Matematisk-naturvetenskapliga fakulteten

Examensarbete i matematik
C-nivå, 10 poäng

Handledare:	Klas Markström
Examinator:	Robert Johansson
Författare:	Josef Hemmingsson, j@josef.nu

Januari 2003

The AKS Algorithm

Josef Hemmingsson

Matematiska institutionen
Umeå universitet
901 87 Umeå

1 Abstract

The AKS algorithm is the first deterministic polynomial-time algorithm that determines whether a given integer is prime or composite. In this thesis I explain how the algorithm works and prove that it detects primes correctly. This thesis also includes an example how to implement the algorithm in Maple.

2 Acknowledgement

This is the thesis for my Bachelor degree in Mathematics. It was accomplished during the period from September 2002 to January 2003.

I would like to thank my supervisor Klas Markström for help and support during planning and accomplishment of this thesis.

Thanks to my fellow students, teachers and the staff at the Department of Mathematics at Umeå University. Thank you for a memorable time.

Josef Hemmingsson
Umeå, January 2003

Contents

3 Introduction and definitions

3.1 Introduction

Prime numbers are both useful and interesting. A prime is a number only divisible by 1 and itself. It stands alone, not built up by multiplying other numbers. It is powerful to use in cryptation, but it is also a challenge for your mind.

1. Is a given number a prime?
2. What is the largest known prime?
3. Are there infinitely many of them?

Well, the answer to the last question is "yes" and the proof is rather easy [JJ98]. In order to answer the second question it is useful to know how to answer the first. Is a given number, possibly very large, a prime?

In cryptography, prime numbers play a big role. The well known RSA cryptosystem is based on multiplying two large prime numbers, and the product is used as a key when encrypting and decrypting information [Atr]. One of the factors that influence the amount of security is the size of the prime numbers. The larger primes used, the harder it is to hack the system.

Prime numbers and cryptosystems are integrated into our daily life. For example, everytime you use your creditcard, a prime number in an algorithm encrypts your code. Then at the bank, the code is decrypted and the payment is confirmed. The confirmation is encrypted and sent to the store, where the information is decrypted and your payment is confirmed to the store.

So, for secure cryptation we need large primes, and it is the mathematicians job to find them.

3.2 Definitions

The most important definition in this thesis is maybe the easiest to understand. It is the definition of prime numbers.

Definition 3.1 (Prime number).
A number $p \geq 2$ is prime if it's positive divisors are 1 and p itself.

Example 3.2 (A prime number).
7 is prime since no other number divides 7. The first prime numbers are 2, 3, 5, 7, 11, 13 . . . since their only divisors are 1 and the number itself.

Definition 3.3 (Composite number).
A number $n \geq 2$ is composite if $\exists\ k$, $1 < k < n$, such that $k|n$.

Example 3.4 (A composite number).
21 is composite since $1 < 7 < 21$ and $7|21$. The first composite numbers are 4, 6, 8, 9, 10, 12, 14, 15 . . .

Definition 3.5 (Largest Prime Factor).
n is an integer with prime factorization $n = p_1^{a_1} \cdots p_k^{a_k}$ where $p_i < p_j$ for $i < j$. Then p_k is the largest prime factor of n, and is denoted $P(n)$.

Definition 3.6 (Common Divisor).
If $c|a$ and $c|b$ we say that c is a common divisor of a and b. $gcd(a,b)$ denotes the greatest common divisor of a and b.

Definition 3.7 (Coprime).
If $gcd(d,e) = 1$ we say that d and e are coprime, which means that their only common divisor is 1.

Definition 3.8 (Euler's function).

$$\varphi(n) = |\ \{\ i \mid 1\ \leq\ i\ \leq\ n \textit{ and } gcd(i,n) = 1\}\ |$$

which is the number of positive integers, less or equal to n, coprime to n.

Definition 3.9.
$q^k \parallel p$ denotes that $q^k|p$ but q^{k+1} does not divide p.

Definition 3.10 (Perfect-Power).
A number $n > 1$ is a perfect power if there are numbers a and $b \geq 2$ such that $n = a^b$.

Definition 3.11 (Modulus with a polynomial).
If $f(x)$ and $q(x)$ are polynomials and p is a number, $f(x) \pmod{q(x), p}$ evaluates $f(x)$ over $\mathbb{Z}_p[x]/q(x)$.

Definition 3.12 (Unit).
a is a unit $\pmod n$ if $ab \equiv 1 \pmod n$ for some integer b.

Definition 3.13 (Primitive root).
Let U_n denote the group of units in $\mathbb{Z}_n$. If U_n is cyclic, then any generator g for U_n is a primitive root $\pmod n$.

Definition 3.14 (Sophie Germain primes).
If both r and $\frac{r-1}{2}$ are primes, $\frac{r-1}{2}$ is a Sophie Germain prime and r is a co-Sophie Germain prime.

3.3 A theorem and two well known algorithms

Theorem 3.15 (Wilson's theorem [JJ98]).
p is prime iff

$$(p-1)! \equiv -1 \pmod p \tag{1}$$

At first this might seem to be a solution to the problem of deciding if p is prime. Just calculate the factorial and then check the congruence. But the problem with this theorem is that calculating $(p-1)!$ has very high complexity. It may be calculated fast for small p:s, but it is not useful at all when working with large numbers.

Since we after a short while begin to work with rather large numbers, we need algorithms that handle large numbers in as short time as possible. One way to measure the time needed to solve a problem, is complexity analysis (read more in chapter 8). When working with computers complexity analysis may also be used to measure how much memory is needed to implement an algorithm.

$\mathcal{O}(f(n))$ is the notation for the complexity, where $f(n)$ is a function that gives us an idea of how large the complexity is.

