Researching and Implementing on AKS Algorithms

Tong Jin

BSc Honors in Computer Science

University of Bath

May 2005

Researching and Implementing on AKS Algorithms

Submitted by: Tong Jin

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 institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed.....

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

Signed.....

Abstract

The dissertation deals with the implementation of AKS class primality testing. The AKS Algorithm is the first deterministic polynomial time primality test named after its authors M. Agarwal, N. Kayal and N. Saxena. A primality testing algorithm is the algorithm that checks whether a given number is prime number or not. “Deterministic” means that operations carried out by the algorithm are determined entirely by the algorithm and the input. In particular the algorithm does not make any random choices. “Polynomial time” means that there is some polynomial *p* such that, for every input *n*, the algorithm takes at most *p (n)* steps. This algorithm is presented in the paper “PRIMES is in P” [1] and since then many people had focused on this and put many efforts on its implementation. We have implemented the AKS algorithm and have measured it running results, besides we also discussed other three algorithms such as AKS Version 3, AKS Conjecture, and AKS Algorithm improved by Daniel Bernstein. We had implemented them and compared with AKS Algorithm. After given some running time data, we analysed the performance of these algorithms. Some of them would support the theoretic analysis in those papers. The result show that the algorithms based on the Conjecture is the fastest and the AKS algorithm improved by D. Bernstein outperforms then the AKS Version 3 which revised by AKS author themselves. We also have found out from running results that the average complexity of original AKS Algorithm seems to be $\tilde{O}\left(\log^{7.5} n\right)$.

Acknowledgements

With many thanks to my project supervisor, Prof. James H Davenport, for his many helpful suggestions throughout the project. I am also need to thank Kaye Collies for her proofread my document.

Contents

Chapter 1

Introduction

Nowadays, testing a number whether is prime number or not is one of the fundamental problems in computational number theory. It since ancient times, every mathematician has been fascinated by problems concerning prime numbers. Primality testing has wide applications in computer science, especially in cryptography.

Many researchers have been interested in the primality-testing algorithm from a theoretical standpoint. Search for a polynomial time deterministic algorithm for primality testing has been an open problem for a long time. The importance of the algorithm needs little mention. Consequently, many researchers have provided various algorithms. There are some very efficient algorithms such as the Solvay - Strassen or Miller Rabin algorithm. They are termed as "probabilistic" polynomial time primality testing algorithms. The term probabilistic implies that they can make mistakes some times. In 1983, Adleman, Pomerance, and Rumely achieved a major breakthrough by giving a deterministic algorithm that runs in $(\log n)^{O(\log\log\log n)}$ time, though correctness proofs of this deterministic algorithm is very complex.

After tremendous efforts invested by researchers in hundreds of years, and despite such efforts of many researchers, no one could find the algorithms in the class **P**. It was finally proved by M. Agrawal, N. Kayal and N. Saxena that the set of primes is in the complexity class **P**. The name of this algorithm is called AKS Algorithm. They named after these three inventors. For any given integer n, the AKS algorithm can determine the number n is prime number or not in the running time of $O(\log^{12} n)$, while the best deterministic algorithm known before has polynomial time complexity. The AKS Algorithm is based on the derandomization of a polynomial identity testing. It includes

many iterations of polynomial modular exponentiation. Since the algorithm has been presented many efforts are made to implement the algorithm. The major challenge is to implement the congruence, which is the most time consuming step in the algorithm. Most of the efforts are focussed on speeding up this congruence check.

In this project we are concentrating on implementing of the AKS algorithm and its run time measurements and to discover any improvement during the development process. Besides we will also look into some related algorithms, such as AKS Version 3, AKS Conjecture, and AKS Algorithm improved by D. Bernstein. The reasons we are looking other three algorithms are. Firstly, they are all did some improvements after the AKS original has published except AKA Conjecture which has not yet been proved its correctness. Secondly, they are all belonging to the AKS-class algorithm, they have very similar features around these algorithm. In the project we use Microsoft Visual C++ 6 programming language together with NTL library to develop the AKS algorithms. NTL is portable C++ number theory library; it supports basic computational arithmetic, polynomial computation, congruence, polynomial ring, and so on, which we use these NTL's properties to develop the algorithms.

This dissertation has divided into several different parts. In the chapter 2, we are more focus on the history of primality testing, basic knowledge of number theory for understanding the problem, and give brief introduction to AKS Algorithm. In chapter 3 we will discuss the four algorithms which we will mostly concentrated in this report, those algorithms are AKS Algorithm (original), AKS Version 3, AKS Conjecture, and AKS Algorithm improved by D. Bernstein. We will look its algorithm in steps and its complexity analysis. Afterwards, we will concentrate on requirements specification in chapter 4, which includes functional, user interface and NTL requirements. The requirements have been listed and ready to be constructed in the design process, and the design process can be seen in the chapter 5. In chapter 6, we are focus on the implementation of these four algorithms; we will discuss some basic algorithms which needed in order to implement these four algorithms mentioned in chapter 3. After we implemented the system, then we need to test it and all the testing process will discuss in

the chapter 7. The testing includes both system testing and results measurement of the algorithm. According to the testing result and their measurements we will discuss them and summarize our work in the chapter 8, and then we will look at any further works which should be carried out in the future and extra features that can be implemented in chapter 9.

Chapter 2

Literature Review

2.1 Introduction

"The problem of distinguishing prime numbers from composite numbers and of resolving the latter into their prime factors is known to be one of the most important and useful in arithmetic. It has engaged the industry and wisdom of ancient and modern geometers to such an extent that it would be superfluous to discuss the problem at length… Further, the dignity of the science itself seems to require that every possible means be explored for the solution of a problem so elegant and so celebrated."

- Karl Friedrich Gauss, Disquisitiones Arithmeticae, 1801 (translation from [Knu98])

Prime numbers are central to mathematics. They are the building blocks of the natural numbers - any integer can be expressed uniquely as a product of primes. Finding out whether a given number *n* is a prime or not is a problem that was formulated in ancient times, and has caught the interest of mathematicians again and again for centuries. Polynomial and polynomial arithmetic also play an important role in many areas of mathematics. Tow such areas are cryptography and primality testing. On August 2002, a paper with the title "PEIMES is in P" [1], by M. Agrawal, N. Kayal, and N. Saxena, appeared on the website of Indian Institute of Technology at Kanpur, India. In this paper it was shown that the *"primality problem"* has *"deterministic algorithm"* that runs in *"polynomial time"*.

This literature review will look into various aspects based on the AKS Algorithm, such as its history, its recent improvements, background knowledge of number theory, and look into itself in more detail such as basic idea and complexity analysis. We also will look at NTL, which is a portable C++ library for dealing with number theories, and it will be required in the further AKS development.

2.2 Understanding the AKS algorithm

2.2.1 What is AKS algorithm?

The AKS Algorithm was invented by three men, named M. Agrawal, N. Kayal and N. Saxena in August 2002. The idea is that the prime (or composite) character of an input integer is determined without doubt, in *polynomial time*. This in turn means that the time to prove or reject primality is $O(\log^{\mu} n)$ for some constant power μ; the run time is bounded by a polynomial function of the number of bits in *n*. This algorithm was a major breakthrough, because although there were some algorithms existed and could be used to test numbers for primality testing, they took longer than *polynomial time* to run. For example: Eratosthenes' Sieve, Fermat's Little Theorem, and so on.

2.2.2 History of primality testing

In 17th Century, Fermat's Little Theorem says for any prime integer *n* and any integer *a* coprime to *n*, $a^{n-1} = 1 \pmod{n}$. Although the converse of this theorem does not hold, this result also has been the starting point for several efficient primality testing algorithms. 1976, Miller used this property to obtain a deterministic polynomial time algorithm for primality testing assuming *Extended Riemann Hypothesis* (ERH). In the 1980, Rabin modified Miller's test, yield an unconditional but randomized polynomial time algorithm.

In 1983, Adleman, Pomerance, and Rumely achieved a major breakthrough by giving a deterministic algorithm for primality testing that runs in $(\log n)^{O(\log \log \log n)}$ time (all the previous deterministic algorithm required exponential time). In 1986, Goldwasser and Kilian proposed a randomized algorithm based on Elliptic curves running in expected polynomial time on almost all inputs (all inputs under a widely believed hypothesis) that produces a certificated for primality (until then, all randomized algorithms produced certificates for compositeness only). In 1992, Adleman and Huang modified Goldwasser and Kilian algorithm to obtain a randomized polynomial time algorithm that always produced a certificate for primality. Finally, in August 2002, M. Agrawal, N. Kayal and N. Saxena have proved the possibility of testing a prime in polynomial time, along with an algorithm. The AKS Algorithm can determine the input number n is prime number or not in the runs time $O(\log^{12} n)$, while the best deterministic algorithm known before has subexponential complexity.

2.2.3 What is prime number?

One of the most important and beautiful fields of mathematics is number theory - the study of numbers and their properties. Despite the fact that mathematicians have been studying numbers for as long as humans have been able to count, the field of number theory is far from being outdated; some of the most exciting and important problems in mathematics today have to do with the study of numbers. In particular, prime numbers are of great interest. Prime numbers are special because they are the elementary building blocks of the multiplicative structure on the integers; every integer can be written in only one way as a product of its prime factors. The mathematically precise version of this assertion is known as The Fundamental Theorem of Arithmetic.

The definition of prime numbers is: A number p is prime if it is a positive integer greater than 1 ($p > 1$) and is divisible by no other positive integers other than 1 and itself. Positive integers greater than 1 that are not prime are called composite integers.

2.3 What is NTL & What is for?

NTL is a high-performance, portable C++ library providing data structure and algorithm for arbitrary length integer numbers; for vectors, matrices, and polynomials over integer numbers and over finite fields; and for arbitrary precision floating point arithmetic.

NTL provides high quality implementations of algorithms for:

- Arbitrary length integer arithmetic and arbitrary precision floating point arithmetic;
- Polynomial arithmetic over the integers and finite fields including basic arithmetic, polynomial factorization, irreducibility testing, computation of minimal polynomials, traces, norms, and more;
- Basic linear algebra over the integer numbers, and finite fields,

NTL is written entirely in C++, it can be easily installed in a matter of minutes on just about any platform, including virtually any 32 or 64-bit machine running any flavor of Unix, as well as PCs running Windows 95, 98, or NT or above, and Macintoshes. NTL achieves this portability by avoiding esoteric C++ features, and by avoiding assembly code; it should therefore remain usable for years to come with little or no maintenance, even as processors and operating systems continue to change and evolve. NTL provides a clean and consistent interface to a large variety of classes representing mathematical objects. It provides a good environment for easily and quickly implementing new number-theoretic algorithms, *without sacrificing performance*. NTL is free software that is intended for research and educational purposes only.

2.4 The Number theory

To understand the AKS algorithm, first we need to understand some aspect from the number theory. Number theory is one of the oldest branches of pure mathematics, and

also one of the largest. It concerns questions about numbers, usually meaning whole numbers or rational numbers (fractions).

The basic and most important part in the number theory is call elementary number theory. It involves divisibility among integers -- the division "algorithm", such as the Euclidean algorithm and thus the existence of greatest common divisors, elementary properties of primes (the unique factorization theorem, the infinitude of primes), congruence (and the structure of the sets $\mathbf{Z}/n\mathbf{Z}$ as commutative rings), including Fermat's little theorem and Euler's theorem extending it.

Number theorem asserts the approximate density of primes among the integers, which has difficult but "elementary" proofs. The number theory has been divided into many different parts according to it uses in certain aspects. For instead, "Computational number theory" involves the effectiveness of algorithms for computation of number-theoretic quantities, "Algebraic Number Theory" extends the concept of "number" to mean an element of some ring, usually the ring of integers in a finite algebraic extension of the rational number field, and so on. We will look into a few number theory algorithms in the coming sections, and they are maybe involved in the later project development.

2.4.1 Euclidean GCD algorithms

The mathematical definition is: The **greatest common divisor** (GCD) of two integers *m* and *n* is the greatest integer that divides both *m* and *n* with no remainder.

The main algorithm for the calculation of the GCD of two integers is the binary Euclidean algorithm. It is based on the following identities: gcd(m, n) = gcd(n, m), gcd(m, n) = gcd(m-n, n), and for odd *n*, gcd(2*m, n) = gcd(m, n). Thus we can produce a sequence of pairs with the same GCD as the original two numbers, and each pair will be at most half the size of the previous pair. The number of steps is logarithmic in the number of digits in *m*, *n*. The only operations needed for this algorithm are binary shifts and subtractions (no modular division is necessary).

To speed up the calculation when one of the numbers is much larger than another, one could use the property gcd(m, n)=gcd(m, mod(m, n)). This will introduce an additional modular division into the algorithm; this is a slow operation when the numbers are large.

2.4.2 Congruence

If two numbers *a* and *b* have the property that their difference *a-b* is integrally divisible by a number *m* (i.e., *(a-b)/m* is an integer), then *a* and *b* are said to be "congruent modulo *m*." The number *m* is called the modulus, and the statement "*a* is congruent to *b* (modulo *m*)" is written mathematically as

$$a \equiv b \pmod{m}$$

If *a-b* is *not* integrally divisible by *m*, then we say "*a* is *not* congruent to *b* (modulo *m*)," which is written

$$a \not\equiv b \pmod{m}$$

Congruence satisfies a number of important properties, and is extremely useful in many areas of number theory. Using congruence, simple divisibility tests to check whether a given number is divisible by another number can sometimes be derived. For example, if the sum of a number's digits is divisible by 3, then the original number is divisible by 3.

Congruence also has their limitations. For example, if $a \equiv b$ and $c \equiv d \pmod{n}$, then it follows that $a^x \equiv b^x$, but usually not that $x^c \equiv x^d$ or $a^c \equiv b^d$. In addition, by "rolling over," congruence discards absolute information.

2.4.3 Factorization

If a number *n* is not prime, then we often seek to factor it into primes. The Fundamental Theorem of Arithmetic says: every positive integer greater than one can be expressed

uniquely as a product of primes, apart from the rearrangement of terms. The canonical (or standard) form of the factorization is to write $n = \prod_{i=1}^{k} p_i^{e_i}$ where the primes p_i satisfy $p_1 < p_2 < ... < p_k$ and the exponents are positive integers. For example, $60 = 2^{2} \cdot 3^{1} \cdot 5^{1}$. We can reword the Fundamental Theorem this way: the canonical factorization of an integer greater than one is unique.

2.5 The Primality Testing Algorithms

2.5.1 Probabilistic Algorithms

A probabilistic algorithm is an algorithm where the result and/or the way the result is obtained depend on chance, but has a theoretical chance of being wrong. These algorithms are also sometimes called randomized algorithms.

One incentive for using probabilistic algorithms is that their application does not normally require sophisticated mathematical knowledge. Further, the programming is often rather trivial which means that an acceptable approximation can be obtained quickly. One can say that the use of probabilistic algorithms sometimes allow that theoretical knowledge and analytical work is compensated for by making extensive simple machine computations. In some other cases the probabilistic algorithms are the simplest and even the most efficient available and for some problems no other feasible algorithm is known to exist such as primality testing.

In the primality testing aspect, there are some people use probabilistic algorithms to test whether a number is prime or not, such as Lehamann-Peralta, Solovay-Strassen, Miller-Rabin.

2.5.2 Deterministic Algorithms

An algorithm whose behaviour can be completely predicted from the input and it is a finite set of well-defined instructions for accomplishing some task which, given an initial state, will result in a corresponding recognizable end-state.

Different algorithms may complete the same task with a different set of instructions in more or less time, space, or effort than others. In the deterministic algorithm, each time a certain set of input is presented, the algorithm gives the same results as any other time the set of input is presented. For algorithms with state or that maintain information between inputs, "*the input*" means everything since the algorithm was stared from an initial state.

An algorithm is polynomial time if the number of 'simple' steps (such as additions, multiplications, comparisons, and so on) required is bounded by polynomial in the size of the inputs. The example of deterministic algorithm, such as recently developed algorithm: The AKS Algorithm. The algorithm always returns right answer and it works in the polynomial time and it also knows as the first algorithm ever in the history to determine an input integer whether a prime or not.

2.6 The AKS Algorithm

2.6.1 Basic idea of approach

The primality testing is based on the following identity [1] for prime numbers. This same identity was basis for a randomized polynomial time algorithm.

Identity: Suppose that a is co-prime to p, then p is prime if and only if

$$(x\text{-}a)^p \equiv (x^p - a)(\bmod p) \qquad (1)$$

which *a*, *p* relatively prime, we consider when *0< i<p*, the coefficient of x^i. If *p* is prime numbers, then all the coefficients are zero. If *p* is composite: consider a prime *q* that is a factor of *p*, then if the coefficient of x^q is not zero (mod *p*), then *p* is not a prime.

For example:

$$p := 5 \qquad a := 4 \qquad \gcd\left(\begin{vmatrix} p \\ a \end{vmatrix}\right) = 1$$

$$(x-a)^p \text{ expand } \rightarrow x^5 - 20x^4 + 160x^2 + 1280x - 1024$$

$$(x^p - a) \rightarrow x^5 - 4$$

Thus given *a p* as input, one could pick a polynomail P(x) = *x-a*, and compute whether the congruence (1) is satisfied or not. However, this takes time $\Omega(p)$ because we need to evaluate *p* coefficients in the LHS in the worst case. Therefore, we need to evaluate both side of (1) modulo a polynomial of the form x^r - *1*. On iteration of the algorithm will consist of evaluating whether the following holds:

$$(x-a)^p \equiv (x^p - a)(\bmod x^r - 1, p) \qquad (2)$$

From the identity it is immediate that all primes p satisfy the above congruence for all values of *a* and *r*; however, some composites p may also satisfy (2) for a few values of (a, r). The above congruence takes $O(r^2 \log^3 p)$ time for verification, or even better if use Fast Fourier Multiplication then we have $O(r \log^2 p)$.

2.6.2 Implementing of the AKS algorithm

The AKS Algorithm in the paper “PIMES is in P”[1] is having 12 steps pseudo code stated. But we can compress into three main steps, and each of these three steps is using various mathematical algorithms to develop. In the later project development process we

will discuss the implementation of the algorithm in more details. Below are the three main steps required in the AKS Algorithm:

a) *Perfect power checking the input n*

b) *Find r, q=LargestPrimeFactor(r) that* $q \geq 4\sqrt{r}\log n \;\&\; n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$

c) *a=1 to* $2\sqrt{r}\log n$ *check* $(x-a)^n \not\equiv x^n - a \pmod{x^r - 1, n}$

2.6.3 Complexity analysis of AKS algorithm

Below we will introduce the time complexity analysis of the AKS Algorithm. For the better understanding the complexity in each step of the algorithm we will produce a graph basic complexity analysis. The algorithm is obtained from the paper "PRIMES is in P" [1]. The first step of the algorithm takes asymptotic time $O(\log^3 n)$. And the *while loop* in step b makes $O(\log^6 n)$ iterations. The *for loop* at last step takes asymptotic time $O\left(r^{3/2}\log^3 n\right) = O(\log^{12} n)$. Therefore, the total complexity of AKS Algorithm is $O(\log^{12} n)$.

2.7 Recent improvements

In recent development there have appeared new analyses and variants, notably those of (Lenstra 2002) [9], (Pomerance 2002) [12], (Berrizbeitia 2003) [6], (Cheng 2003) [7], (Bernstein 2003a, b) [4,5], (Lenstra and Pomerance 2003) [10]. (Berrizbeitia 2003), (Cheng 2003), (Bernstein 2003a, b), (Lenstra and Pomerance 2003) general trend is the lowering of the complexity exponent *k*, which began as an unconditional $k = 12 + \varepsilon$ with the original AKS paper and now stands at $k = 4 + \varepsilon$ for certain input integers, or a similar low exponent for general input integers but with a possible, minuscule chance of

the answer "maybe". The precise complexity exponent depends, in these various treatments, on whether one is proceeding unconditionally, or heuristically and so on. Incidentally, the current best bound for unconditional complexity with effectively computable big-O constant is the $O\left(\log^{6+\varepsilon} p\right)$ result of (Lenstra and Pomerance 2003) [10]; we stress that in some treatments there are either conditions or an ineffective constant (e.g. the original AKS result did not involve an effective constant). On the practical side, the speed improvements over the original AKS breakthrough are already in just a matter of month's orders of magnitude such as D. Bernstein 2003 a [4].

2.8 Use of prime numbers

The uses of primes are manifold. They were first studied because many of the properties of numbers are directly tied to their factorizations. Beside their austere intrinsic beauty, prime numbers are now the key to the Internet revolution because they are used for a variety of encryption methods used to keep transactions safe. NASA scientists even decided that they are a good sign of intelligence and have included a short list of primes on the plaques sent out with the voyager spacecraft.

