
        MultiInsertionSort(multielements1, flag1);
    } else if (c1 == 3) {
        // selection
        MultiSelectionSort(multielements1, flag1);
    } else if (c1 == 4) {
        // heap
        MultiHeapSort(multielements1, flag1);
    }
    if (c2 == 0) {
        // call bub sort
        MultiBubSort(multielements2, flag2);
    } else if (c2 == 1) {
        // quick sort
        MultiQuickSort(multielements2, flag2);
    } else if (c2 == 2) {
        // insertion
        MultiInsertionSort(multielements2, flag2);
    } else if (c2 == 3) {
        // selection
        MultiSelectionSort(multielements2, flag2);
    } else if (c2 == 4) {
        // heap
        MultiHeapSort(multielements2, flag2);
    }
    if (c3 == 0) {
    } else if (c3 == 1) {
        // call bub sort
        MultiBubSort(multielements3, flag3);
    } else if (c3 == 2) {
        // quick sort
        MultiQuickSort(multielements3, flag3);
    } else if (c3 == 3) {
        // insertion
        MultiInsertionSort(multielements3, flag3);
    } else if (c3 == 4) {
        // selection
        MultiSelectionSort(multielements3, flag3);
    } else if (c3 == 5) {
        // heap
        MultiHeapSort(multielements3, flag3);
    }
    if (c4 == 0) {
    }
else if (c4 == 1) {
    // call bub sort
    MultiBubSort(multielements4, flag4);
}
else if (c4 == 2) {
    // quick sort
    MultiQuickSort(multielements4, flag4);
}
else if (c4 == 3) {
    // insertion
    MultiInsertionSort(multielements4, flag4);
}
else if (c4 == 4) {
    // selection
    MultiSelectionSort(multielements4, flag4);
}
else if (c4 == 5) {
    // heap
    MultiHeapSort(multielements4, flag4);
}
else if (multicounter == 5) {
    if (c1 == 0) {
        // call bub sort
        MultiBubSort(multielements1, flag1);
    }
    else if (c1 == 1) {
        // quick sort
        MultiQuickSort(multielements1, flag1);
    }
    else if (c1 == 2) {
        // insertion
        MultiInsertionSort(multielements1, flag1);
    }
    else if (c1 == 3) {
        // selection
        MultiSelectionSort(multielements1, flag1);
    }
    else if (c1 == 4) {
        // heap
        MultiHeapSort(multielements1, flag1);
    }
    if (c2 == 0) {
        // call bub sort
        MultiBubSort(multielements2, flag2);
    }
    else if (c2 == 1) {
        // quick sort
        MultiQuickSort(multielements2, flag2);
    }
    else if (c2 == 2) {
        // insertion
        MultiInsertionSort(multielements2, flag2);
    }
    else if (c2 == 3) {
        // selection
        MultiSelectionSort(multielements2, flag2);
    }
    else if (c2 == 4) {
MultiHeapSort(multielements2, flag2);  
}
else if (c3 == 1) {
    // call bub sort
    MultiBubSort(multielements3, flag3);
}
else if (c3 == 2) {
    // quick sort
    MultiQuickSort(multielements3, flag3);
}
else if (c3 == 3) {
    // insertion
    MultiInsertionSort(multielements3, flag3);
}
else if (c3 == 4) {
    // selection
    MultiSelectionSort(multielements3, flag3);
}
else if (c3 == 5) {
    // heap
    MultiHeapSort(multielements3, flag3);
}
else if (c4 == 1) {
    // call bub sort
    MultiBubSort(multielements4, flag4);
}
else if (c4 == 2) {
    // quick sort
    MultiQuickSort(multielements4, flag4);
}
else if (c4 == 3) {
    // insertion
    MultiInsertionSort(multielements4, flag4);
}
else if (c4 == 4) {
    // selection
    MultiSelectionSort(multielements4, flag4);
}
else if (c4 == 5) {
    // heap
    MultiHeapSort(multielements4, flag4);
}
else if (c5 == 1) {
    // call bub sort
    MultiBubSort(multielements5, flag5);
}
else if (c5 == 2) {
    // quick sort
    MultiQuickSort(multielements5, flag5);
}
else if (c5 == 3) {
    // insertion
MultiInsertionSort(multielements5, flag5);
}
else if (c5 == 4) {
    // selection
    MultiSelectionSort(multielements5, flag5);
}
else if (c5 == 5) {
    // heap
    MultiHeapSort(multielements5, flag5);
}
}
mawjButton3.setEnabled(false);
}

D.16 Setting the contents of the help text area method

private void helpjButton1MouseClicked(java.awt.event.MouseEvent evt) {
    int tempi = helpjComboBox1.getSelectedIndex();
    if (tempi == 0) {
        helpjTextArea1.setText(blurb);
    }
    if (tempi == 1) {
        helpjTextArea1.setText(faqs);
    }
    if (tempi == 2) {
        helpjTextArea1.setText(stepThru);
    }
    if (tempi == 3) {
        helpjTextArea1.setText(wt1);
    }
    if (tempi == 4) {
        helpjTextArea1.setText(wt2);
    }
    if (tempi == 5) {
        helpjTextArea1.setText(wt3);
    }
    if (tempi == 6) {
        helpjTextArea1.setText(links);
    }
}
Bibliography

Books


Web sites


http://www.ida.liu.se/labs/aslab/groups/um/hci/

http://www.sum-it.nl/enguilin.html

http://research.compaq.com/src/jcat


[13] Sorting Homepage
http://mathworld.wolfram.com/sorting.html

[15] Sort Benchmark Home Page

[16] Terabyte Sort
[17] NOW-Sort Homepage (25/11/03)
http://now.cs.berkelely.edu/nowsort

[21] Algorithm Visualisation Using Animation

[25] Zeus and Algorithm Animation at SRC

[26] XTango Algorithm Animation System

[27] Polka Animation System

http://www.spelman.edu/ssmj/vol1_1/tech/clary.htm

[31] Using Animation of State Space Algorithms to Overcome Student Learning
Difficulties

[33] GUIS: Past, Present, Future

[37] User Interaction Guidelines and Standards
http://www.acm.org/~perlman/interactions/21-std.html

[38] Algoritms and Animations
http://www.cc.gatech.edu/people/alumni/PhD/Colleen.Kehoe/papers/lit_review/lit_review.html

[40] Computer Science Education Links
http://www.cs.cofc.edu/~mccauley/edlinks/#

[41] Kevin White’s University Home Page
http://www.bath.ac.uk/~ma0kjw/project

**Journals and papers**


[18] Arpaci-Dusseau et al. (1988), Searching for the Sorting Record: Experiences in Tuning NOW-Sort. SPDT.


Videos