Complexity when detecting primes is based on how many bits the number consists of. The number of bits required for n is $\lceil \log_{10} n \rceil$. Therefore $\mathcal{O}(n)$ is exponential and $\mathcal{O}(\log n)$ is linear in the number of bits.

There are several easy ways to get a list of primes. For example the sieve of Eratosthenes, a simple but useful algorithm.

Algorithm 3.16 (The sieve of Eratosthenes).
Input $n > 2$

```
1. Make a list of the numbers from 2 to $n$.
2. $d \leftarrow 2$.
3. Erase every $d$:th number following $d$ ($2d$, $3d$, $4d \ldots$).
4. Give $d$ the value of the first unerased number after $d$.
5. Repeat from step 3 until $d \geq n$.
```

Eratosthenes algorithm in example 3.16 gives us a list of all the primes up to n. The algorithm erases all numbers that are multiples of some other number. The numbers left are the ones that are not multiples, indeed the primes. This algorithm is rather easy to understand, but it needs a lot of memory to implement.

What if we want to decide whether a given number is a prime or not?
We could, of course, use the simplest possible algorithm.

Algorithm 3.17 (Is n prime?).
Input $n > 2$

```
1. $d \leftarrow 2$.
2. If $d|n$, output COMPOSITE.
3. if $d = n - 1$, output PRIME.
4. else $d \leftarrow d + 1$.
5. Repeat from step 2.
```

The algorithm (3.17) tries to divide the given number with all smaller numbers. It is obvious that the algorithm needs $n - 2$ tries, since it tries to divide with all the first n numbers, except from 1 and n. We see that when n is large, the algorithm needs a lot of time since it has exponential complexity $\mathcal{O}(n)$.

Can we do this faster? Well, for example we do not need to try any number bigger than $\sqrt{n}$. Because if n is divisible by a number larger than $\sqrt{n}$, it is a product of that number and a number less than $\sqrt{n}$, which we already would have found. Now the algorithm gets complexity $\mathcal{O}(\sqrt{n})$, but is still very slow for large n.

So we need a new way of thinking in order to find an algorithm that has lower time complexity. For a long time, it has been a challenge for mathematicians to find an algorithm that determines whether n is a prime or composite in polynomial time.

In August 2002, three Indian mathematicians presented a report with a new algorithm [AKS02]. Their names are Manindra Agrawal, Neeraj Kayal and Nitin Saxena, and their algorithm, the AKS algorithm, determines whether a given number is a prime or composite in polynomial time.

4 Theory and pseudoprimes

In this chapter I will introduce the definitions of pseudoprimes and witnesses. They may be used to determine if a number is composite, and they also play an interesting role in the proof of correctness for the AKS algorithm.
First we need a well known theorem.

4.1 Fermat's little theorem

Theorem 4.1 (Fermat's little theorem [Fra99]).
If p is a prime and coprime to a, then

$$a^{p-1} \equiv 1 \pmod{p} \tag{2}$$

Using Fermat's little theorem is one way to explain pseudoprimes. The statement (2) is valid for all primes, but we also know that it holds for some composites.

Example 4.2.
49 is coprime to 75 and

$$49^{74} \equiv 1 \pmod{75}$$

We see that the statement (2) from Fermat's little theorem is valid, but 75 is composite.

4.2 Pseudoprimes and witnesses

Definition 4.3 (Pseudoprimes).
Assume that a statement S is true for all primes, but not for all n. Then n_0 is a S-pseudoprime if S is true for n_0.

This means that some composites sometimes behave as if they were primes. At first this may be confusing, but using the knowledge about pseudoprimes the right way, we may for example be able to conclude that a number is composite faster. Because if the number is not a pseudoprime, it is of course not a prime.

Example 4.4.
49 and 75 are coprime, and

$$49^{74} \equiv 1 \pmod{75}$$

The statement (2) is valid, but 75 is indeed composite. We say that 75 is a Fermat pseudoprime base 49. But as soon as we examine

$$48^{74} \equiv 9 \pmod{75} \tag{3}$$

we see that 75 is not a pseudoprime base 48. This implies 75 is not a prime.

We may use Fermats little theorem to show that a given number is composite. If the statement fails for some given number, we know that the number is not a prime, thus it is composite.

Definition 4.5 (Fermat Witness).
Given a number n, a is a Fermat witness for n if

$$a^{n-1} \not\equiv 1 \pmod{n}.$$

Recall (3) where congruence (2) does not hold. We see that 48 is a Fermat witness for 75, since 75 is not a pseudoprime base 48. We then know 75 is not prime. Using a witness is one of the easiest ways to show that the given number is composite. We only need to find one witness, to be sure that the number is not a prime.

4.3 Strong pseudoprimes

There are more ways to determine that a given number is composite. Let's take a closer look at numbers that are called strong pseudoprimes.

Theorem 4.6.
Suppose that n is a prime and $n \geq 3$. n is then odd, and we may write $n-1$ on the form $n-1 = 2^s t$, where t is odd. If n does not divide a, then either

$$\begin{cases} a^t \equiv 1 \pmod{n} \text{ or} \\ a^{2^i t} \equiv -1 \pmod{n} \text{ for some } i,\ 0 \leq i \leq s-1 \end{cases} \tag{4}$$

Proof 4.6. With Fermat's little theorem and the knowledge that for any odd prime n, the only possible solution to $x^2 \equiv 1 \pmod{n}$ is $x \equiv \pm 1 \pmod{n}$, gives us the theorem. □

Definition 4.7 (Strong pseudoprime).
n is a strong pseudoprime base a if n is an odd composite $n-1 = 2^s t$, with t odd, and (4) holds.

We now create a set including all bases for which n is a strong pseudoprime.