Prime numbers have attracted much attention from mathematicians for many centuries. Questions such as "How many are there?", "Is there a formula for generation them?" and so on. However, the first actual use of prime numbers in an important area outside of the theory of numbers was discovered only in the mid to late 1900s. This was in the establishment of a technical system to be used in maintaining the secrecy of electronic communications. The system graph shows below:

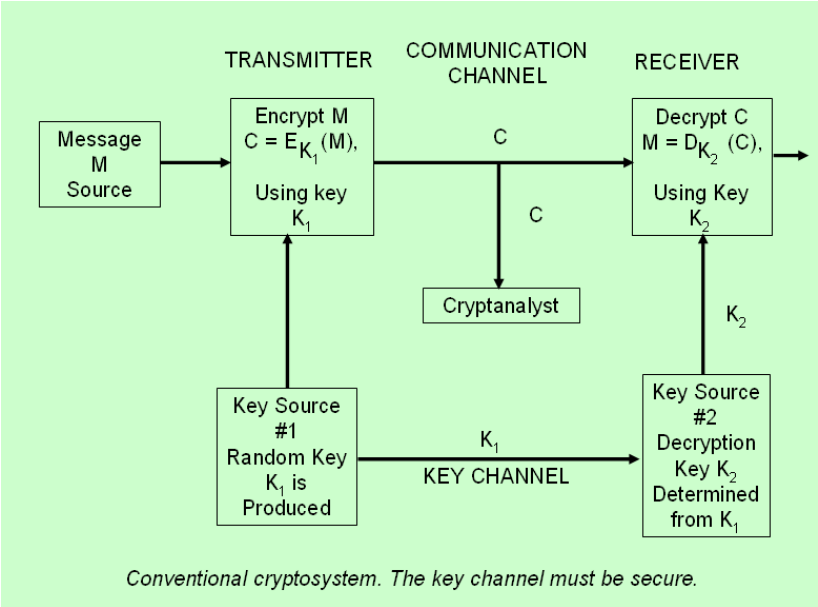

Figure 1: Conventional Cryptosystem

The Diffie-Hellman scheme proposed in 1976, was a radical departure from what up to then, had all been essentially 'private key' schemes. The idea was that everyone would own both a 'private key' and a 'public key'. The public key would be published in a directory, like a telephone book. If X wanted to send Y an encrypted message, X simply looked up Y's public key, applied it and sent the message. Only Y knew Y's private key and could use it to decrypt the message. Then along came the Riveat, Shamir, Adleman (RSA) solution in 1977. There is no way of breaking RSA is known, other than finding the secret information. So the primes p and q in RSA must be of sufficient size that factorization of their product is beyond computational reach. Moreover, they should be random primes in the sense that they be chosen as a function of a random input which defines a pool of candidates of sufficient cardinality that an exhaustive attack is infeasible. In practice, the resulting primes must also be of a pre-determined bit length, to meet system specifications.

2.9 Summary

This literature review has discussed in detail about the AKS Algorithm and its related areas. In the first section, we mainly talked about the understanding the concept of AKS algorithm, such as its history, because if we need to understand some material we first need look its background and relevant aspects. We explanted in more detailed about the AKS algorithm, as the previous works of the AKS algorithm which means its history, we brief talked from the 17^{th} century's Fermat's Little theorem until the recent development, and then we discuss the number theory and prime numbers which both important aspect for understanding the AKS algorithm. We also talk about the NTL, which is a portable C++ library for doing number theory (such as the AKS algorithm) and it is required in the later project development.

We discussed the primality testing algorithms, which separate into to parts, one is called probabilistic algorithm and other one called deterministic algorithm. In the probabilistic algorithm we discussed what is probabilistic algorithm and its history. We also had briefly talked about deterministic algorithm. Afterwards, we stared with the idea of approach, which with the idea of Suppose that a is co-prime to p, then p is prime if and only if $(x\text{-}a)^p \equiv (x^p - a)(\bmod p)$ which a, p are relative prime. We then looked into the implementation of the AKS algorithm; we listed the algorithm of its main steps, then we looked into the complexity analysis of the AKS Algorithm, give out the total complexity of the AKS Algorithm. We have discussed recent developments based on the AKS algorithm. Since the AKS algorithm paper has published in August 2002, there are groups of people were interested in this aspect, such as D. Bernstein, Lenstra, Cheng Q and so on. The aims of those people are trying to speed up the AKS Algorithm.

Primality testing has wide applications in computer science, especially in cryptography. So in the last section, we discussed the uses of prime numbers. The first use of prime numbers in an important area outside of the theory of numbers was discovered only in the mid to late 1900s, and the RSA is the most powerful Cryptosystem so far.

Chapter 3

AKS algorithms & Complexities

3.1 Introduction

On of our project aim is to implement the AKS Algorithm, so in this chapter we will consider its algorithm and its time complexity analysis. Besides, we will also consider other 3 related algorithms, and they are named AKS Version 3, AKS Conjecture, and AKS improved by D. Bernstein. The AKS Version 3 and AKS improved by D. Bernstein have been proved their correctness and can been used in the practices. But the AKS Conjecture was published at some time when the AKS Algorithm has first introduced in August 2002 'PRIMES is in P' paper [1]. It is only conjecture and has not yet been proved. At this chapter we are mainly discuss the algorithms and its complexities. The correctness of these four algorithms will not be included this dissertation.

3.2 AKS Algorithm

3.2.1 AKS Algorithm Original

This algorithm was first published in August 2002 by three India computer scientists named M. Agrawal, N. Kayal, and N. Saxena. This was first time in the history that to

determine a number whether or not a prime in polynomial time complexity. We have already looked into its basic ideas of the AKS Algorithm in the chapter 2. The AKS Algorithm can be written in three main steps and they are listed below:

d) Perfect power checking the input n

e) Find r, q=LargestPrimeFactor(r) that $q \geq 4\sqrt{r}\log n \;\&\; n^{\frac{r-1}{q}} \not\equiv 1 \pmod{r}$

f) a=1 to $2\sqrt{r}\log n$ *check* $(x-a)^n \not\equiv x^n - a \pmod{x^r - 1, n}$

The most time consuming step is at the last step of the *for loop* to compute the congruence modulo. We will discus its time complexity in the next section just below the algorithm. The algorithm is obtained from the 'PRIMES is in P' paper [1].

Input: integer $n > 1$

1. *if (n is of the form* $a^b, b > 1$*) output COMPOSITE;*
2. $r = 2$;
3. *while* $(r < n)$*{*
4. *if (* $\gcd(n, r \neq 1)$ *) output COMPOSITE;*
5. *if (r is prime)*
6. *let q be the largest prime factor of* $r - 1$*;*
7. *if (* $q \geq 4\sqrt{r}\log n$ *) and (* $n^{\frac{r-1}{q}} \neq 1 \pmod{r}$*)*
8. *break;*
9. $r \leftarrow r + 1$*;*
10. *}*
11. *for* $a = 1$ *to* $2\sqrt{r}\log n$
12. *if (* $(x-a)^n \not\equiv (x^n - a) \pmod{x^r - 1, n}$ *) output COMPOSITE;*
13. *output PRIME;*

3.2.2 Time complexity analysis

Though correctness of this deterministic algorithm and has been rigorously proved in [1], the main issue in its implementation is the time complexity or the work done to decide the prime nature of a given number. It is quite obvious from the algorithm that the maximum work is done when a given number is prime. The first step in the algorithm takes asymptotically $O(\log^3 n)$ number of calculations to check if the given number is of a^b nature.

The *while loop* consisting of step 4 to step 10 of the algorithm, it uses to choose the most suitable r. It is proved [1] that there exists an *r* in the interval [$C_1(\log^6 n)$, $C_2(\log^6 n)$], which satisfies the condition $q \geq 4\sqrt{r}\log n$ and $q \mid O_r(n)$ for $n>n_o$, *q* is the largest prime factor of *r –1* and $O_r(n)$ is the order of *n*(mod *r)*. So the number of *r* values to be checked is of the order $\log^6 n$. The GCD can be evaluated using Euclid's Algorithm. This takes *poly (log log r)* steps. The check for primality of *r* and finding the largest factor *q* can be done by within $O\left(\sqrt{r}\,poly(\log r)\right)$ steps. The total complexity of the while loop is $O\left(r \bullet \sqrt{r}\,poly(\log r)\right) = O(\log^9 n)$. Though more efficient and sophisticated algorithm can be used but even a very straightforward and slow implementation of these does not effect the time taken by the algorithm as most of the time is spent in the painful congruence *for loop* from step 11 to 13.

The *for loop* iterates $\lfloor 2\sqrt{r}\log n \rfloor$ times, in one iteration of the loop, if repeated squaring and multiplication is used then *O(log n)* multiplication and squaring of polynomials of degree less than r are done. By using FFT for these polynomial and integer operations, a single iteration takes $O(r\log^2 n)$ time. Hence the time taken by the *for loop,* $O(r\log^2 n)$ becomes the total complexity of the algorithm.

3.3 AKS Algorithm Version 3

3.3.1 AKS Version 3 Algorithm

The algorithm was published in 2003. It has been improved from its original AKS Algorithm. And its time complexity is faster than its original. This algorithm also has three main steps to compute the input *n*. The algorithm listed below:

a) Perfect power checking the input n

b) Find r, $\mathrm{O}_r(n) \geq 4\log^2 n$

c) a=1 to $\lfloor 2\sqrt{\phi(r)}\log n \rfloor$ *check* $(x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$

The AKS Version 3 algorithm also needs three main steps to determine the input data. But there are some changes in the step 2 and 3, which increase it computation time. Therefore, its time complexity is faster than original AKS Algorithm. The more detailed time complexity analysis is stated after the algorithm below.

The algorithm shows below is AKS Version 3 [2]:

Input: integer $n > 1$.

1. If ($n = a^b$ *for* $a \in N$ *and b>1) output COMPOSITE.*

2. Find the smallest r such that $\mathrm{O}_r(n) \geq 4\log^2 n$ *.*

3. If 1< (a, n)<n for some $a \leq r$ *, output COMPOSITE.*

4. If $n \leq r$ *, output PRIME.*

5. For $a = \lfloor 2\sqrt{\phi(r)}\log n \rfloor$ *do*

if $\left((X+a)^n \equiv X^n + a \pmod{X^r - 1, n}\right)$*, output COMPOSITE;*

6. Output PRIME;

3.3.2 Time Complexity analysis

In step 1, it is same as original AKS Algorithm, so it requires $O(\log^3 n)$. In step 2, we find an r with $O_r(n) \geq 4\log^2 n$. This can be done by trying out successive values of r and testing if $n^k \neq 1 (\mathrm{mod}\, r)$ for every $k \leq 4\log^2 n$. For a particular *r*, this will involve at most $O(\log^2 n)$ multiplications modulo r and so will take time $\tilde{O}(\log^2 n \log r)$. We know that only $O(\log^5 n)$ different *r*'s need to be tried. Thus the total time complexity of step 2 is $\tilde{O}(\log^7 n)$.

The third step involves computing GCD of *r* numbers. Each GCD computation takes time $O(\log n)$ and therefore, the time complexity of this step is $O(\log n) = O(\log^6 n)$. The time complexity of step 4 is just $O(\log n)$.

In step 5, we need to verify $\lfloor 2\sqrt{\phi(r)} \log n \rfloor$ equations. Each equation requires $O(\log n)$ multiplications of degree r polynomials with coefficients of size $O(\log n)$. So each equation can be verified in time $\tilde{O}(r \log^2 n)$ steps. Thus the time complexity of step 5 is $\tilde{O}(r\sqrt{\phi(r)} \log^3 n) = \tilde{O}\left(r^{\frac{3}{2}} \log^3 n\right) = \tilde{O}(\log^{10.5} n)$. This time dominates all the other and is therefore the time complexity of the algorithm.

3.4 AKS Algorithm Conjecture

3.4.1 AKS Conjecture Algorithm

This algorithm was introduced when the first AKS paper was published in 2002, but this algorithm has not yet been proved for its correctness. This algorithm provides much faster time complexity than the others. It also has three main steps to compute the input *n*.

We will consider the three steps of the AKS Conjecture, and then look the its algorithm from its paper. They are listed below:

a) Perfect power checking the input n

b) Find r, such that r is not divide $n^2 - 1$

c) Check $(x-1)^n \equiv x^n - 1 \left(\mathrm{mod}\, x^r - 1, n\right)$

This algorithm was obtained from 'PRIMES is in P' [1].

Input: integer $n > 1$

1. *if (n is of the form* $a^b, b > 1$*) output COMPOSITE;*
2. $r = 2$;
3. *Find r such that r not divides* $n^2 - 1$.
4. *check* $(x-1)^n \equiv x^n - 1 \left(\mathrm{mod}\, x^r - 1, n\right)$*;*

3.4.2 Time Complexity analysis

The conjecture says that if r does not divide and if $(x-1)^n \equiv x^n - 1 \left(\mathrm{mod}\, x^r - 1, n\right)$, then wither n is prime or $n^2 \equiv 1 \left(\mathrm{mod}\, r\right)$.

If the conjecture is true, then above algorithm is valid. Such an r can assuredly be found in the range $\lceil 2,4 \log n \rceil$. This is because the product of prime numbers less than x is at least e^x. Then we can test whether the congruence in above conjecture holds or not.

Verifying congruence in conjecture takes time $\tilde{O}(r(\log^2 n))$ using FFT for multiplication. Overall, this would give us a time complexity of $\tilde{O}(\log^3 n)$.

3.5 AKS Algorithm by Daniel Bernstein

3.5.1 The AKS Algorithm with Daniel Bernstein Version

The algorithm was published in 2003 after the original AKS Algorithm was invented. It has made some improvement base on the original AKS and its time complexity is faster than the original AKS. The algorithm also has three main steps when computing the input integer number *n*. The algorithm will show below, and along with its complexity.

a) Perfect power checking the input n

b) Find r, q=LargestPrimeFactor(r) that $\binom{2q-1}{q} \geq 2^{2\lfloor\sqrt{r}\rfloor\log n}$ & $n^{\frac{r-1}{q}} \leq 1$

c) a=1 to $2\sqrt{r}\log n$ *check* $(x-a)^n \equiv x^n - a \pmod{x^r - 1, n}$

Below shows the algorithm by D. Bernstein:

Input: integer $n > 1$.

1. If ($n = a^b$ *for* $a \in N$ *and b>1) output COMPOSITE.*

2. Find r, q=LargestPrimeFactor(r) that $\binom{2q-1}{q} \geq 2^{2\lfloor\sqrt{r}\rfloor\log n}$ & $n^{\frac{r-1}{q}} \leq 1$

3. For a=1 to $a = \lfloor 2\sqrt{r}\log \rfloor$ *do*

if $((X-a)^n \not\equiv X^n - a \pmod{X^r - 1, n})$, *output COMPOSITE;*

4. Output PRIME;

3.5.2 Time complexity analysis

We have analyzed the time complexity of the original AKS algorithm before .We know that the most time-consuming step is the for-loop. We have to check ($q = 2\sqrt{r}\log n$) polynomial congruence equations. Each of these equations is time-consuming. So if we can reduce the size of these equations, we will speed up the algorithm. D. Bernstein [4, 5] proved that the number of polynomial congruence equations can be reduced to q such that $\binom{q+s-1}{q} \geq 2^{2\lfloor\sqrt{r}\rfloor\log n}$. Since the restriction of r is reflexes, we can find a smaller r too. Now the total running time of the *for loop* is $\tilde{\mathrm{O}}\left(r \cdot s \cdot \log^2 n\right)$, so the algorithm was speeded up.

Chapter 4

Requirements Specification

4.1 Introduction

The aim of this project is to implement functional requirements based on developing AKS Algorithm and other three algorithms. The requirement specifications of the development process therefore play an important role of this project. They describe, in a high level, what should to be achieved, and provide the basis to which the success of the final results will be measured.

The literature review discussed the basic approach of the AKS Algorithm itself, and also discussed the reasons for using the NTL Library to implement the AKS Algorithm. This chapter discusses the requirements specification that was gathered for implementing the system in order to achieve our initial objectives. The requirements are divided into three sub-sections. First one is requirements based on consideration of the project constraints; second one is based on required features to system functionalities, third one is to discuss about NTL, as importance in the later system development. Finally, we will look in to the system user interface requirements. Even though, this system does not require fancy user interface. But the user interface needs to be simple, usable, and efficient. In this chapter, we will use bullet points to list all necessary requirements in each sub-section.

4.2 Project requirements

The project requirements were important to consider the constraints that would necessarily be placed upon the project, the obvious one is being the limited time available. The project development and dissertation writing are only a few months away from the deadline. Therefore, time is the key in the whole process. The project requirements are listed as follows:

- The system must consistently develop.
- A suitable programming language must choose quickly.
- The system error checking must carry out at each development stage.
- The system must contain no unnecessary functionality.

4.3 Functional requirements

The functional requirements must be simple and consistent and must cover all necessary functionalities, which the system required. The following requirements listed are essential to the system.

4.3.1 Input Data

- The input type should only be large positive integers.
- The system should be able to store the integer size over 10- or 20-decimal-digit.
- The system should not allow user to input any illegal characters such as punctuations, alphabet characters in both upper and lower case, and so on.

4.3.2 Output

- The system must be capable of displaying the correct output results.
- The system must keep all the testing data.
- The system should be able to compute any arbitrary long integer input.

4.3.3 Error handling

- All errors must be handled by the system, and must not lead to a system failure.
- The system must response when user input incorrect data.

4.3.4 System Performance

- The system must always remain responsive.
- The system must indicate when system is performing process.
- The algorithms should be implemented efficiently.
- The system should be stable.
- The memory should be freed automatically, when system has shutdown.

4.4 NTL requirements

- The minimal platform for installing NTL must above Windows 95 or NT platform.
- A C++ compiler is needed for build the NTL library.
- The NTL library should be quickly downloaded and unpacked in the specific directory.
- The NTL should be quickly compiled and configured with the chosen running environment.

- The NTL should be tested after compile and configured. If there are errors, they should be fixed quickly.

4.5 User interface requirements

- The system should provide a place for user to input the numbers.
- The system should allow user to run the algorithm by pressing buttons.
- The system should display the results for the user.
- The system should allow the user to stop the computation process at any time by pressing buttons.
- The displayed results should contain following information, computing time, r, and q, and Y / N (for telling the user that the input number whether a prime or not).
- The system should keep all the testing samples for the user.

Chapter 5

Design

5.1 Introduction

This chapter mainly discusses some of the main designs of the system which were made when planning the initial design. This includes justification of the programming language chosen for the implementation of this project, the user interface design. The four algorithms we are going to implement have been already exited. So there will no need to redesign them. But we will discuss some basic algorithms needed to implement these four algorithms.

5.2 Choosing Programming Language

One of the first important decisions to make during the design stage was which programming language to use for the implementation of the project. The first choice pop in to my mind was Java, which the programming language I used most in the past two years of my studies at university. There is no reason to learn any unfamiliar languages in this stage. But when I was discussing with my project supervisor at starting of the semester 1, he said that the most efficient programming language to develop this project is C++ together with NTL Library, and he also point out some important features which the best doing in this project. Afterwards, I had further thought about choosing the correct programming language. Finally, I decided to use Microsoft Visual C++ 6.0 to develop my project. This was preferred for several reasons. Firstly, the C++ programming language which I was unfamiliar with, and it takes time to get used to it. Because of time consuming, so I thought that Microsoft Visual C++ 6.0 is very similar to C++ and

simpler than C++. It also provides a lot of exist features ready to use such as interface implementation. Secondly, the NTL Library is also easy to link with. The reason to use NTL Library is, it provides algorithms for arbitrary length integers, for vectors, matrices, and polynomials over the integers and over finite fields. NTL's polynomial arithmetic is one of the fastest available anywhere. It has been mentioned in the chapter 2.

Overall, Microsoft Visual C++ 6.0 together with NTL Library is the most suitable choice for developing this project, and it will be quick, simple, and efficient.

5.3 Basic Algorithms

At this section, we are going to consider some basic algorithms which needed in order to implement these four algorithms which we mentioned earlier. We will briefly talk those basic algorithms which used in each step of the AKS algorithms. We will use these main steps in each algorithm which we mentioned in chapter 3. The three steps in these four algorithms are: step1 perfect power check, step 2 while loop, step 3 for loop. We will list the basic algorithms required in order to implement these three steps. More details of these basic algorithms will be discussed in the next chapter.

Step 1: For a fixed k, we can check whether n is a perfect kth power in polynomial time. We can use Newton's method on $x^b - n$ to estimate $n^{1/b}$ to within .5 and then check whether the kth power of the nearest integer is n. So, the algorithms we used are perfect power algorithm and Newton's method.

Step 2: In this step we need to find a prime r which $ord_r n$ is fairly large. First, need to use GCD algorithm to check r with n, second use IsPrime algorithm to check r, third, to find largest prime factor of r-1. So the algorithms we used are GCD, IsPrime, largest prime factor, order.

Step 3: The last step is the most time consuming step in the algorithms. In this step we need to check $(x+a)^n \equiv x^n + a \left(\mathrm{mod}\, x^r - 1, n\right)$ from $a=1$ to r. We could use repeated

squaring and multiplication of polynomials of degree less than *r*. So the algorithm we used in this step is repeated squaring and multiplication.

5.4 User Interface

The user interface contains two parts; user input and user feedback. They are based on previous requirements chapter. We will discuss how we are going to design the system user interface in order to meet our requirements.