Definition 4.8 (Set of strong pseudoprime bases).

$$S(n) = \{\ a \pmod{n} \mid n \textit{ is a strong pseudoprime in base } a\ \}$$

and if we let $S_n = |S(n)|$, the size of $S(n)$, we get the following interesting theorem:

Theorem 4.9 (Strong pseudoprime base distribution [CP01]).
For each odd composite number $n > 9$, we have $S_n \leq \frac{1}{4}\varphi(n)$.

Definition 4.10 (Witness).
If n is an odd composite and a is a number, $1 \leq a \leq n-1$, for which (4) is not fulfilled, we say that a is a witness for n.

Theorem 4.9 is very useful when evaluating probability for a number being prime. We may use witnesses and strong pseudoprimes when showing that a given number is composite. This is shown in the following algorithm.

Algorithm 4.11 (Finding witnesses).
Input : integer $n > 1$, $k \geq 1$.

```
1. $q \leftarrow 1$.
2. for $j$ from 1 to $k$.
3.         Choose a random number $i$, $2 \leq i \leq n-2$.
4.         If $i$ is a witness for $n$, output COMPOSITE.
5.         If $i$ is not a witness for $n$, $q = q * \frac{1}{4}$.
6. Output $q$.
```

Input n is the number we want to check and k is how many times we want the *for* loop to iterate.
$\langle 1 \rangle$ q is the probability that n is a composite. $\langle 2 \rangle$ Variable j is just a dummy counter in the *for* loop where we first $\langle 3 \rangle$ choose a random number. If it is a witness $\langle 4 \rangle$ we may be sure that n is composite.
$\langle 5 \rangle$ We know from Theorem 4.9 that if we choose a random number i, $2 \leq i \leq n-2$, and n is composite we have at least the probability of $\frac{3}{4}$ that i will be a witness for n. So, if i is not a witness, the probability for n being composite is at most $\frac{1}{4}$.
If we then go back to $\langle 3 \rangle$ and again choose a random number i_2, and i_2 is not a witness for n, we have the probability of $(\frac{1}{4})^2$ that n is composite.
$\langle 6 \rangle$ We see that after k iterations of the *for* loop, all failing to find a witness, the probability that n is composite is as small as $(\frac{1}{4})^k$. With k large, $(\frac{1}{4})^k$ is small, but never zero.

How does this help us? Most importantly if n is composite, we will probably find a witness after a few iterations. And if we after many iterations still have not found a witness, we may assume that n is a prime. The probability that n is composite can be made arbitrarily small, in particular less than other sources of error.

In the proof of correctness (chapter 7) I will show that if the AKS algorithm finds no witness within a certain interval, we can be sure that the input n is prime.

4.4 The Miller primality test

The Extended Riemann Hypothesis is too extensive to be explained in this thesis, so if the reader is interested in details I recommend [CP01].

If the Extended Riemann Hypothesis is true, we may in a rather simple way decide whether a number is prime or not. The test consists of two algorithms, one that decides whether the given number is a strong pseudoprime in a given base, and the other algorithm calls the first algorithm with different bases.

Algorithm 4.12 (Strong probable prime test [CP01]).
Input $n > 3$ and $1 < a < n - 1$.

```
1. $t \leftarrow \frac{(n-1)}{2^s}$ where $2^s \parallel n - 1$.
2. $b \leftarrow a^t \pmod{n}$.
3. if $b = 1$ or $b = n - 1$, output TRUE.
4. for $j$ from 1 to $s - 1$.
5.         $b \leftarrow b^2 \pmod{n}$.
6.         if $b = n - 1$ output TRUE.
7. output FALSE.
```

$\langle 1 \rangle$ We want n on the form $n = 1 + 2^s t$ where t is odd. This is done by finding s where $2^s \parallel n - 1$.
$\langle 2 \rangle$ and $\langle 3 \rangle$ If n fulfills the condition in part 1 of (4) we know that n is a strong pseudoprime base a, so TRUE is returned.
$\langle 4 \rangle$ The *for* loop iterates $s - 1$ times, since it checks part 2 of the condition in (4) on rows $\langle 5 \rangle$ and $\langle 6 \rangle$. As before, TRUE will be returned if n is found to be a strong pseudoprime base a.
$\langle 7 \rangle$ If n is not a strong pseudoprime in the given base a, it is of course not a prime, and FALSE is returned.

Algorithm 4.13 (Miller primality test [CP01]).
Input $n > 3$.

```
1. $W \leftarrow min\{\lfloor 2ln^2 n \rfloor, n - 1\}$
2. for $a$ from 2 to $W$
3.         call algorithm 4.12 with $n$ and $a$.
4.         If FALSE is returned, output NO.
5. output YES.
```

$\langle 1 \rangle$ W is the witness bound and $\langle 2 \rangle$ the *for* loop will $\langle 3 \rangle$ check if n is a strong pseudoprime in all of the bases from 2 to W.
$\langle 4 \rangle$ If FALSE is returned for some base a_0, we know n is not a strong pseudoprime base a_0 and then n is of course composite. NO will then be returned from this algorithm.
$\langle 5 \rangle$ If YES is returned we know from [CP01] that either

$$\begin{cases} \textit{n is prime or} \\ \textit{Extended Riemann Hypothesis is false.} \end{cases}$$

So, if Extended Riemann Hypothesis one day is proved true, we may be sure that the Miller primality test will correctly determine whether a given number is prime or not.

5 Basic idea

This chapter contains theorems and lemmas that will be important later in the thesis.