5.4.1 User Input

In order for the system to be usable it must provide a place to enable the user to provide input. This action should be achieved by using a mouse and keyboard. In order to do this action, the user need to move the mouse to the input place and click, then use keyboard to type in the data.

The system will offer several different buttons for user to choose appropriate algorithms which they want to use in order to run the test. User may also choose to clear up the input data or quit the system by simply press appropriate buttons.

5.4.2 User Feedback

The system must provide a space to display all necessary information which user required, such as whether the input number is prime number, executing time and related *r* and *s*. The system must keep all the past testing samples in ordering (the latest sample at the top), which for the user doing results testing.

The system must provide the user with information regarding its current status. Doing so at regular intervals will reassure the user that the system has not failed. It will achieve

this in two ways. The mouse pointer displayed will be indicative of the system's current state. If the system is busy calculating then the mouse will show an appropriate icon. The position of the mouse cursor will be constantly updated, even when the system is busy. Figure below shows the design of the user interface:

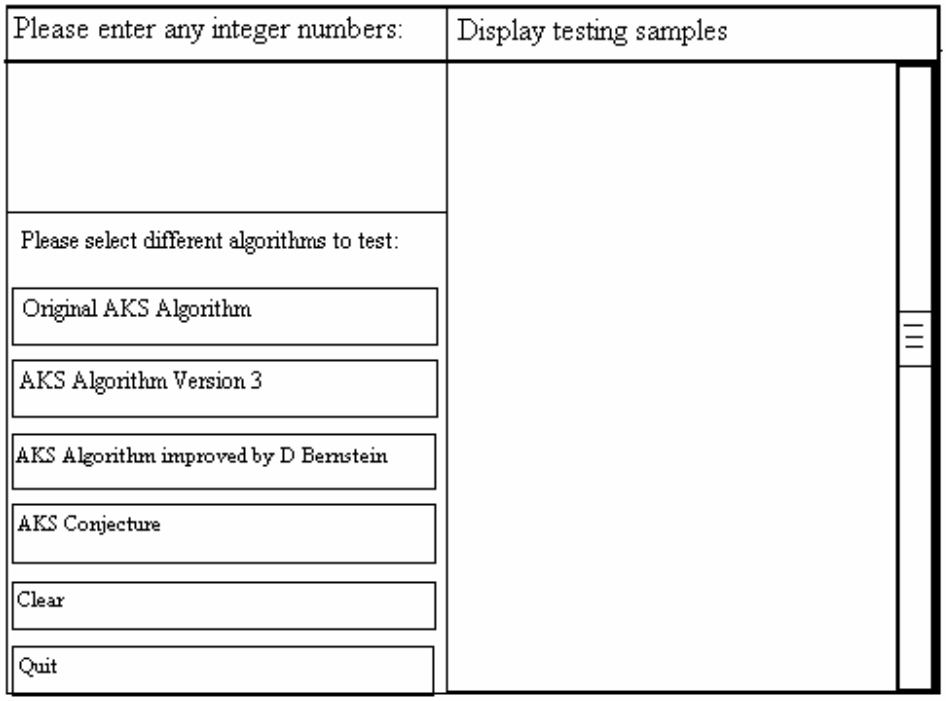

Figure 2: User Interface Design

Chapter 6

Implementation

6.1 Introduction

The implementation will look at the actual system that was developed. Some of the key features and algorithms will be discussed. We are not concentrating on those four algorithms such as AKS original, AKS Version 3, AKS Conjecture, and AKS improved by D. Bernstein. Instead we will discuss the key algorithms which involved in each step when those algorithms, such as perfect power check, Greatest Common Divisor (GCD), and so on. We will also look in to the user interface implementation. Beyond those we will discuss some features which use from the NTL.

6.2 Simple Primality test.

In the three AKS algorithms, such as original AKS Algorithm, AKS Version 3, and AKS algorithm improved by D. Bernstein. We have to find and suitable *r* which is a prime. So we need a primality test routine. Though we can use the AKS algorithm recursively to implement this purpose, we will not do that because the AKS algorithms are so slow for large integer. We use the following simple algorithm.

1.Primality check (n)
2. {
3. *For i =2 to* $\lfloor \sqrt{n} \rfloor$
4. *{*
5. *If* (*i*|*n*) *return FALSE ;*
6. *}*
7. *return TRUE;*
8.}

This simple primality testing algorithm will speed up the program greatly. But we have to know all the primes smaller than $\lfloor \sqrt{n} \rfloor$. In this project, we will store the first 1000 primes in an array. If *i* exceed the 1000^{th} prime, we will check the odd numbers from the 1000 prime to $\lfloor \sqrt{n} \rfloor$.

6.3 Perfect Power check

The perfect power check is the first step in all those four algorithms, and it is important in the algorithm. Detecting perfect powers is required by some factorization algorithms. If a prime divisor *p* with multiplicity *e* can be found, then only roots which are divisors of *e* need to be considered, much reducing the work necessary. To this end divisibility by a set of small primes is checked.

It is efficient step to check the input number. If the input number fail in this step then there is no need to go any farther, then the program can terminate at this point. Below is the algorithm which uses in the program development:

---

1. Power check (n)
2. {
3. *For b =2 to* $\lfloor \log n \rfloor$
4. *{*
5. $s = n^{1/b}$;
6. *If* $\left(s^b = n\right)$ *return TURE;*
7. *Else return FALSE;*
8. *}*
9. }

In the above algorithm, we have to compute the approximate *bth* integer root of an integer number. We use the Newton Method to implement it. The following is a description in details.

Our input data required large positive integer and large integers and high-precision approximate numbers are stored as arrays of base 2^{16} or 2^{32} digits, depending on the lengths of machine integers. Precision is internally maintained as a floating-point number. `IntegerDigits` and related base conversion functions use a recursive divide-and-conquer algorithm. `N` uses an adaptive procedure to increase its internal working precision in order to achieve whatever overall precision is requested. `Floor`, `Ceiling` and related functions use an adaptive procedure similar to `N` to generate exact results from exact input.

As we mentioned earlier at AKS Algorithm complexity analysis section, so $n^{1/b}$ could be calculated by integer Newton method, which is the efficient way to do. The algorithm is showing below:

---

1. Find root (n, b) // this algorithm returns $n^{1/b}$

2. {

3. $P = 2^{\lfloor B(n)/b \rfloor + 1}$*; // B(n) returns bit length of n*

4. While (1) {

5. $Q = \lfloor ((b-1)p + \lfloor n / p^{b-1} \rfloor) / b \rfloor$*;*

6. If $(Q > P)$ *return P;*

7. P=Q;

8. }

9. }

6.4 Largest factor Algorithm

The integer prime-factorization (also known as prime decomposition) problem is: given a positive integer, write it as a product of prime numbers. For example, given the number 45, the prime factorization would be $3^2 \cdot 5$. This algorithm is placed in the step 2 *while loop* of AKS Algorithms except AKS Conjecture. $3^2 \cdot 5$. This algorithm is placed in the step2 *while loop* of AKS Algorithms except AKS Conjecture. It needs to find out the largest prime factor of *r*. We ues a simple algorithm to implement. In this algorithm, we test the integers from 2 to $\lfloor \sqrt{n} \rfloor$ until we find the largest divisor. The algorithm is showing below:

```
1. Largest factor (p)
2. {
3.      s = p;
4.      i = 2;
5.      j = ⌊√s⌋;
6.      While ( i ≤ j ) {
7.                              If ( s = 0(mod i))
8.                              {
9.                                      s = s/i;
10.                                     j = ⌊√s⌋;
11.                             }
12.                             Else i = i + 1;
13.                     }
14.     Returns;
15. }
```

6.5 Euclidean Algorithm

The Euclidean algorithm, also called Euclid's algorithm, is an algorithm for finding the **Greatest Common Divisor** (also write as GCD) of two numbers *a* and *b*. The algorithm can also be defined for more general rings than just the integers $\mathbb{Z}$. There are even principal rings which are not Euclidean but where the equivalent of the Euclidean algorithm can be defined. This is required at the second step in the algorithm to check random *r* and input number *n*.

The **Euclidean Algorithm** to compute the Greatest Common Divider for two integers *a* and *b* (not zero) is based on the following fact:

If *r* is the remainder when *a* is divided by *b*, then gcd(*a*,*b*)=gcd(*b*,*r*).

The algorithm can be written in pseudo-code as follows:

```
1. Eculid (a, b) {
2.      While (b not 0) {
3.              Interchange (a, b)
4.              B := b mod a
5.      }
6.      Return (a)
7. }
```

6.6 Compute the Order of a modulo r

In the AKS Version algorithm, we have to compute the order of an integer *a* modulo *r*. The order of a modulo *r*, which *a* and r is coprime, is defined as the smallest positive integer *k* such that $a^k=1(\text{mod } r)$. The algorithm can be written in pseudo-code as follows:

```
1. Order (a, r)
2.{
3.      m=a;
4.      for i=1 to r{
5.              If(m=1(mod r)) return i;
6.              Else {
7.                      i=i+1;
8.                      m=m*a(mod r);
9.              }
10.     }
11. }
```

6.7 Compute the Euler Function $\psi(n)$

In the AKS Version 3 algorithm and some of other improvements, we have to compute the Euler function $\psi(n)$ which is define the number of all the integers that is smaller than *n* and is coprime to *n*. We simply test all the integers that is smaller than n and count the number of those integer that is coprime to *n*. The algorithm can be written in pseudo-code as follows:

```
1. Euler (n)
2.{
3.      count=0;
4.      for i=1 to n-1{
5.              If(GCD(I,n)=1) count=count+1;
6.      }
7.      return count;
11. }
```

6.8 Implementation of the congruence

The congruence computation is the most costly step in these algorithms, so the most improvements are focus on the last step of the AKS algorithms. At this section we are discussing the congruence algorithm developments in the AKS algorithms.

The evaluation of $(x-a)^n$ using repeated squaring could be represented as

$$y^n = \left(\left(\left(\left(y^{b_l} \right)^2 y^{b_{l-1}} \right)^2 y^{b_{l-2}} \ldots \right)^2 y^{b_1} \right)^2 y^{b_0}$$

Where y = x – a, $n = \sum_{i=0}^{l} b_i 2^i$ and $l = \lfloor \log n \rfloor + 1$

For example: $y^{53} = \left(\left(\left(\left((y)^2 y \right)^2 1 \right)^2 y \right)^2 1 \right)^2 y$

Let P(x) to be a polynomial whose nth power is to be computed and PXP(x) be the polynomial which contains $(P(x))^2$. The following algorithm computes $F(x) = (P(x))^n$.

6.8.1 *Computing Power via Repeat Square Algorithm*

In the polynomial congruence equation checking step, we have to compute a power of polynomial. If we simply multiply the polynomial to kth power, we have to do the polynomial multiplication operation for *k* times. But if we use a method called repeat square algorithm, we need only $\log k$ polynomial multiplications. The following is the algorithm:

---

1. { F(x) = 0 for x>0; F(x) = 1 for x = 0;
2. For i = l to 0 {
3. PXP(x) = $(F(x))^2$ *;*
4. If $b_i = 1$
5. F(x) = PXP(x) × P(x);
6. Else
7. F(x) = PXP(x);
8. }
9. }

6.8.2 Implementation of congruence through basic polynomial multiplication and squaring

Computation of the RHS of the congruence in step 12 of AKS Algorithm

$(x^n - a)$ can be represented as $(x^r - 1)(x^{n-r} + x^{n-2r} + x^{n-3r} + \cdots + x^{n-kr}) + x^{n-kr} - a$.

So the RHS = $x^{n \bmod r} - a \pmod{x^r - 1, n}$ as *n-nk* is *n mod r*.

Computation of the LHS of the congruence in step 12 of AKS Algorithm

$n = \sum_{i=0}^{l} b_i 2^i$ and $l = \lfloor \log n \rfloor$

P[0]=-a; P[1] = 1; //P(x) = x-a

F[0] = 1; F[i] = 0 For all i;

PXP [j] = 0 For all j;

degF = degPXP = 0;

// degF and degPXP are the degress of the F(x) and PXP(x) respectively

For i = l to 0

{

For i = 0 to degPXP

PXP[j] = 0;

For i = 0 to degF // PXP = F × F

For j = 0 to degF

PXP[(j+i)mod r] = (PXP[(j+i)mod r] + P[i] × P[j]) mod n;

degPXP = 2 × degF;

if (degPXP ≥ r) degPXP = r-1;

if (b_i =1) // F = PXP × Y

{

For j = 0 to degF

F[j] = 0;

For i =0 to degPXP

```
            For j = 0 to 1
            F[(j+i) mod r ] = (F[(j+i) mod r ] + PXP[i] × P[j] ) mod n;
        }
        Else   // F = PXP
        {
            For j =0 to degPXP
            F[j] = PXP [j];}
}
```

6.9 User Interface

According to the user interface requirements specification and design, we should implement our system user interface as it required, and try to not fail to meet any requirements.

The system provides an edit field which for user to input arbitrary integer numbers. Then user can select different algorithms to test their inputs by pressing buttons. Depending on the input number's size and different algorithm, the system could take while to compute the input number. Once the results have been computed, they will display at the Right-Hand-Side which called display field.

The system also makes use of mouse cursors to indicate what the system is currently doing. They change of mouse action depending upon the user's action. The second cursor shows a timer, and this is displayed when the system is computing. The pictures below show the mouse actions.

When system does not computing.

When system computing.

The representation of the output data will include computational time, random *r* and *s*, and Y/N (for telling the input number whether or not a prime). We also need to insure that the user can only input positive integer numbers, if the user tries to input illegal input data then the system will inform the user.

6.10 NTL Description

At this section will mainly concern of NTL Library, such as how important it is, how it involved in the system design process. The NTL is already exiting recourse. Therefore, we do not need to consider its design in pervious chapter. As mentioned in the chapter 2 that NTL is a high-performance, portable C++ library providing data structures and algorithms for arbitrary length integers; for vectors, matrices, and polynomials over the integers and over finite fields; and for arbitrary precision floating point arithmetic. Because NTL is good at dealing with number theories, and also according to its performance, so it highly recommended in this project development. We will use existing algorithms and arithmetic in the NTL to develop our algorithms. The source we needed it already existed in NTL. Therefore, we do not need to consider of implementing the NTL, we just need to use its sources to implement the algorithms. But we need to consider of how to compile and configure NTL, and make it ready to use. More details of NTL installation, compiling, and configuration see appendix A.

The four algorithms we are developing require a lot of mathematical algorithms and arithmetic over integers or polynomials, such as congruence, GCD (Great Common Divisor), big integral, polynomial computations, and so on. NTL provides such support for these materials, such as big integral(ZZ) which the class ZZ implements signed, arbitrary length integers, polynomial(ZZ_x), congruence(ZZ_p) which the algorithm to deal with congruence, polynomial ring(ZZ_pX) which the class ZZ_pX implements polynomial arithmetic modulo p, The multiplication and squaring routines use routines from ZZX, which is faster for small degree polynomials and so on.

Chapter 7

Testing

7.1 Introduction

The aim of this project is to implement the AKS Algorithm and to measure its running time results. Because we have include other three algorithms in our development, so we will look into their running time results and compare them with original AKS Algorithm. In order to test the system we need to compare it with initial requirements, and the requirements of the project were laid out in chapter 4 of this dissertation. The testing of the system involves the analysis the system in order to validate that the system meets its initial requirements. We are mainly concerning to test system's functional requirements and its user interface. This includes checking that the design and the implementation meet the requirements. After checking the system requirements validation, we will discuss the results measurements. In this section, we give various input data to the system. After we obtain running results from the system, then we gather all the running results together to measure them, and try to find out the some conclusions from them. After all we could conclude our testing.

7.2 Requirements Testing

The requirements of the system were split in to three main categories: functional requirements and user interface requirements and NTL. We will discuss the functional requirements and user interface requirements in this section. It is recommended that the

requirements specification referred to in order for the demands of the requirements to be understood again. To make the comparison between the requirements and the final system, the discussion will be presented in the same structure as the requirements.

7.2.1 Functional Testing

Input Data

The system is capable to store any large integer and with any arbitrary size. The system is also capable to distinguish between legal and illegal inputs from the user.

Output

The system's output results are correct and as expected. The system also keeps the past tested data in the Right-Hand-Side of the system. But there is only one problem which the latest testing sample always stores at the last of the testing queen.

Error Handling

The system runs really slow when the input data getting larger and larger and it has no response when testing really integer number. On the other hand, the system handles illegal input well, when the users try to input incorrect data, the system always response and inform user.

System Performance

The system capable to test any input data at any time, and also indicates the user when it is in the middle of computation. When the system is computing large integer the progress is slow. The memory always be freed when system not in computation.

7.2.2 Interface Testing

The system's user interface provides all the functionalities which required in the requirements specification chapter. The output representation of data is exactly what the user required. We provide the existing system user interface below to show that we have to meet the user interface requirements.

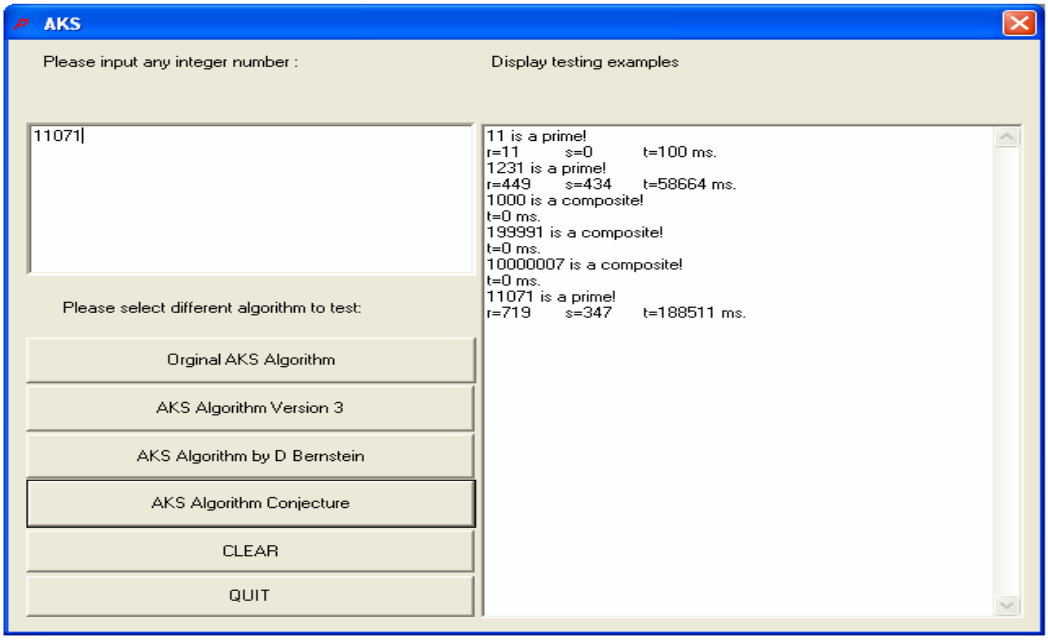

Figure 3: Existing System

7.3 Results Measurement

7.3.1 Testing Environment

The testing process is carried out at my personal laptop. The properties of the machine are: IBM A31p with Microsoft Windows XP installed, Mobile Intel(R) Pentium(R) 4-M CPU 2.00 GHz 718 MHz, 1.00 GB of RAM.

7.3.2 Running Results Measurement

1. The Frequency of Perfect Power.

In the AKS Algorithm, we have to check whether the given number is a perfect power before the while-loop. If we omit this step, we can only prove that the given number is either a prime or a power of prime after the for-loop. The proof can be found in Bernstein's article [4, 5] or other related article. Since a trivial algorithm for *perfect power* take only O $((\log n)^3)$, we can check it before the time-consuming loop. Here is a table of the frequency of perfect power. We give the frequency of perfect power of different length. (PP stands for power, PPP stands for perfect power of prime.)

Bits	Total	PP	Total/PP	Log(Total/PP)	PPP	Total/PPP	Log(Total/PPP)
2	2	0	0	0	0	0	0
3	4	1	4	2	1	4	2
4	8	2	4	2	2	4	2
5	16	3	5	2.41504	3	5	2.41504
6	32	3	10	3.41504	2	16	4
7	64	5	12	3.67807	4	16	4
8	128	7	18	4.19265	3	42	5.41504
9	256	8	32	5	4	64	6
10	512	11	46	5.54057	6	85	6.41504
11	1024	16	64	6	5	204	7.67807
12	2048	24	85	6.41504	9	227	7.83007
13	4096	32	128	7	10	409	8.67807
14	8192	42	195	7.60768	11	744	9.54057
15	16384	61	268	8.06926	17	963	9.91254
16	32768	82	399	8.64245	15	2184	11.0931
17	65536	118	555	9.11736	26	2520	11.2996
18	131072	166	789	9.62496	31	4228	12.0458

19	262144	231	1134	10.1483	39	6721	12.7146
20	524288	322	1628	10.6691	53	9892	13.2721

Table 1: Frequency of perfect power

From this table we can find out that the 5^{th} column and the 8^{th} column increase nearly linearly, which mean that the frequencies of these kinds of numbers in 1 to *n* are mostly a polynomial of the length of *n*. We can found from the above table that most of the prefect powers are composite, so the algorithm runs faster if we add the perfect power checking step.