5.1 A very important theorem

Theorem 5.1.
Suppose that the numbers a and p are coprime. Then p is prime if and only if the following identity holds:

$$(x-a)^p \equiv (x^p - a) \pmod{p} \tag{5}$$

Proof 5.1. From our knowledge of binomial equations we see that

$$(x-a)^p \equiv \sum_{i=0}^{p}(-1)^i \binom{p}{i} a^{p-i}x^i \tag{6}$$

If p is prime, then for $0 < i < p$ the coefficients are zero since

$$\binom{p}{i} = \frac{p!}{i!(p-i)!}$$
$$= \frac{(p)(p-1)\cdots(i+2)(i+1)}{(p-i)(p-i-1)\cdots(2)} \tag{7}$$

and in (7) p is not a factor of the nominator. So p is a factor of (7), and

$$\binom{p}{i} = 0 \pmod{p}.$$

The sum (6) may then be written as $x^p - a^p$ and by Fermat's little Theorem (4.1) we see

$$a^p \equiv a\ a^{p-1} \equiv a \pmod{p}.$$

This leads to the conclusion that the sum (6) is congruent to $x^p - a \pmod{p}$ when p is prime.

If p is composite, we know that p has a prime divisor q. Recall the sum (6).

$$\binom{p}{q} = \frac{p}{q}\binom{p-1}{q-1} = \frac{p}{q}\frac{(p-1)\cdots(p-q+1)}{(q-1)\cdots(2)} \tag{8}$$

We have $q^k \parallel p$, so q^k is a factor if the numerator in (8). But since q is a factor of the nominator, we see that q^k is divided by q one time. So $p|\binom{p}{q}$ only if $q|(p-1)\cdots(p-q+1)$. But since $q|p$ we know that q does not divide $p-i$ for $1 \leq i \leq q-1$. Then q^k is no longer a factor of (8). So $\binom{p}{q}$ is not a multiple of p and then

$$\binom{p}{q} \neq 0 \pmod{p}.$$

Since p is coprime to a it is of course also coprime to a^{p-q} and then

$$a^{p-q} \neq 0 \pmod{p}.$$

This leads us to the conclusion that

$$\binom{p}{q} a^{p-q} \not\equiv 0 \pmod{p}$$

which implies $(x-a)^p \not\equiv (x^p - a)$ over the field $\mathbb{F}_p$, when p is composite. □

5.2 Basic idea behind the AKS algorithm

If we are given a number p to investigate, we may choose a polynomial $P(x) = x - a$ and see if congruence (5) is satisfied with our given p. If it is, we know that p is prime. And if it is not, we know that p is composite. But checking this has complexity $\mathcal{O}(p)$ which is exponential, and therefore not fast enough.
To be able to do this faster we will evaluate congruence (5) modulo a polynomial on the form $x^r - 1$.

$$(x-a)^p \equiv (x^p - a) \pmod{x^r - 1, p} \tag{9}$$

It is obvious from congruence (5) that congruence (9) holds if p is prime. But there may also be some composites for which congruence (9) holds.
Congruence (9) may be computed with Fast Fourier Multiplication [Stan] and then has time complexity $\mathcal{O}(r \log^2 p)$.
The AKS algorithm first chooses an r that satisfies

$$\begin{cases} r \textit{ is a prime.} \\ r \textit{ is somewhere about the size } \mathcal{O}(\log^6 p). \\ r - 1 \textit{ has at least one prime factor } q \textit{ such that } q > r^{\frac{1}{2}+\delta} \textit{ for a } \delta > 0 \end{cases} \tag{10}$$

Reading [Fou85] and [BH96] assures us such an r exists. In the proof of Lemma 7.1 I will prove that the algorithm will find such an r.

Now, the AKS algorithm checks the congruence for a small number of a:s. In fact, we only need to check $\mathcal{O}(\sqrt{r} \log^2 p)$ a:s to determine whether p is prime or composite.
In chapter 7 I will show the proof, and we will then see that this idea works, which means that the algorithm determines whether p is a prime or not in polynomial time.

5.3 Three important lemmas

In the remainder of this thesis the following notations will be used:
1. $\mathbb{F}_{p^d}$ is the finite field of order p^d, where p is a prime.
2. $h(x) = \frac{x^r-1}{x-1}$.
3. $\log n$ denotes $\log_2 n$.
4. $P(n)$ denotes the largest prime factor of n.
5. $\pi(n) = |\{\ i \mid i \textit{ is prime and } i \leq n\ \}|$.
6. If s is the smallest integer solution to $n^s \equiv 1 \pmod r$ we say that s is the order of $n \pmod r$, and is denoted $o_r(n)$. I will now introduce three important lemmas. They are used later in the proof, but it is already here important to have seen them. The first lemma is purely algebraic.

Lemma 5.2.
Let p and r be primes, $p \neq r$. Then:

1. *Let $t > 0$. The multiplicative group $\mathbb{F}^*_{p^t}$ of a field $\mathbb{F}_{p^t}$ is a cyclic group.*
2. *Let f(x) be a polynomial with integer coefficients. Then*
$$f(x)^p \equiv f(x^p) \pmod p$$
3. *Let $q(x)$ be any factor of $x^r - 1$ and $m \equiv m_r \pmod r$. Then*
$$x^m \equiv x^{m_r} \pmod{q(x)} \tag{11}$$
4. *In $\mathbb{F}_p$, $h(x)$ factors into irreducible polynomials, each of degree $o_r(p)$.*

Proof 5.2. Since the lemma was divided into four parts, so is also the proof.
 1. See [LN86].
 2. Let
$$f(x) = \sum_{i=0}^{d} a_i x^i$$
Then the coefficient for x^i in $f(x)^p$ is
$$\sum_{\substack{i_0+\ldots+i_d=p \\ i_1+2i_2+\ldots+di_d=i}} a_0^{i_0}\cdots a_d^{i_d} \frac{p!}{i_0!\cdots i_d!}$$
Now we see that the sum is divisible by p, unless $i_j = p$ for some j, $0 \leq j \leq d$. If so, $i = pj$ and the coefficient of x^i is $a_j^p \equiv a_j \pmod p$, which gives us the congruence $f(x)^p \equiv f(x^p) \pmod p$
 3. The proof for part three is purely algebraic. Recall that we denoted $m \equiv m_r \pmod r$. For some k we have $m = kr + m_r$ and
$$x^r - 1 \equiv 0 \pmod{x^r - 1}$$
$$\Rightarrow x^r \equiv 1 \pmod{x^r - 1}$$
$$\Rightarrow x^{kr} \equiv 1^k \equiv 1 \pmod{x^r - 1}$$
$$\Rightarrow x^{kr+m_r} \equiv x^{m_r} \pmod{x^r - 1}$$

$$\Rightarrow x^m \equiv x^{m_r} \pmod{q(x)}$$

4. Let $d = o_r(p)$ and suppose $\frac{x^r-1}{x-1}$ has an irreducible factor, say $h(x)$, in $\mathbb{F}_p$ and $h(x)$ is a polynomial of degree k. Now $\mathbb{F}_p[x]/h(x)$ forms a field of size p^k. The multiplicative subgroup of $\mathbb{F}_p[x]/h(x)$ is cyclic, and has a generator $g(x)$. Also, in the Galois field (see [Fra99] for a definition of Galois fields) we have from part 2 of this lemma:

$$g(x)^p \equiv g(x^p)$$
$$\Rightarrow g(x)^{p^d} \equiv g(x^{p^d})$$
$$\Rightarrow g(x)^{p^d} \equiv g(x)\ (\mathit{from}\ (11))$$
$$\Rightarrow g(x)^{p^d-1} \equiv 1$$

We know that $p^k - 1$ is the order of $g(x)$, and we get

$$p^k - 1 | p^d - 1 \Rightarrow k|d$$

Since $h(x)$ is a factor of $\frac{x^r-1}{x-1}$ we have

$$h(x)|x^r - 1$$

in the field $\mathbb{F}_p$. We also, in $\mathbb{F}_p[x]/h(x)$, have

$$x^r \equiv 1.$$

So r must be the order of x, since r is prime, and

$$r|p^k - 1 \Rightarrow p^k \equiv 1 \pmod{r} \Rightarrow d|k.$$

This gives us

$$k|d\ \mathit{and}\ d|k \Rightarrow k = d$$

and we have proved the lemma. □

The previous lemma was purely algebraic, but the following two comes from number theory.

Lemma 5.3.
There exists constants $c > 0$ and n_1 such that, for all $x \geq n_1$

$$|\{\ p \mid p\ \mathit{is\ prime},\ p \leq x\ \mathit{and}\ P(p-1) > x^{\frac{2}{3}}\}| \geq c\frac{x}{\log x}$$

Proof 5.3. For the proof and more information about this lemma, read [BH96]. □

Lemma 5.4.
For $n \geq 1$:

$$\frac{n}{6\log n} \leq \pi(n) \leq \frac{8n}{\log n}$$

Proof 5.4. For the proof, see [Apo97]. □

6 The AKS Algorithm

In this chapter I present and explain the algorithm. In the appendix of this thesis, there is also an example available of how to implement the algorithm in Maple 5.

6.1 The AKS Algorithm.

Input : integer $n > 1$

```
1. if $n$ is a perfect power, output COMPOSITE.
2. $r \leftarrow 2$.
3. $while(r < n)$
4.         if ($gcd(n, r) \neq 1$ ), output COMPOSITE.
5.         if ($r$ is prime)
6.                 $q \leftarrow P(r-1)$
7.                 if ($q \geq 4\sqrt{r}\log n$ and $n^{\frac{r-1}{q}} \neq 1 \pmod r$)
8.                         break $while$ loop and go to step 11.
9.         else $r \leftarrow r+1$ and return to step 3.
10. end $while$ loop.
11. $for\ a = 1$ to $2\sqrt{r}\log n$
12.         if ($(x-a)^n \neq (x^n - a) \pmod{x^r - 1, n}$), output COMPOSITE.
13. output PRIME.
```

6.2 Description of the AKS Algorithm

I will now explain how the algorithm works. For some users this part might be interesting, while some others might already be confident in reading algorithms. I will not introduce any new facts in the remainder of this chapter, I will only develop what has already been stated.

First note that the algorithm is based on a beginning, two loops and an ending.

In the beginning we give the algorithm the number we want to investigate as an input n. $\langle 1 \rangle$ If n is a perfect power, it is of course not a prime. This may be checked rather quickly, as seen later in chapter 8. $\langle 2 \rangle$ A variable, r, is created. It is given the value 2, but will most likely be modified later in the loops.

$\langle 3 \rangle$ In the first loop the algorithm looks for the "suitable r". $\langle 4 \rangle$ If any number investigated divides n, n is of course composite. Otherwise we continue. $\langle 5 \rangle$ The r we want has to be a prime. $\langle 6 \rangle$ We create a variable q, and it is given the value of the largest prime factor of $r-1$. $\langle 7 \rangle$ If an r is found, that suites our conditions $\langle 8 \rangle$ we break the first loop and go to $\langle 11 \rangle$. $\langle 9 \rangle$ Otherwise we give r the value $r+1$, and start the loop over again from $\langle 3 \rangle$.

$\langle 11 \rangle$ When the r is found, the algorithm checks the modified congruence from Fermat's little theorem for a number of a:s. $\langle 12 \rangle$ If the congruence fails for any a, n is composite.

$\langle 13 \rangle$ And finally, if n is not on the form a^b, the r is found and the congruence holds, our n is indeed a prime.