2. The Frequency Composites and Primes that pass each stage of the algorithms

In the original AKS algorithm, the while-loop takes at most O $((\log n)^6)$ time and the *for loop* takes O$((\log n)^{12})$ time. The less number's pass the first loop, the faster AKS Algorithm runs. We get some statistic data of the frequency of the primes and composites by running these algorithms from 4 bits integer to 18 bits integer. It is a time-consuming work. The following is the tables we get. We group the statistic data by the input length. *P1* denotes the number of primes verified by the *while loop*, which mean no suitable *r* is found. *P2* denotes the number of primes verified by the most time-consuming polynomial congruence. Actually, our data show that all large prime numbers are verified in this way. *C0* is the number of composites decided by perfect power routine. *C1* is the number of composites decided by the *greatest common divisor* routine in the *while loop*. Ours data show that almost all the composites decided in this routine. *C2* is the number of all the composites that fail the polynomial congruence, which is zero up to present. *C* denote the composite and *P* denote the prime.

Bits	Total	P	P1	P2	C	C0	C1	C2
4	8	2	2	0	6	2	4	0
5	16	5	5	0	11	3	8	0

6	32	7	7	0	25	3	22	0
7	64	13	13	0	51	5	46	0
8	128	23	23	0	105	7	98	0
9	256	43	43	0	213	8	205	0
10	512	75	75	0	437	11	426	0
11	1024	137	137	0	887	16	871	0
12	2048	255	255	0	1793	24	1769	0
13	4096	464	464	0	3632	32	3600	0
14	8192	872	383	489	7320	42	7278	0
15	16384	1612	0	1612	14772	61	14711	0
16	32768	3030	0	3030	29738	82	29656	0
17	65536	5709	0	5709	59827	118	59709	0
18	131072	10749	0	10749	120323	166	120157	0

Table 2: Frequency of primes and composites decided in different step by the original AKS Algorithm.

Bits	Total	P	P1	P2	C	C0	C1	C2
4	8	2	2	0	6	2	4	0
5	16	5	5	0	11	3	8	0
6	32	7	7	0	25	3	22	0
7	64	13	13	0	51	5	46	0
8	128	23	23	0	105	7	98	0
9	256	43	4	39	213	8	205	0
10	512	75	0	75	437	11	426	0
11	1024	137	0	137	887	16	871	0
12	2048	255	0	255	1793	24	1769	0
13	4096	464	0	464	3632	32	3600	0

14	8192	872	0	872	7320	42	7278	0
15	16384	1612	0	1612	14772	61	14711	0
16	32768	3030	0	3030	29738	82	29656	0
17	65536	5709	0	5709	59827	118	59709	0
18	131072	10749	0	10749	120323	166	120157	0

Table 3: Frequency of primes and composites decided in different step by the AKS Version 3

Bits	Total	P	P1	P2	C	C0	C1	C2
4	8	2	2	0	6	2	4	0
5	16	5	5	0	11	3	8	0
6	32	7	7	0	25	3	22	0
7	64	13	13	0	51	5	46	0
8	128	23	23	0	105	7	98	0
9	256	43	15	28	213	8	205	0
10	512	75	0	75	437	11	426	0
11	1024	137	0	137	887	16	871	0
12	2048	255	0	255	1793	24	1769	0
13	4096	464	0	464	3632	32	3600	0
14	8192	872	0	872	7320	42	7278	0
15	16384	1612	0	1612	14772	61	14711	0
16	32768	3030	0	3030	29738	82	29656	0
17	65536	5709	0	5709	59827	118	59709	0
18	131072	10749	0	10749	120323	166	120157	0

Table 4: Frequency of primes and composites decided in different step by the AKS Algorithm improved by D. Bernstein

From above tables we come up with some conclusions and they are listed below:

1. When the input number is small, we can not find a suitable *r*, so all these small primes are verified by simple division (the GCD routine).
2. The D. Bernstein's version and the AKS Version 3 find the suitable *r* first. The first version of AKS is the latest one to find that *r*.
3. We can find suitable *r* for all large primes.
4. No composite survives after the while-loop in our tables.

3. The Found *r*.

Table below shows the first *r* found with corresponding *n* in three different algorithms.

Algorithm	*n*	*r*
Original AKS	11701	11699
AKS Version 3	271	269
D. Bernstein Improved AKS	349	347

Table 5: First suitable r found by these algorithms

The AKS Version 3 algorithm finds the suitable *r* first. AKS algorithm improved by D. Bernstein also finds suitable *r* from small integers. The input has only 9 bits long. Although, these two algorithms find suitable *r* early, they work slowly for small input integers, because they have to verify a mount of polynomial congruence. So the original AKS algorithm is better for small input.

4. The Key Performance of these three above algorithms

We have mentioned that the most time-consuming step is the *for loop*. We have to check the polynomial congruence for a set of *a*. The polynomial operation is complex and slow. If we can reduce the size of polynomial congruence equations (denoted by *s*). We could

improve the performance of these AKS algorithms. Now we list some of them below. *AS* stand for average size of congruence equations we have to check. *AKS* stands for original AKS Algorithm, *V3* stands for AKS Version 3 algorithm, *DB* stands for D. Bernstein improved AKS Algorithm and *P2* is denoted same as table 4.

Bits	Total	P	P2(AKS)	Ave(AKS)	P2(V3)	Ave (V3)	P2(DB)	Ave (DB)
4	8	2	0	0	0	0	0	0
5	16	5	0	0	0	0	0	0
6	32	7	0	0	0	0	0	0
7	64	13	0	0	0	0	0	0
8	128	23	0	0	0	0	0	0
9	256	43	0	0	39	302	28	150
10	512	75	0	0	75	371	75	177
11	1024	137	0	0	137	451	137	217
12	2048	255	0	0	255	541	255	258
13	4096	464	0	0	464	637	464	305
14	8192	872	489	3046	872	743	872	353
15	16384	1612	1612	3401	1612	855	1612	405
16	32768	3030	3030	3885	3030	974	3030	474
17	65536	5709	5709	4404	5709	1104	5709	538
18	131072	10749	10749	4947	10749	1240	10749	600

Table 6: Average Number of the polynomial congruence equations

From the above table we can see that the average polynomial congruence equation size of D. Bernstein's improvement is the smallest. So it is the fastest one of these three algorithms. The original AKS Algorithm has the largest average size if suitable r is founded. So when n>11701, it's the worst one. The AKS Version 3 algorithm over performs the original AKS, but it is still slower than D. Bernstein's improvement.

5. Compare Average case and Worst case

Original AKS Algorithm

In the paper of original AKS algorithm [1], the author give a lemma which show that the r found is in $O(\log^6 n)$, we will check this lemma by the following table. The table shows below the average case:

Bits	Ave(r)	Ave(s)	Ave(r)/Bits^2	Ave(r)/Bits^3	Ave(r)/Bits^4	Ave(r)/Bits^5	Ave(r)/Bits^6
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0
14	12231	3045	62.40306	4.457362	0.318383	0.022742	0.001624
15	13652	3400	60.67556	4.045037	0.269669	0.017978	0.001199
16	15599	3884	60.93359	3.80835	0.238022	0.014876	0.00093
17	17693	4403	61.22145	3.601262	0.211839	0.012461	0.000733
18	19854	4946	61.27778	3.404321	0.189129	0.010507	0.000584

Table 7: Range of *r* found by original AKS Algorithm

From this table we can find out that when $k \geq 3$, Ave(r)/Bits^k decreases monotonously when the input length (Bits) increases. This result show that the expectation of *r* found by AKS algorithm is $O(\log^3 n)$. Since $s=2\cdot\sqrt{r}\cdot \log n$ and the time complexity of the *for loop* is $\tilde{O}\left(r\cdot s\cdot \log^2 n\right)$, the average complexity of original AKS seems to be $\tilde{O}\left(\log^{7.5} n\right)$.

For each input of given length, we find their worst length (max length). They are shown in the following table.

Bits	max(r)	max(s)	max(r)/Bits^2	max(r)/Bits^3	max(s)/Bits^2	max(s)/Bits^3
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0	0	0	0	0	0
13	0	0	0	0	0	0
14	12647	3148	64.52551	4.608965	16.06122	1.14723
15	14423	3602	64.10222	4.273481	16.00889	1.067259
16	16487	4108	64.40234	4.025146	16.04688	1.00293
17	18587	4635	64.31488	3.783228	16.03806	0.943415
18	21059	5224	64.99691	3.61094	16.12346	0.895748

Table 8: Max *r* and *s* found by original AKS Algorithm

AKS Version 3 algorithm

The following table is the average length of r and s found by AKS Version 3. Table below shows the average case:

Bits	Ave(r)	Ave(s)	Ave(r)/Bits^2	Ave(r)/Bits^3	Ave(r)/Bits^4	Ave(r)/Bits^5
4	0	0	0	0	0	0

5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	311	302	3.839506	0.426612	0.047401	0.005267
10	379	371	3.79	0.379	0.0379	0.00379
11	458	450	3.785124	0.344102	0.031282	0.002844
12	549	540	3.8125	0.317708	0.026476	0.002206
13	645	636	3.816568	0.293582	0.022583	0.001737
14	753	742	3.841837	0.274417	0.019601	0.0014
15	864	854	3.84	0.256	0.017067	0.001138
16	983	974	3.839844	0.23999	0.014999	0.000937
17	1114	1104	3.854671	0.226745	0.013338	0.000785
18	1248	1239	3.851852	0.213992	0.011888	0.00066

Table 9: Range of *r* and s found by AKS Version 3

From the above table we can find out that when k>3 Ave(r)/$Bits^k$ decreases monotonously when the input length (Bits) increases. We can also find out that Ave(s) is slightly less than Ave(r).

The table below shows the worst case:

Bits	max(r)	max(s)	max(r)/$Bits^2$	max(r)/$Bits^3$	max(s)/$Bits^2$	max(s)/$Bits^3$
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	349	334	4.308642	0.478738	4.123457	0.458162

10	443	417	4.43	0.443	4.17	0.417
11	529	500	4.371901	0.397446	4.132231	0.375657
12	641	605	4.451389	0.370949	4.201389	0.350116
13	757	714	4.47929	0.344561	4.224852	0.324989
14	839	809	4.280612	0.305758	4.127551	0.294825
15	997	932	4.431111	0.295407	4.142222	0.276148
16	1117	1062	4.363281	0.272705	4.148438	0.259277
17	1259	1205	4.356401	0.256259	4.16955	0.245268
18	1433	1361	4.42284	0.245713	4.200617	0.233368

Table 10: Max *r* and s found by AKS Version 3

AKS algorithm improved by D. Bernstein

The following table is the average length of *r* and *s* found by the AKS algorithm improved by D. Bernstein. The table below shows the average case:

Bits	Ave(r)	Ave(s)	Ave(r)/Bits^2	Ave(r)/Bits^3	Ave(r)/Bits^4	Ave(s)/Bits^2	Ave(s)/Bits^3	Ave(s)/Bits^4
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0
9	347	149	4.283951	0.475995	0.052888	1.839506	0.20439	0.02271
10	375	176	3.75	0.375	0.0375	1.76	0.176	0.0176
11	468	217	3.867769	0.351615	0.031965	1.793388	0.163035	0.014821
12	546	258	3.791667	0.315972	0.026331	1.791667	0.149306	0.012442
13	710	304	4.201183	0.323168	0.024859	1.798817	0.138371	0.010644
14	767	352	3.913265	0.279519	0.019966	1.795918	0.12828	0.009163

15	848	405	3.768889	0.251259	0.016751	1.8	0.12	0.008
16	984	473	3.84375	0.240234	0.015015	1.847656	0.115479	0.007217
17	1147	538	3.968858	0.233462	0.013733	1.861592	0.109505	0.006441
18	1257	599	3.87963	0.215535	0.011974	1.848765	0.102709	0.005706

Table 11: Range of *r* and s found by AKS algorithm improved by D. Bernstein

From the above table we can found that the average length of *r* found by this AKS algorithm improved by D. Bernstein is slightly larger than which is found by the AKS Version 3, but the average length of *s* is smaller than it and it is less than half.

The table below shows the worst case:

Bits	max(r)	max(s)	max(r)/Bits^2	max(r)/Bits^3	max(s)/Bits^2	max(s)/Bits^3
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	347	158	4.283951	0.475995	1.950617	0.216735
10	467	196	4.67	0.467	1.96	0.196
11	479	233	3.958678	0.35988	1.92562	0.175056
12	563	278	3.909722	0.32581	1.930556	0.16088
13	719	326	4.254438	0.327264	1.928994	0.148384
14	863	384	4.403061	0.314504	1.959184	0.139942
15	887	435	3.942222	0.262815	1.933333	0.128889
16	1187	505	4.636719	0.289795	1.972656	0.123291
17	1283	571	4.439446	0.261144	1.975779	0.116222
18	1307	647	4.033951	0.224108	1.996914	0.11094

Table 12: Max *r* and s found by AKS algorithm improved by D. Bernstein

6. The Running Time of algorithm based on AKS Conjecture

This algorithm does not like the other three algorithms. It runs extremely faster than any one of others. Even for larger integers. Since those three algorithms are too slow for a larger input, so we can not compare AKS Conjecture to the other three algorithms for some larger inputs. The following we did some comparisons of the average running time of simple primality algorithm and the algorithm based on AKS Conjecture. The simple primality algorithm here is a slightly revised version of the trivial algorithm of dividing the input number from 2 to its square root. We store the first 1000 primes in memory to accelerate it. *P* stands for prime number, *C* stands for composite number.

Bits	Total	P	C	Ave. Time for P (Simple)	Ave.Time for C (Simple)	Ave.Time for P (AKSC4)	Ave.Time for C (AKSC4)
4	8	2	6	0	0	0	0
5	16	5	11	0	0	0	0
6	32	7	25	0	0	0	0
7	64	13	51	0	0	0	0
8	128	23	105	0	0	0	0
9	256	43	213	0	0	0	0
10	512	75	437	0	0	0	0
11	1024	137	887	0	0	0	0
12	2048	255	1793	0	0	0	0
13	4096	464	3632	0	0	0	0
14	8192	872	7320	0	0	0	0
15	16384	1612	14772	0	0	0	0
16	32768	3030	29738	0	0	0	0

17	65536	5709	59827	0	0	0	0
18	131072	10749	120323	0	0	0	0
19	262144	20390	241754	0	0	0	0
20	524288	38635	485653	0	0	0	0
21	1048576	73586	974990	0	0	0	0

Table 13: Average running time of AKS Conjecture and simple priamlity testing algorithm

From above we can see that the algorithm based on AKS Conjecture is extremely fast for not much larger input. Actually it keeps efficient for larger integers.

Chapter 8

Conclusions

It was the objectives of this dissertation to implement the AKS Algorithm and analysis its running results. The system we have developed meets the essential demands of the requirements specification in chapter 4. Along with the initial objectives, we had also looked into some other algorithms related to the AKS Algorithm, and we had made some developments on them as well. We called these algorithms 'AKS class' algorithm. To conclude this dissertation, we should go through the project development process and to see was every thing went well. As mentioned in the early of this chapter that the final system has met our initial requirements. The design was entirely based on the requirements. We had used various basic algorithms to implement the AKS algorithms and they are discussed in the chapter 6. After we implement these four algorithms, we first tested the system functionalities and compare the results with initial requirements. After we had moved on to measure the running time results of these four algorithms. The testing include perfect power frequency testing, composite frequency testing, test the *r* and *s* found, and so on. From those testing results we have come up with some conclusions. (1).The AKS Version 3 algorithm finds the suitable *r* first. AKS Algorithm improved by D. Bernstein also finds suitable *r* from small integers. Although, these two algorithms find suitable *r* early, they work slowly for small input integers. The original AKS algorithm is better for small input, but it really slow to test larger inputs. (2).The AKS Algorithm improved by D. Bernstein performs better than original AKS Algorithm and AKS Version 3 algorithm even under the worst case. We can also find out that the max value of *r* and *s* is still between $\tilde{O}\left(\log^2 n\right)$ and $\tilde{O}\left(\log^3 n\right)$, and is much smaller than $\tilde{O}\left(\log^6 n\right)$, which is given in the paper of original AKS algorithm. (3).From our

testing results it seems that the average complexity of original AKS Algorithm is $\tilde{O}\left(\log^{7.5} n\right)$. (4).The AKS Conjecture is the fastest algorithm to compute the really large integers and it efficient to use in practice.

Chapter 9

Future Works

From the testing chapter, we can realize that the original AKS Algorithm run very slowly on testing really large input integer. Even AKS Version 3 and AKS Algorithm improved by D. Bernstein, they are not good for practical uses. Nowadays, many improvements have been provided since the original AKS Algorithm was published. We could take some advantages of these improvements to implement a more efficient and practical primality testing algorithm. On the other hand, the algorithm based on the AKS Conjecture mentioned in the same article of the original AKS Algorithm [1] runs extremely fast according to the testing results. But this conjecture has not been proved up to today. If some one could prove it in the future, then we will get an efficient and practical algorithm for primality testing, which can be used in modern cryptography field and other filed which required for prime numbers.

References

[1]. M. Agrawa, N. Kayal, and N. Saxena. “Primes is in P”, Preprint, IIT Kanpur, August 2002. (http://www.cse.iitk.ac.in/news/primality.pdf).

[2]. M. Agrawa, N. Kayal, and N. Saxena. “Primes is in P”, revised ‘v3’ paper. (http://www.cse.iitk.ac.in/news/primality_v3.pdf).

[3]. A. Klappenecker. An introduction to the AKS primality test. September 2002.

[4]. Bernstein D 2003a, “Proving primality after Agrawal-Kayal-Saxena,” preprint 25 Jan. http://cr.yp.to/papers.html

[5]. Bernstein D 2003b, “Proving Primality in Essentially Quartic Expected Time,” preprint 28 Jan. http://cr.yp.to/papers.html

[6]. Berrizbeitia P 2002, “Sharpening ‘Primes is in P’ for a large family of numbers,” preprint 20 Nov.

[7]. Cheng Q 2003, “Primality proving via one round of ECPP and one iteration in AKS,” http://www.cs.ou.edu/~qcheng/

[8]. Carndall R and Fagin B 1994, “Discrete Weighted Transforms and Large-Integer Arithmetic,” *Math. Comp.* 62, 305-324.

[9]. Lenstra H W Jr, “Primality testing with cyclotomic tings,” preprint, 14 Aug.

[10]. Lenstra H W Jr and Pomesance C, “Primality testing with Gaussian periods,” manuscript Mar 2003

[11]. Percival C 2003, “Papid multiplication modulo the sum and difference of highly composite numbers,” *Math. Comp*. 72, 387-395.

[12]. Pomerance C 2002, “The cyclotomic ring test of Agrawal, Kayal, and Saxena.” preprint.

[13]. Joachim von zur Gathen and Jurgen Gerhard. Modern Computer Algebra. Cambridge University Press, 1999.

[14]. R. Crandall and C. Pomerance, “Prime numbers: A computational perspective”. Springer, 2001.

References

[15]. D. Knuth, Art of computer programming, VII. Addison-Wesley, 1998.

[16]. NTL: A Library for doing Number Theory. http://shoup.net/

[17]. Victor Shoup. A Computational Introduction to Number Theory and Algebra (BETA version 5). Copyright © 2004.

[18]. R. C. Baker and G. Harman. The Brun-Titchmarsh Theorem on average. In proceeding of conference in Honour of Heini Halberstam, Volume 1, pages 39-103, 1996.

[19]. G. L. Miller. Riemann's hypothesis and test for primality. J. Comput. Sys. Sci, 13:300-317, 1976.

[20]. M. O. Rabin. Probabilistic algorithm for testing primality. J. Number Theory, 12:128-138. 1980.

[21]. R. Solovay and V. Stressen. A fast Monte-Carlo test for primality. SIAM Journal on Computing, 6:84-86, 1977.

[22]. H. Williams, Edouard Lucas and Primality Testing, CMS Monographs, Wiley, 1998.

[23]. Number theory. http://en.wikipedia.org/wiki/Number_theory

[24]. The AKS "PRIMES in P" Algorithm Resource. http://www.dcc.unicamp.br/~rdahab/cursos/mo637-mc933/Welcome_files/numTheory/WebPages/The%20AKS%20%22PRIMES%20in%20P%22%20Algorithm%20Resource.html

[25]. Caldwell's Prime Pages. Finding primes & proving primality. 4.3: A Polynomial-Time Algorithm. http://www.utm.edu/research/primes/prove/prove4_3.html

[26]. Prashant Pandey. Rajat Bhattacharjee. Primality Testing, April, 2001. http://www.cse.iitk.ac.in/research/btp2001/primality.html

[27]. Nell Dale, Chip Weems. Programming and problem solving with C++ 4th edition, published 2004.

[28]. Herbert Schildt. C++ from the Ground Up. 1st edition, published 1994.

[29]. Kris Jamsa's. Starting with Microsoft Visual C++ 1st edition, © 2001 by Prima Publishing.

References

[30]. Tom Cargill. C++ Programming Style, © 1992 by Addison-Wesley Publishing Company, Inc.