Recall the definition of a witness for a number. In both the *while* and the *for* loop the algorithm is looking for witnesses. If a witness is found, as before, the number is composite. But if no witness is found, neither in the *while* nor in the *for* loop, the input number is prime. I will prove this in the next chapter.

7 The correctness of the AKS algorithm

7.1 Introduction

In this chapter I will prove that the algorithm works, that it determines whether a number is prime or not correctly.

Recall from the last chapter that the algorithm is based on two loops. The first loop searches for the "suitable r", and will find it in an interval, as in the following lemma:

Lemma 7.1.
Given n there exists two constants independent of n, c_1 and c_2, and a prime r such that

$$\begin{cases} c_1 \log^6 n \le r \le c_2 \log^6 n \textit{ and} \\ r-1 \textit{ has a prime factor } q \ge 4\sqrt{r}\log n \textit{ and} \\ q | o_r(n) \end{cases}$$

Proof 7.1. Recall the definition of $P(n)$ and the c from Lemma 5.3. For large n the number of prime r:s (we call them special primes) such that

$$c_1 \log^6 n \le r \le c_2 \log^6 n$$

and

$$P(r-1) > (c_2 \log^6 n)^{\frac{2}{3}} > r^{\frac{2}{3}}$$

is

$$\ge \textit{Number of special primes in } [1 \ldots c_2 \log^6 n]$$
$$- \textit{ Number of primes in } [1 \ldots c_1 \log^6 n]$$

and from Lemma 5.4 we get

$$\ge \frac{cc_2 \log^6 n}{7 \log\log n} - \frac{8c_1 \log^6 n}{6 \log\log n}$$

and with some algebraic rewriting we have

$$= \frac{\log^6 n}{\log\log n}\left(\frac{cc_2}{7} - \frac{8c_1}{6}\right)$$

We choose $c_1 \ge 4^6$ and c_2 such that $\frac{cc_2}{7} - \frac{8c_1}{6}$ is positive.
Now to the most important part of the proof. We write $c_2 \log^6 n = x$ and take a closer look at the product

$$\prod (n-1)(n^2-1)\cdots(n^{x^{\frac{1}{3}}}-1)$$

We know that any number n has at most $\log n$ prime factors. This leads us to the conclusion that the product above has atmost

$$\sum_{i=1}^{x^{\frac{1}{3}}} \log n^i$$

$$= \sum_{i=1}^{x^{\frac{1}{3}}} i(\log n)$$

$$\begin{aligned}
&= (\log n)\sum_{i=1}^{x^{\frac{1}{3}}} i \\
&= (\log n)(x^{\frac{1}{3}})(x^{\frac{1}{3}}+1)/2 \\
&= (\log n)(x^{\frac{2}{3}}+x^{\frac{1}{3}})/2 \\
&\leq x^{\frac{2}{3}}\log n
\end{aligned}$$

prime factors.
Now we see

$$x^{\frac{2}{3}}\log n = (c_2\log^6 n)^{\frac{2}{3}}\log n < \frac{\log^6 n}{\log\log n}\left(\frac{cc_2}{7}-\frac{8c_1}{6}\right).$$

This means that there are more prime r:s than prime factors in the product. And therefore one of the r:s will not divide the product. This r has the following properties:

$$\begin{cases} r-1 \textit{ has a large prime factor } q \geq r^{\frac{2}{3}} \geq 4\sqrt{r}\log n \textit{ and} \\ q|o_r(n) \end{cases}$$

□

By this lemma we can be sure that the *while* loop will find the wanted r. This means that the loop will halt. Let's now prove that the algorithm detects primes correctly.

7.2 The algorithm and primes

Theorem 7.2 (AKS and prime input).
The AKS algorithm returns PRIME if input n is prime.

***Proof** 7.2.* We see that $gcd(n,r) = 1$ for all $r < n$. And since we proved in Lemma 5.2, part 2, that the congruence in the *for* loop holds for all primes, the algorithm will return PRIME if input n is prime. □

7.3 The algorithm and composites

Theorem 7.3 (AKS and composite input).
The AKS algorithm returns COMPOSITE if input n is composite.

If a composite n is the input to the algorithm, the *while* loop will of course begin to look for the r. If the r is found before any divisor of n is detected, the algorithm will continue.
Now our n has the prime factors p_i where $1 \leq i \leq k$ for some k. Then we have

$$o_r(n)|lcm_i\{o_r(p_i)\}$$

and therefore there exist a prime factor p such that

$$q|o_r(p)$$

where q is as before, the largest prime factor of $r-1$.

In the *for* loop the found r is used when checking the congruences. The algorithm checks $l = 2\sqrt{r}\log n$ a:s in $(x-a)^n \pmod n$.

From Lemma 5.2, part 4, we have $h(x)$ of degree $d = o_r(p)$ and

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, n}$$

$$\Rightarrow (x-a)^n \equiv (x^n - a) \pmod{h(x), n}$$

The binomials computed form a cyclic group, in the field $\mathbb{F}_p[x]/h(x)$. The importance of the following lemma is to get an idea how large that group is.

Lemma 7.4.
In the field $\mathbb{F}_p[x]/h(x)$, the group

$$G = \{ \prod_{1\le a\le l} (x-a)^{\alpha_a} | \alpha_a \ge 0, \forall 1 \le a \le l\}$$

is cyclic, and of size $> (\frac{d}{l})^l$.

Proof 7.4. Since G is a subgroup of a cyclic group, $(\mathbb{F}_p[x]/h(x))^*$, it is of course cyclic.
Now we look at a subgroup of G, let's call it S_G.

$$S_G = \{ \prod_{1\le a\le l} (x-a)^{\alpha_a} | \sum_{1\le a\le l} \alpha_a \le d-1, \forall 1 \le a \le l\}.$$

All the elements in this group are distinct in the field $\mathbb{F}_p[x]/h(x)$. When the *while* loop halts, the found r is $r > q \ge 4\sqrt{r}\log n > 2\sqrt{r}\log n = l$. Step 4 in the algorithm checks $gcd(n,r)$. And if any of the a:s are congruent $\pmod p$, and $p < l < r$, step 4 in the algorithm would have identified n as composite already. So for any two elements $a_1, a_2 \in S_G$

$$a_1 \not\equiv a_2 \pmod p.$$

The cardinality of S_G is:

$$\binom{l+d-1}{l} = \frac{(l+d-1)(l+d-2)\cdots(d)}{l!}$$

and since

$$(l+d-1)(l+d-2)\cdots(d) = (l-1)(l-2)\cdots(1)(d)^l > d^l$$

and

$$l! < l^l$$

we have

$$\binom{l+d-1}{l} > \left(\frac{d}{l}\right)^l.$$

Since S_G is a subgroup of G

$$|G| \ge |S_G| > \left(\frac{d}{l}\right)^l$$

and the lemma follows. □

From the fact that $d \geq 2l$, and

$$l = 2\sqrt{r}\log n \Rightarrow 2^l = 2^{2\sqrt{r}\log n} = n^{2\sqrt{r}}$$

we have

$$|G| > 2^l = n^{2\sqrt{r}}.$$

Let $g(x)$ be a generator of G. Then the order of $g(x)$ in the field $\mathbb{F}_p[x]/h(x)$ is $> n^{2\sqrt{r}}$.
Now we define the set

$$I_{g(x)} = \{m | g(x)^m \equiv g(x^m) \pmod{x^r - 1, p}\}$$

and we have an interesting lemma:

Lemma 7.5.
The set $I_{g(x)}$ is closed under multiplication. That is

$$m_1, m_2 \in I_{g(x)} \Rightarrow m_1 m_2 \in I_{g(x)}$$

The proof uses algebraic calculations, based on Lemma 5.2, part 2.

Proof 7.5. Let m_1, $m_2 \in I_{g(x)}$. Then we have

$$g(x)^{m_1} \equiv g(x^{m_1}) \pmod{x^r - 1, p},$$

$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p}.$$

If we substitute x with x^{m_1} we see

$$g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^{m_1 r} - 1, p},$$

$$\Rightarrow g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p}.$$

and we have

$$g(x)^{m_1 m_2} \equiv (g(x)^{m_1})^{m_2} \equiv g(x^{m_1})^{m_2} \equiv g(x^{m_1 m_2}) \pmod{x^r - 1, p}.$$

and the lemma follows. □

Before we prove Theorem 7.3 we need one last lemma. In the remainder of this thesis, we let o_g denote the order of $g(x)$ in $\mathbb{F}_p[x]/h(x)$.

Lemma 7.6.
Let $m_1, m_2 \in I_{g(x)}$. Then

$$m_1 \equiv m_2 \pmod{r} \Rightarrow m_1 \equiv m_2 \pmod{o_g}$$

Proof 7.6. We have $m_2 = m_1 + kr$ for some $k \geq 0$ and since we said that $m_2 \in I_{g(x)}$ we have

$$g(x)^{m_2} \equiv g(x^{m_2}) \pmod{x^r - 1, p}$$

$$\Rightarrow g(x)^{m_2} \equiv g(x^{m_2}) \ (\textit{in the field } \mathbb{F}_p[x]/h(x))$$

$$\Rightarrow g(x)^{m_1 + kr} \equiv g(x^{m_1 + kr}) \ (\textit{in } \mathbb{F}_p[x]/h(x))$$

$$\Rightarrow g(x)^{m_1} g(x)^{kr} \equiv g(x^{m_1}) \ (\textit{from Lemma } 5.2, \textit{ part } 3)$$

Since $g(x)$ is the generator, it is obvious that $g(x) \neq 0$ which implies $g(x)^{m_1} \neq 0$. Then we may cancel $g(x)^{m_1}$ from both sides of the congruence and we get

$$g(x)^{kr} \equiv 1$$

And since o_g is the order of $g(x)$, we have

$$g(x)^{o_g} \equiv 1$$

and therefore kr is a multiple of o_g. This implies

$$kr \equiv 0 \pmod{o_g} \Rightarrow m_2 \equiv m_1 \pmod{o_g}$$

and the lemma follows. □

The lemma implies that

$$|\{m \in I_{g(x)} | m < o_g\}| \leq r.$$

We are now ready to prove Theorem 7.3

Proof 7.3. We have two options for the output from the algorithm, PRIME or COMPOSITE. Let's suppose the algorithm returns PRIME for a composite number.
The *for* loop ensures us that for the a:s, $1 \leq a \leq 2\sqrt{r}\log n$, the congruence

$$(x-a)^n \equiv (x^n - a) \pmod{x^r - 1, p} \tag{12}$$

holds. Recall the generator $g(x)$ for the cyclic group G. $g(x)$ is here a product of l binomials $(x-a)$, where $1 \leq a \leq l$, and for which all binomials satisfy congruence (12). This means that

$$g(x)^n \equiv g(x^n) \pmod{x^r - 1, p}$$

So, $n \in I_{g(x)}$ and from Lemma 5.2, part 2, we have a prime $p \in I_{g(x)}$.
Now we create a group of products of our input n and the prime p

$$E = \{n^i p^j | 0 \leq i, j \leq \lfloor\sqrt{r}\rfloor\}$$

Since $n, p \in I_{g(x)}$ and from Lemma 7.5 we know that $I_{g(x)}$ is closed under multiplication, we see that $E \subseteq I_{g(x)}$. By inspection we see that the size of E is