[31]. Ladd, Scott. C++ techniques and applications 1st edition, © 1990 by M&T Publishing, Inc.

[32]. Online C++ tutorial http://www.intap.net/~drw/cpp/

[33]. The cplusplus.com tutorial http://www.cplusplus.com/doc/tutorial/

Appendices

A: NTL Instruction

A-1: Installing on Windows or other platforms

Because NTL is portable C++ library, so it has not yet installed in any machine. Therefore, before using the NTL Library you must download it, then install and configure it. Below listing the procedures of initializing the NTL:

Step 1: Download & Unpack

You should download the NTL from its official website at: http://www.shoup.net/ntl/download.html . In that website, it shows various versions of NTL and for porting at different environments. You can place the NTL in a specific directory as you wish, and then working in this directory. After you have placed the NTL in the directory, then you can unzip the folder into current director. This will unpack everything into current directory and called "WinNTL-version No.". Inside this directory, you will find several directories (folders).

Step 2: Configuration flags.

In directory "`include/NTL`" is a file called "`config.h`". Edit this file to override some of NTL's default options for *basic configuration* and *performance*. Below are some options which we can choose when configuring our NTL.

Basic configuration options.

- Most of the these flags are rather esoteric and can be safely ignored.
- One exception to this is the `NTL_STD_CXX` flag (or perhaps just the `NTL_PSTD_NNS` flag) which you will want to set if you want to take advantage of the new *namespace* feature of `C++`. Go here for details.
- Another exception is the flags to use GMP for potentially faster long integer arithmetic.

Performance options.

These flags let you fine tune for best performance. It can let the *configuration wizard* set them for you.

TIP for Pentium platforms:

- Users running on a Pentium, or other x86-like processor, will almost surely want to set the `NTL_LONG_LONG` flag, or possibly the `NTL_AVOID_FLOAT` flag, in file `config.h` to get the best performance for long integer arithmetic. If you set either of these flags, you should also set the `NTL_TBL_REM` flag as well, to get the best performance for `ZZ_pX` arithmetic.
- These flags can be useful on other platforms as well, especially on processors with slow int/float conversion.
- The best thing is to experiment, and compile and run program `Test` to see the impact on the running time of various basic operations.

Note that the file "`def_config.h`" contains a backup copy of the original `config.h` file.

Step 3: Compiling NTL in Microsoft Visual C++ 6.0

Because we use Microsoft Visual C++ 6.0 on Windows to develop over system, so here are some steps for compiling and using NTL. These steps work with Microsoft Visual C++ 6.0. The following steps use to build the library and to build and run a test program to insure the NTL is fully installed the program called `test`:

```
File -> New -> Projects
   project name: ntl
   location[default]: c:\Program Files\Microsoft Visual
Studio\MyProjects\ntl
   Click on Win32 static library
   Click on OK
   pre-compiled headers[default]: no
   MFC support[default]: no
   Click on Finish
   Click on OK

Project -> Add to Project -> Files
   select all files in c:\mystuff\WinNTL-5_4\src and click on OK.

Project -> Settings -> C/C++
   Category: Preprocessor.
   Additional include directories: c:\mystuff\WinNTL-5_4\include.
   Click on OK.

Build -> build ntl.lib

File -> New -> Projects -> Win32 Console Application
   project name: test
   location[default]: c:\Program Files\Microsoft Visual
Studio\MyProjects\ntl
   Click on Win32 Console Application
   Click on OK
   What kind of windows application...? [default]: An empty project
   Click on Finish
   Click on OK
```

```
Project -> Add to Project -> Files
   select the file c:\mystuff\WinNTL-5_4\tests\Test.cpp
   Click on OK

Project -> Add to Project -> Files
   select the file
      c:\Program Files\Microsoft Visual
Studio\MyProjects\ntl\Debug\ntl.lib
   Note: one must select Files of type: Library Files (.lib) to make
this
      file visible in the pop-up window.
   Click on OK

Project -> Settings -> C/C++
   Category: Preprocessor.
   Additional include directories: c:\mystuff\WinNTL-5_4\include.
   Click on OK.

Build -> build test.exe

Build -> execute test.exe
```

A-2: Installing on UNIX

Step 1: Download

You should download the NTL from its official website at: http://www.shoup.net/ntl/download.html . In that website, it shows various versions of NTL and for porting at different environments. You can place the NTL in a specific directory as you wish.

Step 2: Unpack

The xxx denotes the version of NTL you have downloaded from the website. Use the following command to unpack the NTL package.

```
% gunzip ntl-xxx.tar.gz
% tar xvf ntl-xxx.tar
```

Note that this will unpack everything into a sub-directory `ntl-xxx`, creating this directory if necessary. Next:

```
% cd ntl-xxx
% ls
```

You should see a file "`README`", and directories "`include`", "`doc`", and "`src`". The directory "`doc`" contains all the documentation. The file "`doc/tour.html`" contains a copy of the on-line documentation. The directory "`include`" contains all the header files

within a subdirectory "include/NTL". The directory "src" contains everything else. Go there now:

```
% cd src
```

Step 3: Run & Configure

Execute the command

```
% ./configure [ variable=value ]...
```

This configure script generates the file "makefile" and the file "../include/NTL/config.h", based upon the values assigned to the variables on the command line.

Here are the most important variables, and their default values.

```
   CC=gcc               # The C compiler
   CXX=g++              # The C++ compiler
   CFLAGS=-O2           # C complilation flags
   CXXFLAGS=$(CFLAGS)   # C++ compilation flags (by default, same as
CFLAGS)

   PREFIX=/usr/local    # Directory in which to install NTL library
components

   NTL_STD_CXX=off      # ISO Mode switch

   NTL_GMP_LIP=off      # Switch 'on' to enable the use of GMP as the
primary
                        # long integer package
   NTL_GMP_HACK=off     # Switch 'on' to enable the use of GMP as a
supplemental
                        # long integer package

   GMP_PREFIX=none      # Directory in which GMP components have been
installed
```

Examples

- If you are happy with all the default values, run:

  ```
  % ./configure
  ```

  Actually, the initially installed makefile and config.h files already reflect the default values, and you do not have to even run the configure script.

- If your C/C++ compilers are called cc/CC, run:

  ```
  % ./configure CC=cc CXX=CC
  ```

Step 4: Execute *make*

Just type:

```
% make
```

Step 5: Testing
Executing `make check` runs a series of timing and test programs. It is a good idea to run this to see if everything really went well.

A-3: Installing on Dev C++

The only different from NTL in stalling on Windows is we need to compile and configure the NTL on the Dev C++ platform, below listed the steps of how to stepup NTL on Dev C++
Step 1: Download the Dev C++ 4.9.9.2 on the website.
Step 2: Create a new project called libntl (choose static library). And save in a specific directory.
Step 3: click Tools→Compiler Options→select Directories → select C++ Includes →browse your downloaded WinNTL-5_4\include→click add→click ok.
Step 4: compile→copy the libntl.a file to the Dev-Cpp\lib folder
Step 5: click Tools→ Compiler Options→tick 'Add these commands to the linker command line→type in '-lntl'.

B: Programs

B-1: Program listing

We summarize of what you will find in each of the files that make up AKS1 application.

Primality.cpp

This file is the main source of this program; it contains four algorithms and other basic algorithm.

Primality.h

This the header file which need together with Primality.cpp

AKS1.dsp

This file (the project file) contains information at the project level and is used to build a single project or subproject. Other users can share the project (.dsp) file, but they should export the makefiles locally.

AKS1.h

This is the main header file for the application. It includes other project specific headers (including Resource.h) and declares the CAKS1App application class.

AKS1.cpp

This is the main application source file that contains the application class CAKS1App.

AKS1.rc

This is a listing of all of the Microsoft Windows resources that the program uses. It includes the icons, bitmaps, and cursors that are stored in the RES subdirectory. This file can be directly edited in Microsoft Visual C++.

AKS1.clw

This file contains information used by ClassWizard to edit existing classes or add new classes. ClassWizard also uses this file to store information needed to create and edit message maps and dialog data maps and to create prototype member functions.

res\AKS1.ico

This is an icon file, which is used as the application's icon. This icon is included by the main resource file AKS1.rc.

res\AKS1.rc2

This file contains resources that are not edited by Microsoft Visual C++. You should place all resources not editable by the resource editor in this file.

AKS1Dlg.h, AKS1Dlg.cpp - the dialog

These files contain your CAKS1Dlg class. This class defines the behavior of your application’s main dialog. The dialog's template is in AKS1.rc, which can be edited in Microsoft Visual C++.

StdAfx.h, StdAfx.cpp

These files are used to build a precompiled header (PCH) file named AKS1.pch and a precompiled types file named StdAfx.obj.

Resource.h

This is the standard header file, which defines new resource IDs. Microsoft Visual C++ reads and updates this file.

B-2: Programs Code

Primality.cpp

```
/*/////////contains all four algorithms and other basic algorithm used//////////
*
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "Primality.h"

//////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1
bool IsPrime(U32 n)
{
   if(n==2)return true;
   if (n % 2 == 0) return false;
   U32 root = (U32)sqrt(n);
            U32 i;
            for (i = 3; i <=root; i += 2)if (n % i == 0)return false;
            return true;
}

//////////////////////////////////////////////////////////////////////////////
//Compute greatest common divisor of a and b
U32 Gcd(U32 a,U32 b)
{
   U32 m=a,n=b;
   while(n!=0)
   {
      m%=n;
      m^=n;n^=m;m^=n; //swap(m,n)
   }
   return m;
};

//////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e
void IntRoot(ZZ &root,const ZZ &n,U32 e)
{
   ZZ y;
   double ll=NumBits(n)*1.0/e;
   power2(root,ceil(ll));
   while(true)
   {
     y=(e-1)*root+n/power(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}

//////////////////////////////////////////////////////////////////////////////
//Test whether n is a perfect power: true:false?n=a^b
bool IsPower(const ZZ &n)
{
   ZZ rt;
   U32 i,le=NumBits(n);
   for(i=2;i<le;i++)
   {
      IntRoot(rt,n,i);
      if(power(rt,i)==n)return true;
   }
   return false;
```

```
}

////////////////////////////////////////////////////////////////////////////
//Return largest prime factor of n
U32 LargestPrimeFactor(U32 n)
{
	if (n < 2) return 1;

	U32 r = n, p;
	if (r % 2 == 0)
	{
		p = 2;
		do { r /= 2; } while (r % 2 == 0);
	}
	U32 i;
	for (i = 3; i <= r; i += 2)
	{
		if (r % i == 0)
		{
			p = i;
			do { r /= i; } while (r % i == 0);
		}
	}
	return p;
}

////////////////////////////////////////////////////////////////////////////
//return (n^e)%m
U32 PowerOverM(U32 n, U32 e, U32 m)
{
	U64 r = 1;
	U64 t = n % m;
	U32 i;
	for (i = e; i != 0; i /= 2)
	{
		if (i % 2 != 0)
		{
			r=(r*t)%m;
		}
		t=(t*t)%m;
	}
	return r;
}

////////////////////////////////////////////////////////////////////////////
//Order: the least k such that n^k=1 (mod r),assuming (n,r)=1
U32 Order(U32 n,U32 r)
{
   U32 o=1;
   U64 m=n;
   for(;;)
   {
      if(m%r==1)return o;
      m*=n;m%=r;o++;
   }

}

////////////////////////////////////////////////////////////////////////////
//Euler function phi(x):number of integer smaller than  x and coprime to x
U32 EulerPhi(U32 x)
{
   U32 ct=1,i;
   for(i=2;i<x;i++)
   {
      if(Gcd(x,i)==1)ct++;
   }
   return ct;
}
```

```
////////////////////////////////////////////////////////////////////////////
//AKS Algorithm original by M. Agrawal, N. Kayal and N. Saxena
bool AKS(const ZZ &n,U32 &v,U32 &c,U32 &t)
{
        U32 t1,t2;
        t1=clock();
  if(n==1){t2=clock();t=t2-t1;return false;}
  if(n<4){t2=clock();t=t2-t1;return false;}
  if(IsPower(n)){t2=clock();t=t2-t1;return false;}
  U32 r=3,q,a,s;
  while(r<n)
  {
    if(Gcd(n%r,r)!=1){t2=clock();t=t2-t1;return false;}
    if(IsPrime(r))
    {
      q=LargestPrimeFactor(r-1);
      if(q>=(4*sqrt(r)*log(n)/log(2)))if(PowerOverM(n%r,(r-1)/q,r)!=1)break;

    }
    r++;
  }
  s=2*sqrt(r)*log(n)/log(2);v=r;c=s;
  if(r>=n){c=0;t2=clock();t=t2-t1;return true;}
  ZZ_p::init(n);//mod n
        ZZ_pX f(r, 1); f -= 1;//f=x^r-1
        const ZZ_pXModulus pf(f);

        ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
        for(a=1;a<=s;a++)
  {
    ZZ_pX lhs(1,1);lhs-=a;//x-a
    PowerMod(lhs,lhs,n,pf);//(x-a)^n
    lhs+=a;//(x-a)^n+a
    if(lhs!=rhs){t2=clock();t=t2-t1;return false;}
  }
        t2=clock();
        t=t2-t1;
  return true;
}

////////////////////////////////////////////////////////////////////////////
//Algorithm Conjecture algorithm
bool AKSC4(const ZZ &n,U32 &v,U32 &t)
{
        U32 t1,t2;
        t1=clock();
  U32 r;
  U64 m;
  for(r=2;;r++)
  {
    m=n%r;
    if(m==0){t2=clock();t=t2-t1;return false;}
    if((m*m%r)!=1)break;
  }
  ZZ_p::init(n);
  ZZ_pX f(r,1);f-=1;//f=x^r-1
  const ZZ_pXModulus pf(f);

  ZZ_pX rhs(m,1);rhs-=1;//x^m-1

  ZZ_pX lhs(1,1);lhs-=1;//x-1
  PowerMod(lhs,lhs,n,pf);//(x-1)^n
  if(lhs==rhs){v=r;t2=clock();t=t2-t1;return true;}
  else {t2=clock();t=t2-t1;return false;}
}

////////////////////////////////////////////////////////////////////////////
//AKS version 3 by M. Agrawal, N. Kayal and N. Saxena
bool AKSV3(ZZ &n,U32 &v,U32 &c,U32 &t)
{
```

```
            U32 t1,t2;
            t1=clock();
  if(n==1){t2=clock();t=t2-t1;return false;}
  if(n<4){t2=clock();t=t2-t1;return false;}
  if(IsPower(n)){t2=clock();t=t2-t1;return false;}
  U32 r=3,q,a,s;
  double m,l=log(n)/log(2); m=l*l*4;
  while(r<n)
  {
    if(Gcd(n%r,r)!=1){t2=clock();t=t2-t1;return false;}
    if(Order(n%r,r)>m)break;
    r++;
  }
  s=floor(2*sqrt(EulerPhi(r))*l);v=r;c=s;
  if(n<=r){c=0;t2=clock();t=t2-t1;return true;}

  ZZ_p::init(n);//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
  for(a=1;a<=s;a++)
  {
    ZZ_pX lhs(1,1);lhs+=a;//x+a
    PowerMod(lhs,lhs,n,pf);//(x+a)^n
    lhs-=a;//(x+a)^n-a
    if(lhs!=rhs){t2=clock();t=t2-t1;return false;}

  }
            t2=clock();
            t=t2-t1;
  return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS algorithm improved by D. Bernstein
bool AKSDB(ZZ &n,U32 &v,U32 &c,U32 &t)
{
            U32 t1,t2;
            t1=clock();
  if(n==1){t2=clock();t=t2-t1;return false;}
  if(n<4){t2=clock();t=t2-t1;return false;}
  if(IsPower(n)){t2=clock();t=t2-t1;return false;}
  U32 r=3,q,a,s;
  while(r<n)
  {
    if(Gcd(n%r,r)!=1){t2=clock();t=t2-t1;return false;}
    if(IsPrime(r))
    {
      q=LargestPrimeFactor(r-1);
      if(PowerOverM(n%r,(r-1)/q,r)==1){r++;continue;}
      double cm=2*floor(sqrt(r))*log(n),c=0;
      for(s=1;s<q && c<cm;s++) c+=log(q+s-1)-log(s);
      if(c>=cm)break;

    }
    r++;
  }
  v=r;c=s;
  if(n<=r){c=0;t2=clock();t=t2-t1;return true;}
  ZZ_p::init(n);//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
  {
    ZZ_pX lhs(1,1);lhs-=a;//x-a
    PowerMod(lhs,lhs,n,pf);//(x-a)^n
    lhs+=a;//(x-a)^n+a
```

```
      if(lhs!=rhs){t2=clock();t=t2-t1;return false;}
   }
            t2=clock();
            t=t2-t1;
   return true;
}
```

Primality.h

```
//////////////////////////////////////////////////////////////
//
// this header file linked with Primality.cpp
// and define some functions used there
//////////////////////////////////////////////////////////////


#include "stdafx.h"
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>

#define NTL_NO_MIN_MAX
NTL_CLIENT

typedef unsigned __int32 U32;
typedef unsigned __int64 U64;

bool IsPrime(U32 n);
U32 Gcd(U32 a,U32 b);
void IntRoot(ZZ &root,const ZZ &n,U32 e);
bool IsPower(const ZZ &n);
U32 LargestPrimeFactor(U32 n);
U32 PowerOverM(U32 n, U32 e, U32 m) ;
U32 Order(U32 n,U32 r);
U32 EulerPhi(U32 x);
bool AKS(const ZZ &n,U32 &v,U32 &c,U32 &t);
bool AKSC4(const ZZ &n,U32 &v,U32 &t);
bool AKSV3(ZZ &n,U32 &v,U32 &c,U32 &t);
bool AKSDB(ZZ &n,U32 &v,U32 &c,U32 &t);
```

AKS1.cpp

```
// AKS1.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "AKS1.h"
#include "AKS1Dlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAKS1App

BEGIN_MESSAGE_MAP(CAKS1App, CWinApp)
            //{{AFX_MSG_MAP(CAKS1App)
                        // NOTE - the ClassWizard will add and remove mapping macros here.
                        //    DO NOT EDIT what you see in these blocks of generated code!
            //}}AFX_MSG
            ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
```

```
// CAKS1App construction

CAKS1App::CAKS1App()
{
	// TODO: add construction code here,
	// Place all significant initialization in InitInstance
}

/////////////////////////////////////////////////////////////////////////////
// The one and only CAKS1App object

CAKS1App theApp;

/////////////////////////////////////////////////////////////////////////////
// CAKS1App initialization

BOOL CAKS1App::InitInstance()
{
	// Standard initialization
	// If you are not using these features and wish to reduce the size
	//  of your final executable, you should remove from the following
	//  the specific initialization routines you do not need.

#ifdef _AFXDLL
	Enable3dControls();			// Call this when using MFC in a shared DLL
#else
	Enable3dControlsStatic();	// Call this when linking to MFC statically
#endif

	CAKS1Dlg dlg;
	m_pMainWnd = &dlg;
	int nResponse = dlg.DoModal();
	if (nResponse == IDOK)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with OK
	}
	else if (nResponse == IDCANCEL)
	{
		// TODO: Place code here to handle when the dialog is
		//  dismissed with Cancel
	}

	// Since the dialog has been closed, return FALSE so that we exit the
	//  application, rather than start the application's message pump.
	return FALSE;
}
```

AKS1.h

```
// AKS1.h : main header file for the AKS1 application
//

#if !defined(AFX_AKS1_H__A4D6CFE4_FAF6_4A17_8F4E_A8365C6F7362__INCLUDED_)
#define AFX_AKS1_H__A4D6CFE4_FAF6_4A17_8F4E_A8365C6F7362__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
	#error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"		// main symbols

/////////////////////////////////////////////////////////////////////////////
```

```
// CAKS1App:
// See AKS1.cpp for the implementation of this class
//

class CAKS1App : public CWinApp
{
public:
	CAKS1App();

// Overrides
	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAKS1App)
	public:
	virtual BOOL InitInstance();
	//}}AFX_VIRTUAL

// Implementation

	//{{AFX_MSG(CAKS1App)
		// NOTE - the ClassWizard will add and remove member functions here.
		//    DO NOT EDIT what you see in these blocks of generated code !
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};


/////////////////////////////////////////////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AKS1_H__A4D6CFE4_FAF6_4A17_8F4E_A8365C6F7362__INCLUDED_)
```

AKS1Dlg.cpp

```
// AKS1Dlg.cpp : implementation file
//

#include "stdafx.h"
#include "AKS1.h"
#include "AKS1Dlg.h"

#include "Primality.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

/////////////////////////////////////////////////////////////////////////////
// CAKS1Dlg dialog

CAKS1Dlg::CAKS1Dlg(CWnd* pParent /*=NULL*/)
	: CDialog(CAKS1Dlg::IDD, pParent)
{
	//{{AFX_DATA_INIT(CAKS1Dlg)
	m_input = _T("");
	m_output = _T("");
	//}}AFX_DATA_INIT
	// Note that LoadIcon does not require a subsequent DestroyIcon in Win32
	m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CAKS1Dlg::DoDataExchange(CDataExchange* pDX)
{
	CDialog::DoDataExchange(pDX);
	//{{AFX_DATA_MAP(CAKS1Dlg)
```

```
	DDX_Text(pDX, IDC_Input, m_input);
	DDX_Text(pDX, IDC_output, m_output);
	//}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAKS1Dlg, CDialog)
	//{{AFX_MSG_MAP(CAKS1Dlg)
	ON_WM_PAINT()
	ON_WM_QUERYDRAGICON()
	ON_BN_CLICKED(IDC_AKS, OnAks)
	ON_BN_CLICKED(IDC_AKSV3, OnAksv3)
	ON_BN_CLICKED(IDC_AKSDB, OnAksdb)
	ON_BN_CLICKED(IDC_AKSC4, OnAksc4)
	ON_BN_CLICKED(IDC_Clear, OnClear)
	//}}AFX_MSG_MAP
END_MESSAGE_MAP()

/////////////////////////////////////////////////////////////////////////////
// CAKS1Dlg message handlers

BOOL CAKS1Dlg::OnInitDialog()
{
	CDialog::OnInitDialog();

	// Set the icon for this dialog.  The framework does this automatically
	//  when the application's main window is not a dialog
	SetIcon(m_hIcon, TRUE);			// Set big icon
	SetIcon(m_hIcon, FALSE);		// Set small icon

	// TODO: Add extra initialization here

	return TRUE;  // return TRUE  unless you set the focus to a control
}

// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.

void CAKS1Dlg::OnPaint()
{
	if (IsIconic())
	{
		CPaintDC dc(this); // device context for painting

		SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

		// Center icon in client rectangle
		int cxIcon = GetSystemMetrics(SM_CXICON);
		int cyIcon = GetSystemMetrics(SM_CYICON);
		CRect rect;
		GetClientRect(&rect);
		int x = (rect.Width() - cxIcon + 1) / 2;
		int y = (rect.Height() - cyIcon + 1) / 2;

		// Draw the icon
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

// The system calls this to obtain the cursor to display while the user drags
//  the minimized window.
HCURSOR CAKS1Dlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}
```

```
void CAKS1Dlg::OnAks()
{
	// TODO: Add your control notification handler code here
	ZZ n=to_ZZ(0);
	U32 r,s,t;
	bool b;
	CString tmpoutput;

	UpdateData(1);
	for(int i=0;i<m_input.GetLength();i++)
	{
		n*=10;
		int v=(int)m_input[i]-(int)'0';
		if(v>9 || v<0)
		{
			MessageBox("Illegal char in the string");
			m_input.Empty();
			UpdateData(0);
			return;
		}
		n+=v;
	}

	b=AKS(n,r,s,t);
	tmpoutput.Empty();
	if(b)tmpoutput.Format(" is a prime!\r\nr=%d\ts=%d\tt=%d ms.\r\n",r,s,t);
	else tmpoutput.Format(" is a composite!\r\nt=%d ms.\r\n",t);
	m_output+=m_input;
	m_output+=tmpoutput;
	m_input.Empty();

	UpdateData(0);



}

void CAKS1Dlg::OnAksv3()
{
	// TODO: Add your control notification handler code here
	ZZ n=to_ZZ(0);
	U32 r,s,t;
	bool b;
	CString tmpoutput;

	UpdateData(1);
	for(int i=0;i<m_input.GetLength();i++)
	{
		n*=10;
		int v=(int)m_input[i]-(int)'0';
		if(v>9 || v<0)
		{
			MessageBox("Illegal char in the string");
			m_input.Empty();
			UpdateData(0);
			return;
		}
		n+=v;
	}

	b=AKSV3(n,r,s,t);
	tmpoutput.Empty();
	if(b)tmpoutput.Format(" is a prime!\r\nr=%d\ts=%d\tt=%d ms.\r\n",r,s,t);
	else tmpoutput.Format(" is a composite!\r\nt=%d ms.\r\n",t);
	m_output+=m_input;
	m_output+=tmpoutput;
	m_input.Empty();
```

```
	UpdateData(0);

}

void CAKS1Dlg::OnAksdb()
{
	// TODO: Add your control notification handler code here
	ZZ n=to_ZZ(0);
	U32 r,s,t;
	bool b;
	CString tmpoutput;

	UpdateData(1);
	for(int i=0;i<m_input.GetLength();i++)
	{
		n*=10;
		int v=(int)m_input[i]-(int)'0';
		if(v>9 || v<0)
		{
			MessageBox("Illegal char in the string");
			m_input.Empty();
			UpdateData(0);
			return;
		}
		n+=v;
	}

	b=AKSDB(n,r,s,t);
	tmpoutput.Empty();
	if(b)tmpoutput.Format(" is a prime!\r\nr=%d\ts=%d\tt=%d ms.\r\n",r,s,t);
	else tmpoutput.Format(" is a composite!\r\nt=%d ms.\r\n",t);
	m_output+=m_input;
	m_output+=tmpoutput;
	m_input.Empty();
	UpdateData(0);

}

void CAKS1Dlg::OnAksc4()
{
	// TODO: Add your control notification handler code here
	ZZ n=to_ZZ(0);
	U32 r,t;
	bool b;
	CString tmpoutput;

	UpdateData(1);
	for(int i=0;i<m_input.GetLength();i++)
	{
		n*=10;
		int v=(int)m_input[i]-(int)'0';
		if(v>9 || v<0)
		{
			MessageBox("Illegal char in the string");
			m_input.Empty();
			UpdateData(0);
			return;
		}
		n+=v;
	}

	b=AKSC4(n,r,t);
	tmpoutput.Empty();
	if(b)tmpoutput.Format(" is a prime!\r\nr=%d\tt=%d ms.\r\n",r,t);
	else tmpoutput.Format(" is a composite!\r\nt=%d ms.\r\n",t);
	m_output+=m_input;
	m_output+=tmpoutput;
	m_input.Empty();
	UpdateData(0);
```

```
}

void CAKS1Dlg::OnCancel()
{
	// TODO: Add extra cleanup here

	CDialog::OnCancel();
}

void CAKS1Dlg::OnClear()
{
	// TODO: Add your control notification handler code here
	m_input.Empty();
	UpdateData(0);

}
```

AKS1Dlg.h

```
// AKS1Dlg.h : header file
//

#if !defined(AFX_AKS1DLG_H__CEC9382A_D177_4184_89C1_E4476946B285__INCLUDED_)
#define AFX_AKS1DLG_H__CEC9382A_D177_4184_89C1_E4476946B285__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

/////////////////////////////////////////////////////////////////////////////
// CAKS1Dlg dialog

class CAKS1Dlg : public CDialog
{
// Construction
public:
	CAKS1Dlg(CWnd* pParent = NULL);	// standard constructor

// Dialog Data
	//{{AFX_DATA(CAKS1Dlg)
	enum { IDD = IDD_AKS1_DIALOG };
	CString	m_input;
	CString	m_output;
	//}}AFX_DATA

	// ClassWizard generated virtual function overrides
	//{{AFX_VIRTUAL(CAKS1Dlg)
	protected:
	virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
	//}}AFX_VIRTUAL

// Implementation
protected:
	HICON m_hIcon;

	// Generated message map functions
	//{{AFX_MSG(CAKS1Dlg)
	virtual BOOL OnInitDialog();
	afx_msg void OnPaint();
	afx_msg HCURSOR OnQueryDragIcon();
	afx_msg void OnAks();
	afx_msg void OnAksv3();
	afx_msg void OnAksdb();
	afx_msg void OnAksc4();
	virtual void OnCancel();
	afx_msg void OnClear();
	afx_msg void OnCancelMode();
```

```
	//}}AFX_MSG
	DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_AKS1DLG_H__CEC9382A_D177_4184_89C1_E4476946B285__INCLUDED_)
```

resource.h

```
//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by AKS1.rc
//
#define IDD_AKS1_DIALOG                 102
#define IDR_MAINFRAME                   128
#define IDC_AKSC4                       1000
#define IDC_AKSDB                       1001
#define IDC_AKSV3                       1002
#define IDC_AKS                         1003
#define IDC_Input                       1004
#define IDC_Clear                       1005
#define IDC_output                      1006
#define IDC_BUTTON1                     1007

// Next default values for new objects
//
#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE        129
#define _APS_NEXT_COMMAND_VALUE         32771
#define _APS_NEXT_CONTROL_VALUE         1008
#define _APS_NEXT_SYMED_VALUE           101
#endif
#endif
```

StdAfx.cpp

```
// stdafx.cpp : source file that includes just the standard includes
//	AKS1.pch will be the pre-compiled header
//	stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"
```

StdAfx.h

```
// stdafx.h : include file for standard system include files,
//  or project specific include files that are used frequently, but
//      are changed infrequently
//

#if !defined(AFX_STDAFX_H__1093192C_8B00_4422_AB56_F7931490D92B__INCLUDED_)
#define AFX_STDAFX_H__1093192C_8B00_4422_AB56_F7931490D92B__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN                    // Exclude rarely-used stuff from Windows headers

#include <afxwin.h>         // MFC core and standard components
#include <afxext.h>         // MFC extensions
#include <afxdtctl.h>               // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
```

```
#include <afxcmn.h>                         // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT


//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.

#endif // !defined(AFX_STDAFX_H__1093192C_8B00_4422_AB56_F7931490D92B__INCLUDED_)
```

C: Testing Programs

C-1: Program listing

PrimalityTesting.cpp

It tests the input and output of the four algorithms and also simple prime testing algorithms.

Conjecture.cpp

It uses AKS conjecture to test large integers.

CompositeFrequency.cpp

It uses to test the primes and composites decided in each step of AKS original, AKS version 3, and AKS improved by D. Bernstein.

PerfectPowerFrequency.cpp

It uses to test the fist step of all four algorithms, check number of Prime, Composite, Perfect Power, and Perfect Power with Prime and Composite root.

SimpleandConjecture.cpp

It uses to compare simple prime testing algorithm and AKS Conjecture algorithm.

AVESizeStatics.cpp

It uses to compute the average value of r and s of different size of input from all four algorithms.

Badcse.cpp

It uses to compute the maximum r and s of different size of input from all four algorithms

C-2: Programs Code

PrimalityTesting.cpp

```
/*//////////////general testing for input numbers in all four algorithms////////
*
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 primes

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};
```

```
////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   U32 root = (U32)sqrt(n+1);
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}

////////////////////////////////////////////////////////////////////////////
//Compute greatest common divisor of a and b

U32 Gcd(const U32 a,const U32 b)
{
   U32 m=a,n=b;
   if(m==0)return n;
   while(n!=0)
   {
      m%=n;
      m^=n;n^=m;m^=n; //swap(m,n)
   }
   return m;
};

////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(U32 &root,const U32 &n,U32 e)
{
   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}

   U32 y;
   U32 pp=ceil(bits*1.0/e);
   root=1;root<<=pp;
   while(true)
   {
     y=(e-1)*root+n/power_long(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}
////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e
void IntRoot(ZZ &root,const ZZ &n,U32 e)
{
   ZZ y;
   double ll=NumBits(n)*1.0/e;
   power2(root,ceil(ll));
   while(true)
   {
     y=(e-1)*root+n/power(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}
////////////////////////////////////////////////////////////////////////////
//Test whether n is a perfect power: true:false?n=a^b

bool IsPower(const ZZ &n)
{
   ZZ rt;
   U32 i,le=NumBits(n);
   for(i=2;i<le;i++)
   {
```

```
      IntRoot(rt,n,i);
      if(power(rt,i)==n)return true;
   }
   return false;
}
bool IsPower(const U32 &n)
{
   U32 rt;

   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}

   for(U32 i=2;i<bits;i++)
   {
      IntRoot(rt,n,i);
      if((rt+n)%2)continue;
      if(power_long(rt,i)==n)return true;
   }
   return false;
}
////////////////////////////////////////////////////////////////////////////
//Return largest prime factor of n

U32 LargestPrimeFactor(U32 n)
{
        if (n < 2) return 1;

        U32 r = n, p;
        if (r % 2 == 0)
        {
                p = 2;
                do { r /= 2; } while (r % 2 == 0);
        }
        U32 i;
        for (i = 3; i <= r; i += 2)
        {
                if (r % i == 0)
                {
                        p = i;
                        do { r /= i; } while (r % i == 0);
                }
        }
        return p;
}

////////////////////////////////////////////////////////////////////////////
//return (n^e)%m

U32 PowerOverM(U32 n, U32 e, U32 m)
{
        U64 r = 1;
        U64 t = n % m;
        U32 i;
        for (i = e; i != 0; i /= 2)
        {
                if (i % 2 != 0)
                {
                        r=(r*t)%m;
                }
                t=(t*t)%m;
        }
        return r;
}

////////////////////////////////////////////////////////////////////////////
//Order: the least k such that n^k=1 (mod r),assuming (n,r)=1

U32 Order(U32 n,U32 r)
{
```

```
   U32 o=1;
   U64 m=n;
   for(;;)
   {
      if(m%r==1)return o;
      m*=n;m%=r;o++;
   }

}

////////////////////////////////////////////////////////////////////////////
//Euler function phi(x):number of integer smaller than  x and coprime to x

U32 EulerPhi(U32 x)
{
   U32 ct=1,i;
   for(i=2;i<x;i++)
   {
      if(Gcd(x,i)==1)ct++;
   }
   return ct;
}

////////////////////////////////////////////////////////////////////////////
//original AKS Algorithm by M. Agrawal, N. Kayal and N. Saxena

bool AKS(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {
         q=LargestPrimeFactor(r-1);
         if(q>=(4*sqrt(r)*log(n)/log(2)))if(PowerOverM(n%r,(r-1)/q,r)!=1)break;

      }
      r++;
   }
   s=2*sqrt(r)*log(n)/log(2);v=r;c=s;
   if(r>=n){c=0;return true;}
   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
      if(lhs!=rhs)return false;
   }
   return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS Conjecture algorithm

bool AKSC4(const ZZ &n,U32 &v)
{
   U32 r;
   U64 m;
   if(n<6 && n!=4){v=0;return true;}
   for(r=2;;r++)
```

```
   {
      m=n%r;
      if(m==0){v=0;return false;}
      //if(!IsPrime(r))continue;//not mentioned in [AKS02]
      if((m*m%r)!=1)break;
   }
   ZZ_p::init(n);
   ZZ_pX f(r,1);f-=1;//f=x^r-1
   const ZZ_pXModulus pf(f);

   ZZ_pX rhs(m,1);rhs-=1;//x^m-1

   ZZ_pX lhs(1,1);lhs-=1;//x-1
   PowerMod(lhs,lhs,n,pf);//(x-1)^n
   if(lhs==rhs){v=r;return true;}
   else return false;
}

////////////////////////////////////////////////////////////////////////////
//AKS Version 3 by M. Agrawal, N. Kayal and N. Saxena

bool AKSV3(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   double m,l=log(n)/log(2); m=l*l*4;
   while(r<n)
   {
     if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
     if(Order(n%r,r)>m)break;
     r++;
   }
   s=floor(2*sqrt(EulerPhi(r))*l);v=r;c=s;
   if(n<=r){c=0;return true;}

   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
   for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs+=a;//x+a
      PowerMod(lhs,lhs,n,pf);//(x+a)^n
      lhs-=a;//(x+a)^n-a
      if(lhs!=rhs)return false;

   }
   return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS algorithm improved by D. Bernstein

bool AKSDB(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {
```

```
            q=LargestPrimeFactor(r-1);
            if(PowerOverM(n%r,(r-1)/q,r)==1){r++;continue;}
            double cm=2*floor(sqrt(r))*log(n),c=0;
            for(s=1;s<q && c<cm;s++) c+=log(q+s-1)-log(s);
            if(c>=cm)break;

        }
        r++;
    }
    v=r;c=s;
    if(n<=r){c=0;return true;}
    ZZ_p::init(to_ZZ(n));//mod n
                ZZ_pX f(r, 1); f -= 1;//f=x^r-1
                const ZZ_pXModulus pf(f);

                ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
                for(a=1;a<=s;a++)
    {
        ZZ_pX lhs(1,1);lhs-=a;//x-a
        PowerMod(lhs,lhs,n,pf);//(x-a)^n
        lhs+=a;//(x-a)^n+a
        if(lhs!=rhs)return false;
    }
    return true;
}

////////////////////////////////////////////////////////////////////////////
int main()
{
    U32 n,r,s;ZZ bign;
    int method,log,num2test;

    do{
    cout<<"Please choose which method you\'d like to use:"<<endl;
    cout<<"0)\tSimple Prime testing-M0\n"<<"1)\tAKS original-M1\n"<<"2)\tAKS version 3-M2\n"<<"3)\tAKS improved by D.
Bernstein-M3\n"<<"4)\tAKS Conjecture-M4\n"<<"5)\tAll above\n";
    cin>>method;}while((method<0) || (method >5));
    cout<<"You have chosen "<<method<<endl<<endl;

    do{
    cout<<"Log the result?\n";
    cout<<"0)\tPrint result on screen\n"<<"1)\tLog it to \"Prime.txt\" in current directory\n";
    cin>>log;}while((log<0) || (log >1));
    cout<<"You have chosen "<<log<<endl<<endl;

    do{
    cout<<"Please decide how many numbers to be tested:\n";
    cout<<"0)\tfrom 2 to n: (when n large, it will take a long time)\n";
    cout<<"1)\tonly test n: (n>=2)\n";
    cin>>num2test;}while((num2test<0) || (num2test >1));
    cout<<"You have chosen "<<num2test<<endl<<endl;

    while(true){
        cout<<"Please input n:\n";
        cin>>bign;
        if(method!=4 && bign>100000000L)
        {
            cout<<"n is too large!\n";
            continue;
        }
        if(bign<2)
        {
            cout<<"n is too small\n";
            continue;
        }
        break;
    }


    U32 stime,etime;bool isprime;
```

```
std::ofstream fout("PrimalityTesting.txt");
fout<<"#Table of your testing results.\n\n";
fout<<"Numbers\tPrime?\tMethod\tBits\ttime\tr\ts\n";

ZZ  start=num2test>0?bign:to_ZZ(2);

for(ZZ ii=start;ii<=bign;ii++)
{
   switch(method)
   {
      case 5:
      case 0:
         {

            U32 tt=to_long(ii);
            U32 nn=tt;
            char rs='C';
            int bits=0;
            while(nn!=0){bits++;nn/=2;}
            stime=clock();
            isprime=IsPrime(tt);
            etime=clock();
            if(isprime)rs='P';
            if(log==0)cout<<tt<<"\t"<<rs<<"\tM0\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<endl;
            else fout<<tt<<"\t"<<rs<<"\tM0\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<endl;
            if(method==0)break;
         }
      case 1:
         {
            char rs='C';
            U32 tt=to_long(ii);
            U32 nn=tt,r=0,s=0;
            int bits=0;
            while(nn!=0){bits++;nn/=2;}
            stime=clock();
            isprime=AKS(tt,r,s);
            etime=clock();
            if(isprime)rs='P';
            if(log==0)cout<<tt<<"\t"<<rs<<"\tM1\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
            else fout<<tt<<"\t"<<rs<<"\tM1\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
            if(method==1)break;
         }
      case 2:
         {
            char rs='C';
            U32 tt=to_long(ii);
            U32 nn=tt,r=0,s=0;
            int bits=0;
            while(nn!=0){bits++;nn/=2;}
            stime=clock();
            isprime=AKSV3(tt,r,s);
            etime=clock();
            if(isprime)rs='P';
            if(log==0)cout<<tt<<"\t"<<rs<<"\tM2\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
            else fout<<tt<<"\t"<<rs<<"\tM2\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
            if(method==2)break;
         }
      case 3:
         {
            char rs='C';
            U32 tt=to_long(ii);
            U32 nn=tt,r=0,s=0;
            int bits=0;
            while(nn!=0){bits++;nn/=2;}
            stime=clock();
            isprime=AKSDB(tt,r,s);
            etime=clock();
            if(isprime)rs='P';
            if(log==0)cout<<tt<<"\t"<<rs<<"\tM3\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
            else fout<<tt<<"\t"<<rs<<"\tM3\t"<<bits<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<"\ts="<<s<<endl;
```

```
            if(method==3)break;
          }
        case 4:
          {
            char rs='C';
            U32 r=0;
            stime=clock();
            isprime=AKSC4(ii,r);
            etime=clock();
            if(isprime)rs='P';
            if(log==0)cout<<ii<<"\t"<<rs<<"\tM4\t"<<NumBits(ii)<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<endl;
            else fout<<ii<<"\t"<<rs<<"\tM4\t"<<NumBits(ii)<<"bit\t"<<etime-stime<<"ms"<<"\tr="<<r<<endl;
            if(method==4)break;
          }

      }
  }



  system("PAUSE");
  return 0;
}
////////////////////////////////////////////////////////////////////////////
```

Conjecture.cpp

```
/*//////////////////////AKS Conjecture algorithm testing////////////////////////////////////////////
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef unsigned long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////

//AKS Conjecture algorithm
bool AKSC4(const ZZ &n)
{
  U32 r;
  U64 m;
  if(n<6 && n!=4)return true;
  for(r=2;;r++)
  {
    m=n%r;
    if(m==0)return false;
    if((m*m%r)!=1)break;
  }
  ZZ_p::init(n);
  ZZ_pX f(r,1);f-=1;//f=x^r-1
  const ZZ_pXModulus pf(f);
```

```
   ZZ_pX rhs(m,1);rhs-=1;//x^m-1

   ZZ_pX lhs(1,1);lhs-=1;//x-1
   PowerMod(lhs,lhs,n,pf);//(x-1)^n
   if(lhs==rhs)return true;
   else return false;
}
////////////////////////////////////////////////////////////////////////////

int main()
{
   ZZ n;
   U32 start,end;
   bool isprime;


   while(true){

   cout<<"Please input a large integer for AKS Conjecture algorithm to compute:\n";
   cin>>n;

   start=clock();
   isprime=AKSC4(n);
   end=clock();

   if(isprime) cout<<"\n"<<n<<" is a prime!\n";
   else cout<<"\n"<<n<<" is a composite!\n";
   cout<<"Time used:"<<end-start<<"ms.\n";
   }


   system("PAUSE");
   return 0;
}
////////////////////////////////////////////////////////////////////////////
```

CompositeFrequency.cpp

```
/*////////check the composite frequency from all four algorithms//////////////
*
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 primes.

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
```

```
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};

////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   U32 root = (U32)sqrt(n+1);
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}

////////////////////////////////////////////////////////////////////////////
//Compute greatest common divisor of a and b

U32 Gcd(const U32 a,const U32 b)
{
   U32 m=a,n=b;
   if(m==0)return n;
   while(n!=0)
   {
      m%=n;
      m^=n;n^=m;m^=n; //swap(m,n)
   }
   return m;
};

////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(U32 &root,const U32 &n,U32 e)
{
   int bits=0;
   U32 nn=n;
```

```
   while(nn!=0){bits++;nn/=2;}
   if(bits<=e){root=1;return;}

   U32 y;
   U32 pp=ceil(bits*1.0/e);

   root=1;root<<=pp;
   while(true)
   {
     U32 ee=e-1;nn=n;
     while(ee>0){nn/=root;ee--;}
     y=(e-1)*root+nn; y/=e;
     //y=(e-1)*root+n/power_long(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
     if(root==2)break;
   }
}
////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(ZZ &root,const ZZ &n,U32 e)
{
   ZZ y;
   double ll=NumBits(n)*1.0/e;
   power2(root,ceil(ll));
   while(true)
   {
     y=(e-1)*root+n/power(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}
////////////////////////////////////////////////////////////////////////////
//Test whether n is a perfect power: true:false?n=a^b

bool IsPower(const ZZ &n)
{
   ZZ rt;
   U32 i,le=NumBits(n);
   for(i=2;i<le;i++)
   {
      IntRoot(rt,n,i);
      if(power(rt,i)==n)return true;
   }
   return false;
}

bool IsPower(const U32 &n)
{
   U32 rt;

   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}

   for(U32 i=2;i<bits;i++)
   {
      IntRoot(rt,n,i);
      if((rt+n)%2)continue;
      if(power_long(rt,i)==n)return true;
   }
   return false;
}
////////////////////////////////////////////////////////////////////////////
//Return largest prime factor of n

U32 LargestPrimeFactor(U32 n)
{
        if (n < 2) return 1;
```

```
	U32 r = n, p;
	if (r % 2 == 0)
	{
		p = 2;
		do { r /= 2; } while (r % 2 == 0);
	}
	U32 i;
	for (i = 3; i <= r; i += 2)
	{
		if (r % i == 0)
		{
			p = i;
			do { r /= i; } while (r % i == 0);
		}
	}
	return p;
}

////////////////////////////////////////////////////////////////////////////
//return (n^e)%m

U32 PowerOverM(U32 n, U32 e, U32 m)
{
	U64 r = 1;
	U64 t = n % m;
	U32 i;
	for (i = e; i != 0; i /= 2)
	{
		if (i % 2 != 0)
		{
			r=(r*t)%m;
		}
		t=(t*t)%m;
	}
	return r;
}

////////////////////////////////////////////////////////////////////////////
//Order: the least k such that n^k=1 (mod r),assuming (n,r)=1

U32 Order(U32 n,U32 r)
{
   U32 o=1;
   U64 m=n;
   for(;;)
   {
      if(m%r==1)return o;
      m*=n;m%=r;o++;
   }

}

////////////////////////////////////////////////////////////////////////////
//Euler function phi(x):number of integer smaller than  x and coprime to x

U32 EulerPhi(U32 x)
{
   U32 ct=1,i;
   for(i=2;i<x;i++)
   {
      if(Gcd(x,i)==1)ct++;
   }
   return ct;
}

////////////////////////////////////////////////////////////////////////////
//original AKS Algorithm by M. Agrawal, N. Kayal and N. Saxena

bool AKS(U32 &n,U32 &v,U32 &c)
```

```
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {
         q=LargestPrimeFactor(r-1);
         if(q>=(4*sqrt(r)*log(n)/log(2)))if(PowerOverM(n%r,(r-1)/q,r)!=1)break;

      }
      r++;
   }
   s=2*sqrt(r)*log(n)/log(2);v=r;c=s;
   if(r>=n){c=0;return true;}
   return IsPrime(n);

   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
      if(lhs!=rhs)return false;
   }
   return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS Conjecture algorithm

bool AKSC4(const ZZ &n,U32 &v)
{
   U32 r;
   U64 m;
   if(n<6 && n!=4){v=0;return true;}
   for(r=2;;r++)
   {
      m=n%r;
      if(m==0){v=0;return false;}
      if((m*m%r)!=1)break;
   }
   ZZ_p::init(n);
   ZZ_pX f(r,1);f-=1;//f=x^r-1
   const ZZ_pXModulus pf(f);

   ZZ_pX rhs(m,1);rhs-=1;//x^m-1

   ZZ_pX lhs(1,1);lhs-=1;//x-1
   PowerMod(lhs,lhs,n,pf);//(x-1)^n
   if(lhs==rhs){v=r;return true;}
   else return false;
}

////////////////////////////////////////////////////////////////////////////
//AKS Version 3 by M. Agrawal, N. Kayal and N. Saxena

bool AKSV3(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
```

```
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   double m,l=log(n)/log(2); m=l*l*4;
   while(r<n)
   {
     if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
     if(Order(n%r,r)>m)break;
     r++;
   }
   s=floor(2*sqrt(EulerPhi(r))*l);v=r;c=s;
   if(n<=r){c=0;return true;}
   return IsPrime(n);////////////////////////////////////////////////////////////


   ZZ_p::init(to_ZZ(n));//mod n
              ZZ_pX f(r, 1); f -= 1;//f=x^r-1
              const ZZ_pXModulus pf(f);

              ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
   for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs+=a;//x+a
      PowerMod(lhs,lhs,n,pf);//(x+a)^n
      lhs-=a;//(x+a)^n-a
      if(lhs!=rhs)return false;

   }
   return true;
}

//////////////////////////////////////////////////////////////////////////
//AKS algorithm improved by D. Bernstein.

bool AKSDB(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {

         q=LargestPrimeFactor(r-1);
         if(PowerOverM(n%r,(r-1)/q,r)==1){r++;continue;}
         double cm=2*floor(sqrt(r))*log(n),c=0;
         for(s=1;s<q && c<cm;s++) c+=log(q+s-1)-log(s);
         if(c>=cm)break;

      }
      r++;
   }
   v=r;c=s;
   if(n<=r){c=0;return true;}
   return IsPrime(n);////////////////////////////////////////////////////////////


   ZZ_p::init(to_ZZ(n));//mod n
              ZZ_pX f(r, 1); f -= 1;//f=x^r-1
              const ZZ_pXModulus pf(f);

              ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
              for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
```

```
      if(lhs!=rhs)return false;
   }
   return true;
}

////////////////////////////////////////////////////////////////////////////

int main()
{
   int bits;
   U32 start,end;

   U32 nP,nC,nC0,nC1,nC2,nP1,nP2;
   std::ofstream foutDB("CompositeFrequency.DB.txt");
   foutDB<<"#Table of primes and composites decided in different step by AKS algorithms improved by D. Bernstein.\n\n";
   foutDB<<"Bits\tTotal\tP\tP1\tP2\tC\tC0\tC1\tC2\n";

   std::ofstream foutV3("CompositeFrequency.V3.txt");
   foutV3<<"#Table of primes and composites decided in different step by AKS version 3.\n\n";
   foutV3<<"Bits\tTotal\tP\tP1\tP2\tC\tC0\tC1\tC2\n";

   std::ofstream foutV0("CompositeFrequency.AKS.txt");
   foutV0<<"#Table of primes and composites decided in different step by the original AKS algorithm.\n\n";
   foutV0<<"Bits\tTotal\tP\tP1\tP2\tC\tC0\tC1\tC2\n";

   for(bits=4;bits<19;bits++)
   {
      start=1<<(bits-1);
      end=1<<bits;
      U32 i,r,s;
      bool isprime;

      ////////////////////////////////////////////////////////////////////
      //AKS improved by D.Bernstein below
      nP=nC=nC0=nC1=nC2=nP1=nP2=0;
      for(i=start;i<end;i++)
      {
         isprime=AKSDB(i,r,s);
         if(isprime)
            if(s==0)nP1++;
            else nP2++;
         else
         {
            if(s==0)nC1++;else nC0++;

         }

      }
      nP=nP1+nP2;
      nC=start-nP;
      nC2=nC-nC1-nC0;
      foutDB<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nP1<<"\t"<<nP2<<"\t"<<nC<<"\t"<<nC0<<"\t"<<nC1<<"\t"<<nC2<<endl;

      ////////////////////////////////////////////////////////////////////
      //AKS Version 3 below
      nP=nC=nC0=nC1=nC2=nP1=nP2=0;
      for(i=start;i<end;i++)
      {
         isprime=AKSV3(i,r,s);
         if(isprime)
            if(s==0)nP1++;
            else nP2++;
         else
         {
            if(s==0)nC1++;else nC0++;

         }

      }
      nP=nP1+nP2;
```

```
    nC=start-nP;
    nC2=nC-nC1-nC0;
    foutV3<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nP1<<"\t"<<nP2<<"\t"<<nC<<"\t"<<nC0<<"\t"<<nC1<<"\t"<<nC2<<endl;

    ////////////////////////////////////////////////////////////////////
    //AKS original
    nP=nC=nC0=nC1=nC2=nP1=nP2=0;
    for(i=start;i<end;i++)
    {
       isprime=AKS(i,r,s);
       if(isprime)
          if(s==0)nP1++;
          else nP2++;
       else
       {
          if(s==0)nC1++;else nC0++;

       }

    }
    nP=nP1+nP2;
    nC=start-nP;
    nC2=nC-nC1-nC0;
    foutV0<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nP1<<"\t"<<nP2<<"\t"<<nC<<"\t"<<nC0<<"\t"<<nC1<<"\t"<<nC2<<endl;

  }


  return 0;
}
////////////////////////////////////////////////////////////////////////////
```

PerfectPowerFrequency.cpp

```
/*//////////////////////ckeck perfect power frequency//////////////////////////////
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 prime.

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
```

```
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};

////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   //if(n==4)return false;
   U32 root = (U32)sqrt(n+1);
   //cout<<root<<"in IsPrime!\n";
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}

////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e
void IntRoot(U32 &root,const U32 &n,U32 e)
{
   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}
   if(bits<=e){root=1;return;}

   U32 y;
   U32 pp=ceil(bits*1.0/e);

   root=1;root<<=pp;
   while(true)
   {
      U32 ee=e-1;nn=n;
      while(ee>0){nn/=root;ee--;}
      y=(e-1)*root+nn; y/=e;
      if(y>=root)return;
      else root=y;
      if(root==2)break;
   }
}

////////////////////////////////////////////////////////////////////////////
//Test whether n is a perfect power: true:false?n=a^b
bool IsPower(const U32 &n,U32 &root)
{
```

```
    U32 rt;
    root=0;
    int bits=0;
    U32 nn=n;
    while(nn!=0){bits++;nn/=2;}

    for(U32 i=bits-1;i>=2;i--)
    {
        IntRoot(rt,n,i);
        if((rt+n)%2)continue;
        if(power_long(rt,i)==n){root=rt;return true;}

    }
    return false;
}

////////////////////////////////////////////////////////////////////////
int main()
{
    int bits;
    U32 start,end,rt=0;
    U32 nP,nC,nCP,nCPP,nCPC;
    std::ofstream fout("PerfectPwoerFrequency.txt");
    fout<<"#Table of the number of Prime,Composite,Perfect Power,Perfect Power with Prime and Composite root.\n\n";
    fout<<"Bits\t\Total\tPrimes\tCompst\t\PPower\tPPP\tPPC\tT/PPP\tC/PPP\tLog(T/PPP)\tLog(C/PPP)\n";

    for(bits=2;bits<21;bits++)
    {
        start=1<<(bits-1);
        end=1<<bits;
        int i;
        nP=0;nC=0;nCP=0;nCPP=0;nCPC=0;
        for(i=start;i<end;i++)
        {
            //cout<<i<<"\t";
            if(IsPrime(i)){nP++;continue;}
            if(IsPower(i,rt))
            {
                //fout<<i<<" is power of "<<rt<<endl;
                if(IsPrime(rt))nCPP++;
                else nCPC++;
            }
        }
        nC=start-nP;
        nCP=nCPP+nCPC;

if(bits==2)fout<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nC<<"\t"<<nCP<<"\t"<<nCPP<<"\t"<<nCPC<<"\t"<<0<<"\t"<<0<<"\t"<<0<
<"\t\t"<<0<<endl;
        else
fout<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nC<<"\t"<<nCP<<"\t"<<nCPP<<"\t"<<nCPC<<"\t"<<(start/nCPP)<<"\t"<<(nC/nCPP)<
<"\t"<<log2(1.0*start/nCPP)<<"\t\t"<<log2(1.0*nC/nCPP)<<endl;
    }

    return 0;
}
```

SimpleandConjecture.cpp

```
/*///////////////compare simple prime algorithm and AKS cojecture////////////////////////////////
*
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
```

```
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef unsigned long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 primes

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};

////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   //if(n==4)return false;
   U32 root = (U32)sqrt(n+1);
   //cout<<root<<"in IsPrime!\n";
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}
////////////////////////////////////////////////////////////////////////////
//AKS Conjecture algorithm

bool AKSC4(const U32&n)
{
```

```
    U32 r;
    U64 m;
    //if(n<6 && n!=4)return true;
    for(r=2;;r++)
    {
        m=n%r;
        if(m==0)return false;
        if((m*m%r)!=1)break;
    }
    ZZ_p::init(to_ZZ(n));
    ZZ_pX f(r,1);f-=1;//f=x^r-1
    const ZZ_pXModulus pf(f);

    ZZ_pX rhs(m,1);rhs-=1;//x^m-1

    ZZ_pX lhs(1,1);lhs-=1;//x-1
    PowerMod(lhs,lhs,n,pf);//(x-1)^n
    if(lhs==rhs)return true;
    else return false;
}
////////////////////////////////////////////////////////////////////////////////

int main()
{
    int bits;
    U32 start,end;
    U32 ts,te;

    std::ofstream fout("simpleandconjecture.txt");
    fout<<"#Table of comparing simple prime algorithm and AKS conjecture algorithm in average time determine prime and
composite.\n\n";
    fout<<"Bits\tTotal\tP\tC\tAve.P.S\tAve.C.S\tAve.P.C\tAve.C.C\n";

    U32 psimple,csimple,pc4,cc4;
    U32 nC,nP,i;
    bool isprime;
    for(bits=4;bits<31;bits++)
    {
        start=1<<(bits-1);
        end=1<<bits;
        nC=nP=0;
        psimple=csimple=pc4=cc4=0;

        for(i=start;i<end;i++)
        {
            ts=clock();
            isprime=IsPrime(i);
            te=clock();
            te-=ts;
            if(isprime){psimple+=te;nP++;}
            else {csimple+=te;nC++;}
        }

        for(i=start;i<end;i++)
        {
            ts=clock();
            isprime=AKSC4(i);
            te=clock();
            te-=ts;
            if(isprime)pc4+=te;
            else cc4+=te;
        }

fout<<bits<<"\t"<<start<<"\t"<<nP<<"\t"<<nC<<"\t"<<(psimple/nP)<<"\t"<<(csimple/nC)<<"\t"<<(pc4/nP)<<"\t"<<(cc4/nC)<<endl;

    }


    return 0;
```

```
}
////////////////////////////////////////////////////////////////////////////
```

AVESizeStatics.cpp

```
/*////////////////////////////////////////////////////////////////////////////
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 primes.

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};
```

```
////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   //if(n==4)return false;
   U32 root = (U32)sqrt(n+1);
   //cout<<root<<"in IsPrime!\n";
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}

////////////////////////////////////////////////////////////////////////////
//Compute greatest common divisor of a and b

U32 Gcd(const U32 a,const U32 b)
{
   U32 m=a,n=b;
   if(m==0)return n;
   while(n!=0)
   {
      m%=n;
      m^=n;n^=m;m^=n; //swap(m,n)
   }
   return m;
};

////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(U32 &root,const U32 &n,U32 e)
{
   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}
   if(bits<=e){root=1;return;}

   U32 y;
   U32 pp=ceil(bits*1.0/e);

   root=1;root<<=pp;
   while(true)
   {
     U32 ee=e-1;nn=n;
     while(ee>0){nn/=root;ee--;}
     y=(e-1)*root+nn; y/=e;
     //y=(e-1)*root+n/power_long(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
     if(root==2)break;
   }
}
////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(ZZ &root,const ZZ &n,U32 e)
{
   ZZ y;
   double ll=NumBits(n)*1.0/e;
   power2(root,ceil(ll));
   while(true)
   {
     y=(e-1)*root+n/power(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}
////////////////////////////////////////////////////////////////////////////
```

```
//Test whether n is a perfect power: true:false?n=a^b

bool IsPower(const ZZ &n)
{
   ZZ rt;
   U32 i,le=NumBits(n);
   for(i=2;i<le;i++)
   {
      IntRoot(rt,n,i);
      if(power(rt,i)==n)return true;
   }
   return false;
}
bool IsPower(const U32 &n)
{
   U32 rt;

   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}

   for(U32 i=2;i<bits;i++)
   {
      IntRoot(rt,n,i);
      if((rt+n)%2)continue;
      if(power_long(rt,i)==n)return true;
   }
   return false;
}
////////////////////////////////////////////////////////////////////////////
//Return largest prime factor of n

U32 LargestPrimeFactor(U32 n)
{
	if (n < 2) return 1;

	U32 r = n, p;
	if (r % 2 == 0)
	{
		p = 2;
		do { r /= 2; } while (r % 2 == 0);
	}
	U32 i;
	for (i = 3; i <= r; i += 2)
	{
		if (r % i == 0)
		{
			p = i;
			do { r /= i; } while (r % i == 0);
		}
	}
	return p;
}

////////////////////////////////////////////////////////////////////////////
//return (n^e)%m

U32 PowerOverM(U32 n, U32 e, U32 m)
{
	U64 r = 1;
	U64 t = n % m;
	U32 i;
	for (i = e; i != 0; i /= 2)
	{
		if (i % 2 != 0)
		{
			r=(r*t)%m;
		}
		t=(t*t)%m;
	}
```

```
        return r;
}

////////////////////////////////////////////////////////////////////////
//Order: the least k such that n^k=1 (mod r),assuming (n,r)=1

U32 Order(U32 n,U32 r)
{
   U32 o=1;
   U64 m=n;
   for(;;)
   {
      if(m%r==1)return o;
      m*=n;m%=r;o++;
   }

}

////////////////////////////////////////////////////////////////////////
//Euler function phi(x):number of integer smaller than  x and coprime to x

U32 EulerPhi(U32 x)
{
   U32 ct=1,i;
   for(i=2;i<x;i++)
   {
      if(Gcd(x,i)==1)ct++;
   }
   return ct;
}

////////////////////////////////////////////////////////////////////////
//original AKS Algorithm by M. Agrawal, N. Kayal and N. Saxena
bool AKS(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {
         q=LargestPrimeFactor(r-1);
         if(q>=(4*sqrt(r)*log(n)/log(2)))if(PowerOverM(n%r,(r-1)/q,r)!=1)break;

      }
      r++;
   }
   s=2*sqrt(r)*log(n)/log(2);v=r;c=s;
   if(r>=n){c=0;return true;}
   return IsPrime(n);////////////////////////////////////////////////////////

   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
      if(lhs!=rhs)return false;
   }
   return true;
}
```

```
////////////////////////////////////////////////////////////////////////////
//AKS Conjecture algorithm

bool AKSC4(const ZZ &n,U32 &v)
{
   U32 r;
   U64 m;
   if(n<6 && n!=4){v=0;return true;}
   for(r=2;;r++)
   {
      m=n%r;
      if(m==0){v=0;return false;}
      if((m*m%r)!=1)break;
   }
   ZZ_p::init(n);
   ZZ_pX f(r,1);f-=1;//f=x^r-1
   const ZZ_pXModulus pf(f);

   ZZ_pX rhs(m,1);rhs-=1;//x^m-1

   ZZ_pX lhs(1,1);lhs-=1;//x-1
   PowerMod(lhs,lhs,n,pf);//(x-1)^n
   if(lhs==rhs){v=r;return true;}
   else return false;
}

////////////////////////////////////////////////////////////////////////////
//AKS version 3 by M. Agrawal, N. Kayal and N. Saxena

bool AKSV3(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   double m,l=log(n)/log(2); m=l*l*4;
   while(r<n)
   {
     if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
     if(Order(n%r,r)>m)break;
     r++;
   }
   s=floor(2*sqrt(EulerPhi(r))*l);v=r;c=s;
   if(n<=r){c=0;return true;}
   return IsPrime(n);//////////////////////////////////////////////////////////


   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
   for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs+=a;//x+a
      PowerMod(lhs,lhs,n,pf);//(x+a)^n
      lhs-=a;//(x+a)^n-a
      if(lhs!=rhs)return false;

   }
   return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS algorithm improved by D. Bernstein

bool AKSDB(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
```

```
    if(n<4)return true;
    if(n==4)return false;
    if(IsPower(n)){v=0;c=100;return false;}
    U32 r=3,q,a,s;
    while(r<n)
    {
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {

        q=LargestPrimeFactor(r-1);
        if(PowerOverM(n%r,(r-1)/q,r)==1){r++;continue;}
        double cm=2*floor(sqrt(r))*log(n),c=0;
        for(s=1;s<q && c<cm;s++) c+=log(q+s-1)-log(s);
        if(c>=cm)break;

      }
      r++;
    }
    v=r;c=s;
    if(n<=r){c=0;return true;}
    return IsPrime(n);//////////////////////////////////////////////////////////////


    ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
    {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
      if(lhs!=rhs)return false;
    }
    return true;
}

////////////////////////////////////////////////////////////////////////////

int main()
{
    int bits;
    U32 start,end;

    std::ofstream foutDB("Size.DB.txt");
    foutDB<<"Bits\tTotal\tTSize\n";

    std::ofstream foutV3("Size.V3.txt");
    foutV3<<"Bits\tTotal\tTSize\n";

    std::ofstream foutV0("Size.AKS.txt");
    foutV0<<"Bits\tTotal\tTSize\n";

    for(bits=4;bits<19;bits++)
    {
      start=1<<(bits-1);
      end=1<<bits;
      U32 i,r,s;
      U64 totalsize=0;
      bool isprime;

      ////////////////////////////////////////////////////////////////
      //AKS improved by D.Bernstein below

      for(i=start;i<end;i++)
      {
        isprime=AKSDB(i,r,s);
        if(isprime) if(s>0) totalsize+=s;
```

```
    }
    foutDB<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;

    ////////////////////////////////////////////////////////////////////////////////////////////
    //AKS Version 3 below

    totalsize=0;
    for(i=start;i<end;i++)
    {
       isprime=AKS(i,r,s);
       if(isprime) if(s>0) totalsize+=s;
    }
    foutV0<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;
    ////////////////////////////////////////////////////////////////////
    //AKS original

    totalsize=0;
    for(i=start;i<end;i++)
    {
       isprime=AKSV3(i,r,s);
       if(isprime) if(s>0) totalsize+=s;
    }
    foutV3<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;
  }



  return 0;
}
////////////////////////////////////////////////////////////////////////////
```

Badcse.cpp

```
/*///////compute the average value of r and s for all four algorithms//////////
*
* In this version,we assume that only n is big integer,r,q,s are  below 32
* bit unsigned int.
*
* written by Tong Jin
* version 0.1
*////////////////////////////////////////////////////////////////////////
#include <NTL/ZZ_pX.h>
#include <math.h>
#include <time.h>
#include <fstream>
#include <iostream>
#define NTL_NO_MIN_MAX

NTL_CLIENT

typedef long U32;
typedef  __int64 U64;

////////////////////////////////////////////////////////////////////////////
//Simple prime test procedure for n>1, program stores 1000 primes.

static int
prm[1000]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97,101,103,107,109,113,127,131,137,139,149,151,
157,163,167,173,179,181,191,193,197,199,211,223,227,229,233,239,241,251,257,263,269,271,277,281,283,293,307,311,313,317,331
,337,347,349,353,359,367,373,379,383,389,397,401,409,419,421,431,433,439,443,449,457,461,463,467,479,487,491,499,503,509,52
1,523,541,547,557,563,569,571,577,587,593,599,601,607,613,617,619,631,641,643,647,653,659,661,673,677,683,691,701,709,719,7
27,733,739,743,751,757,761,769,773,787,797,809,811,821,823,827,829,839,853,857,859,863,877,881,883,887,907,911,919,929,937,
941,947,953,967,971,977,983,991,997,1009,1013,1019,1021,1031,1033,1039,1049,1051,1061,1063,1069,1087,1091,1093,1097,1103,
1109,1117,1123,1129,1151,1153,1163,1171,1181,1187,1193,1201,1213,1217,1223,1229,1231,1237,1249,1259,1277,1279,1283,1289,
1291,1297,1301,1303,1307,1319,1321,1327,1361,1367,1373,1381,1399,1409,1423,1427,1429,1433,1439,1447,1451,1453,1459,1471,
1481,1483,1487,1489,1493,1499,1511,1523,1531,1543,1549,1553,1559,1567,1571,1579,1583,1597,1601,1607,1609,1613,1619,1621,
1627,1637,1657,1663,1667,1669,1693,1697,1699,1709,1721,1723,1733,1741,1747,1753,1759,1777,1783,1787,1789,1801,1811,1823,
1831,1847,1861,1867,1871,1873,1877,1879,1889,1901,1907,1913,1931,1933,1949,1951,1973,1979,1987,1993,1997,1999,2003,2011,
2017,2027,2029,2039,2053,2063,2069,2081,2083,2087,2089,2099,2111,2113,2129,2131,2137,2141,2143,2153,2161,2179,2203,2207,
2213,2221,2237,2239,2243,2251,2267,2269,2273,2281,2287,2293,2297,2309,2311,2333,2339,2341,2347,2351,2357,2371,2377,2381,
```

```
2383,2389,2393,2399,2411,2417,2423,2437,2441,2447,2459,2467,2473,2477,2503,2521,2531,2539,2543,2549,2551,2557,2579,2591,
2593,2609,2617,2621,2633,2647,2657,2659,2663,2671,2677,2683,2687,2689,2693,2699,2707,2711,2713,2719,2729,2731,2741,2749,
2753,2767,2777,2789,2791,2797,2801,2803,2819,2833,2837,2843,2851,2857,2861,2879,2887,2897,2903,2909,2917,2927,2939,2953,
2957,2963,2969,2971,2999,3001,3011,3019,3023,3037,3041,3049,3061,3067,3079,3083,3089,3109,3119,3121,3137,3163,3167,3169,
3181,3187,3191,3203,3209,3217,3221,3229,3251,3253,3257,3259,3271,3299,3301,3307,3313,3319,3323,3329,3331,3343,3347,3359,
3361,3371,3373,3389,3391,3407,3413,3433,3449,3457,3461,3463,3467,3469,3491,3499,3511,3517,3527,3529,3533,3539,3541,3547,
3557,3559,3571,3581,3583,3593,3607,3613,3617,3623,3631,3637,3643,3659,3671,3673,3677,3691,3697,3701,3709,3719,3727,3733,
3739,3761,3767,3769,3779,3793,3797,3803,3821,3823,3833,3847,3851,3853,3863,3877,3881,3889,3907,3911,3917,3919,3923,3929,
3931,3943,3947,3967,3989,4001,4003,4007,4013,4019,4021,4027,4049,4051,4057,4073,4079,4091,4093,4099,4111,4127,4129,4133,
4139,4153,4157,4159,4177,4201,4211,4217,4219,4229,4231,4241,4243,4253,4259,4261,4271,4273,4283,4289,4297,4327,4337,4339,
4349,4357,4363,4373,4391,4397,4409,4421,4423,4441,4447,4451,4457,4463,4481,4483,4493,4507,4513,4517,4519,4523,4547,4549,
4561,4567,4583,4591,4597,4603,4621,4637,4639,4643,4649,4651,4657,4663,4673,4679,4691,4703,4721,4723,4729,4733,4751,4759,
4783,4787,4789,4793,4799,4801,4813,4817,4831,4861,4871,4877,4889,4903,4909,4919,4931,4933,4937,4943,4951,4957,4967,4969,
4973,4987,4993,4999,5003,5009,5011,5021,5023,5039,5051,5059,5077,5081,5087,5099,5101,5107,5113,5119,5147,5153,5167,5171,
5179,5189,5197,5209,5227,5231,5233,5237,5261,5273,5279,5281,5297,5303,5309,5323,5333,5347,5351,5381,5387,5393,5399,5407,
5413,5417,5419,5431,5437,5441,5443,5449,5471,5477,5479,5483,5501,5503,5507,5519,5521,5527,5531,5557,5563,5569,5573,5581,
5591,5623,5639,5641,5647,5651,5653,5657,5659,5669,5683,5689,5693,5701,5711,5717,5737,5741,5743,5749,5779,5783,5791,5801,
5807,5813,5821,5827,5839,5843,5849,5851,5857,5861,5867,5869,5879,5881,5897,5903,5923,5927,5939,5953,5981,5987,6007,6011,
6029,6037,6043,6047,6053,6067,6073,6079,6089,6091,6101,6113,6121,6131,6133,6143,6151,6163,6173,6197,6199,6203,6211,6217,
6221,6229,6247,6257,6263,6269,6271,6277,6287,6299,6301,6311,6317,6323,6329,6337,6343,6353,6359,6361,6367,6373,6379,6389,
6397,6421,6427,6449,6451,6469,6473,6481,6491,6521,6529,6547,6551,6553,6563,6569,6571,6577,6581,6599,6607,6619,6637,6653,
6659,6661,6673,6679,6689,6691,6701,6703,6709,6719,6733,6737,6761,6763,6779,6781,6791,6793,6803,6823,6827,6829,6833,6841,
6857,6863,6869,6871,6883,6899,6907,6911,6917,6947,6949,6959,6961,6967,6971,6977,6983,6991,6997,7001,7013,7019,7027,7039,
7043,7057,7069,7079,7103,7109,7121,7127,7129,7151,7159,7177,7187,7193,7207,7211,7213,7219,7229,7237,7243,7247,7253,7283,
7297,7307,7309,7321,7331,7333,7349,7351,7369,7393,7411,7417,7433,7451,7457,7459,7477,7481,7487,7489,7499,7507,7517,7523,
7529,7537,7541,7547,7549,7559,7561,7573,7577,7583,7589,7591,7603,7607,7621,7639,7643,7649,7669,7673,7681,7687,7691,7699,
7703,7717,7723,7727,7741,7753,7757,7759,7789,7793,7817,7823,7829,7841,7853,7867,7873,7877,7879,7883,7901,7907,7919};

////////////////////////////////////////////////////////////////////////////
bool IsPrime(U32 n)
{
   if(n==2)return true;
   //if(n==4)return false;
   U32 root = (U32)sqrt(n+1);
   //cout<<root<<"in IsPrime!\n";
   U32 stage=7907;if(root<stage)stage=root;
            U32 i;
            for (i = 0; prm[i] <=stage; i++)if (n % prm[i] == 0)return false;
            for(i=7919;i<=root;i+=2)if(n%i==0)return false;
            return true;
}

////////////////////////////////////////////////////////////////////////////
//Compute greatest common divisor of a and b

U32 Gcd(const U32 a,const U32 b)
{
   U32 m=a,n=b;
   if(m==0)return n;
   while(n!=0)
   {
      m%=n;
      m^=n;n^=m;m^=n; //swap(m,n)
   }
   return m;
};

////////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(U32 &root,const U32 &n,U32 e)
{
   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}
   if(bits<=e){root=1;return;}

   U32 y;
   U32 pp=ceil(bits*1.0/e);
```

```
   root=1;root<<=pp;
   while(true)
   {
     U32 ee=e-1;nn=n;
     while(ee>0){nn/=root;ee--;}
     y=(e-1)*root+nn; y/=e;
     //y=(e-1)*root+n/power_long(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
     if(root==2)break;
   }
}
//////////////////////////////////////////////////////////////////////////
//Find and int root of n such that root^e<=n<(root+1)^e

void IntRoot(ZZ &root,const ZZ &n,U32 e)
{
   ZZ y;
   double ll=NumBits(n)*1.0/e;
   power2(root,ceil(ll));
   while(true)
   {
     y=(e-1)*root+n/power(root,e-1); y/=e;
     if(y>=root)return;
     else root=y;
   }
}
//////////////////////////////////////////////////////////////////////////
//Test whether n is a perfect power: true:false?n=a^b

bool IsPower(const ZZ &n)
{
   ZZ rt;
   U32 i,le=NumBits(n);
   for(i=2;i<le;i++)
   {
      IntRoot(rt,n,i);
      if(power(rt,i)==n)return true;
   }
   return false;
}
bool IsPower(const U32 &n)
{
   U32 rt;

   int bits=0;
   U32 nn=n;
   while(nn!=0){bits++;nn/=2;}

   for(U32 i=2;i<bits;i++)
   {
      IntRoot(rt,n,i);
      if((rt+n)%2)continue;
      if(power_long(rt,i)==n)return true;
   }
   return false;
}
//////////////////////////////////////////////////////////////////////////
//Return largest prime factor of n

U32 LargestPrimeFactor(U32 n)
{
        if (n < 2) return 1;

        U32 r = n, p;
        if (r % 2 == 0)
        {
                p = 2;
                do { r /= 2; } while (r % 2 == 0);
        }
```

```
	U32 i;
	for (i = 3; i <= r; i += 2)
	{
		if (r % i == 0)
		{
			p = i;
			do { r /= i; } while (r % i == 0);
		}
	}
	return p;
}

////////////////////////////////////////////////////////////////////////////
//return (n^e)%m

U32 PowerOverM(U32 n, U32 e, U32 m)
{
	U64 r = 1;
	U64 t = n % m;
	U32 i;
	for (i = e; i != 0; i /= 2)
	{
		if (i % 2 != 0)
		{
			r=(r*t)%m;
		}
		t=(t*t)%m;
	}
	return r;
}

////////////////////////////////////////////////////////////////////////////
//Order: the least k such that n^k=1 (mod r),assuming (n,r)=1

U32 Order(U32 n,U32 r)
{
   U32 o=1;
   U64 m=n;
   for(;;)
   {
      if(m%r==1)return o;
      m*=n;m%=r;o++;
   }

}

////////////////////////////////////////////////////////////////////////////
//Euler function phi(x):number of integer smaller than  x and coprime to x

U32 EulerPhi(U32 x)
{
   U32 ct=1,i;
   for(i=2;i<x;i++)
   {
      if(Gcd(x,i)==1)ct++;
   }
   return ct;
}

////////////////////////////////////////////////////////////////////////////
//original AKS Algorithm by M. Agrawal, N. Kayal and N. Saxena
bool AKS(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   while(r<n)
   {
```

```
      if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
      if(IsPrime(r))
      {
         q=LargestPrimeFactor(r-1);
         if(q>=(4*sqrt(r)*log(n)/log(2)))if(PowerOverM(n%r,(r-1)/q,r)!=1)break;

      }
      r++;
   }
   s=2*sqrt(r)*log(n)/log(2);v=r;c=s;
   if(r>=n){c=0;return true;}
   return IsPrime(n);////////////////////////////////////////////////////////////

   ZZ_p::init(to_ZZ(n));//mod n
            ZZ_pX f(r, 1); f -= 1;//f=x^r-1
            const ZZ_pXModulus pf(f);

            ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
            for(a=1;a<=s;a++)
   {
      ZZ_pX lhs(1,1);lhs-=a;//x-a
      PowerMod(lhs,lhs,n,pf);//(x-a)^n
      lhs+=a;//(x-a)^n+a
      if(lhs!=rhs)return false;
   }
   return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS Conjecture algorithm

bool AKSC4(const ZZ &n,U32 &v)
{
   U32 r;
   U64 m;
   if(n<6 && n!=4){v=0;return true;}
   for(r=2;;r++)
   {
      m=n%r;
      if(m==0){v=0;return false;}
      if((m*m%r)!=1)break;
   }
   ZZ_p::init(n);
   ZZ_pX f(r,1);f-=1;//f=x^r-1
   const ZZ_pXModulus pf(f);

   ZZ_pX rhs(m,1);rhs-=1;//x^m-1

   ZZ_pX lhs(1,1);lhs-=1;//x-1
   PowerMod(lhs,lhs,n,pf);//(x-1)^n
   if(lhs==rhs){v=r;return true;}
   else return false;
}

////////////////////////////////////////////////////////////////////////////
//AKS version 3 by M. Agrawal, N. Kayal and N. Saxena

bool AKSV3(U32 &n,U32 &v,U32 &c)
{
   if(n==1)return false;
   if(n<4)return true;
   if(n==4)return false;
   if(IsPower(n)){v=0;c=100;return false;}
   U32 r=3,q,a,s;
   double m,l=log(n)/log(2); m=l*l*4;
   while(r<n)
   {
     if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
     if(Order(n%r,r)>m)break;
     r++;
```

```
    }
    s=floor(2*sqrt(EulerPhi(r))*l);v=r;c=s;
    if(n<=r){c=0;return true;}
    return IsPrime(n);////////////////////////////////////////////////////////


    ZZ_p::init(to_ZZ(n));//mod n
                ZZ_pX f(r, 1); f -= 1;//f=x^r-1
                const ZZ_pXModulus pf(f);

                ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
    for(a=1;a<=s;a++)
    {
       ZZ_pX lhs(1,1);lhs+=a;//x+a
       PowerMod(lhs,lhs,n,pf);//(x+a)^n
       lhs-=a;//(x+a)^n-a
       if(lhs!=rhs)return false;

    }
    return true;
}

////////////////////////////////////////////////////////////////////////////
//AKS algorithm improved by D. Bernstein

bool AKSDB(U32 &n,U32 &v,U32 &c)
{
    if(n==1)return false;
    if(n<4)return true;
    if(n==4)return false;
    if(IsPower(n)){v=0;c=100;return false;}
    U32 r=3,q,a,s;
    while(r<n)
    {
       if(Gcd(n%r,r)!=1){v=0;c=0;return false;}
       if(IsPrime(r))
       {

          q=LargestPrimeFactor(r-1);
          if(PowerOverM(n%r,(r-1)/q,r)==1){r++;continue;}
          double cm=2*floor(sqrt(r))*log(n),c=0;
          for(s=1;s<q && c<cm;s++) c+=log(q+s-1)-log(s);
          if(c>=cm)break;

       }
       r++;
    }
    v=r;c=s;
    if(n<=r){c=0;return true;}
    return IsPrime(n);////////////////////////////////////////////////////////


    ZZ_p::init(to_ZZ(n));//mod n
                ZZ_pX f(r, 1); f -= 1;//f=x^r-1
                const ZZ_pXModulus pf(f);

                ZZ_pX rhs(n%r,1);//x^{n%r}=x^n mod(x^r-1)
                for(a=1;a<=s;a++)
    {
       ZZ_pX lhs(1,1);lhs-=a;//x-a
       PowerMod(lhs,lhs,n,pf);//(x-a)^n
       lhs+=a;//(x-a)^n+a
       if(lhs!=rhs)return false;
    }
    return true;
}

////////////////////////////////////////////////////////////////////////////

int main()
```

```
{
   int bits;
   U32 start,end;

   std::ofstream foutDB("AVESize.DB.txt");
   foutDB<<"#Table of average value r and s compute by AKS algorithm improved by D. Bernstein.\n\n";
   foutDB<<"Bits\tTotal\tTSize\n";

   std::ofstream foutV3("AVESize.V3.txt");
   foutV3<<"#Table of average value r and s compute by AKS Version 3 algorithm.\n\n";
   foutV3<<"Bits\tTotal\tTSize\n";

   std::ofstream foutV0("AVESize.AKS.txt");
   foutV0<<"#Table of average value r and s compute by original AKS algorithm.\n\n";
   foutV0<<"Bits\tTotal\tTSize\n";

   for(bits=4;bits<19;bits++)
   {
      start=1<<(bits-1);
      end=1<<bits;
      U32 i,r,s;
      U64 totalsize=0;
      bool isprime;

      ////////////////////////////////////////////////////////////////////
      //AKS improved by D.Bernstein below

      for(i=start;i<end;i++)
      {
         isprime=AKSDB(i,r,s);
         if(isprime) if(s>0) totalsize+=s;
      }
      foutDB<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;

      ////////////////////////////////////////////////////////////////////////////////////////////////
      //AKS Version 3 below

      totalsize=0;
      for(i=start;i<end;i++)
      {
         isprime=AKS(i,r,s);
         if(isprime) if(s>0) totalsize+=s;
      }
      foutV0<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;
      ////////////////////////////////////////////////////////////////////
      //AKS original

      totalsize=0;
      for(i=start;i<end;i++)
      {
         isprime=AKSV3(i,r,s);
         if(isprime) if(s>0) totalsize+=s;
      }
      foutV3<<bits<<"\t"<<start<<"\t"<<totalsize<<endl;
   }



   return 0;
}
////////////////////////////////////////////////////////////////////////////
```