$$|E| = (1 + \lfloor\sqrt{r}\rfloor)^2.$$

And since

$$(1 + \lfloor\sqrt{r}\rfloor)^2 > r$$

we see that for some $n^{i_1}p^{j_1}, n^{i_2}p^{j_2} \in E$ where $i_1 \neq i_2$ or $j_1 \neq j_2$ we have

$$n^{i_1}p^{j_1} \equiv n^{i_2}p^{j_2} \pmod{r} \ (\textit{by pigeon hole principle})$$

$$\Rightarrow n^{i_1}p^{j_1} \equiv n^{i_2}p^{j_2} \pmod{o_g}$$

$$\Rightarrow n^{i_1 - i_2} \equiv p^{j_1 - j_2} \pmod{o_g} \tag{13}$$

Recall that the order of $g(x)$, $o_g \geq n^{2\sqrt{r}}$. And we also have $n^{|i_1-i_2|}$, $p^{|j_1-j_2|} < n^{\sqrt{r}}$. So, congruence (13) is an equality, and since p is a prime we have

$$n = p^b, b \geq 1.$$

But in the beginning of the algorithm we already checked all possible solutions to $b \geq 2$, so $b = 1$. But if $b = 1$, we have $n = p$, so n must be a prime. This contradicts our assumption in the beginning of this proof. So the algorithm will return COMPOSITE when input n is composite. □

7.4 The algorithm detects primes correctly

From the last two theorems, we come to the conclusion that the the following theorem must hold:

Theorem 7.7 (The AKS algorithm works).
The AKS algorithm returns PRIME if and only if input n is prime.

***Proof** 7.7.* From Theorem 7.2 and 7.3, the proof follows. □

8 Time complexity analysis

Definition 8.1 (Asymptotic Time Complexity, [nist]).
Asymptotic time complexity is the limiting behavior of the execution time of an algorithm when the size of the problem goes to infinity.

Definition 8.2.
$\tilde{\mathcal{O}}(t(n))$ denotes $\mathcal{O}(t(n)poly(\log t(n)))$ where $t(n)$ is some function of n. Here $poly(t(n))$ denotes a polynomial of some function $t(n)$.

Time complexity analysis is a way to get an idea of how fast an algorithm works. It is not an exact expression, but when the problem grows large it is an efficient way to compare the efficiency of different algorithms.

The AKS is a polynomial time algorithm. It is, so far, the algorithm with lowest known complexity that determines whether a given number is prime or not. In this chapter I will prove that the algorithm has time complexity $\tilde{\mathcal{O}}(\log^{12} n)$.

8.1 Original algorithm complexity

Theorem 8.3.
The AKS algorithm has the asymptotic time complexity of $\tilde{\mathcal{O}}(\log^{12} n)$

Proof 8.3. In this proof I will use the numbers for the rows in the algorithm at page 14. So $\langle i \rangle$ is a reference to the i:th row in the algorithm.

In $\langle 1 \rangle$ the algorithm checks if input n is a perfect power. According to Daniel J. Bernstein [Ber98] detecting whether a given number is a perfect power or not can be done with complexity $\mathcal{O}(\log n)$.

We know from (10) at page 11 r is somewhere about the size $\mathcal{O}(\log^6 n)$. So the *while* loop, starting at $\langle 3 \rangle$, makes $\mathcal{O}(\log^6 n)$ iterations.

In the *while* loop, step $\langle 4 \rangle$ where *gcd* is computed has complexity $\mathcal{O}(poly(\log\log r))$. $\langle 5 \rangle$ and $\langle 6 \rangle$ takes $\sqrt{r}poly(\log\log n)$ if brute-force, a rather slow algorithm, is used. $\langle 7 \rangle$, $\langle 8 \rangle$ and $\langle 9 \rangle$ takes at most $poly(\log\log n)$ steps, and we have the total complexity

$$\tilde{\mathcal{O}}(r^{\frac{1}{2}} \log^6 n)$$

for the *while* loop. Since

$$r = \mathcal{O}(\log^6 n)$$

we see

$$\tilde{\mathcal{O}}(r^{\frac{1}{2}} \log^6 n) = \tilde{\mathcal{O}}(\log^9 n)$$

In $\langle 12 \rangle$ polynomials are computed, and if we use repeated-squaring and Fast Fourier multiplication we have complexity $\tilde{\mathcal{O}}(r \log^2 n)$. It is obvious from row $\langle 11 \rangle$ that the *for* loop makes $2\sqrt{r}\log n$ iterations. Thus, the complexity for rows $\langle 11 \rangle$ and $\langle 12 \rangle$ is

$$\tilde{\mathcal{O}}(r^{\frac{3}{2}} \log^3 n) = \tilde{\mathcal{O}}(\log^{12} n)$$

From this we see that the total complexity for the algorithm is

$$\tilde{\mathcal{O}}(\log^{12} n)$$

□

8.2 Experimental complexity analysis

We know that if the input n is composite, the algorithm will stop in one of the loops. Therefore the algorithm will have lower run time for composite numbers. Therefore I have, in this experimental complexity analysis, checked primes and composites seperately.

Since the first part of my implementation (as presented in the appendix) is not based on Bernsteins algorithm, I will in this analysis not check if the input is on the form a^b. Of course I will then in this analysis only use numbers that surely are not on the form a^b.

First we check the following composite and prime numbers to get their run time. Note that $R(n)$ here denotes the run time, in seconds, for n.

Composite n	$R(n)$	Prime n	$R(n)$
102	0.100	101	0.417
202	0.100	211	0.616
303	0.233	307	1.034
404	0.333	401	1.616
505	0.484	503	2.450
606	0.650	601	3.284
707	0.816	701	4.500
808	1.050	809	5.883
909	1.234	907	7.733

On pages 24-26 we have six plots of $R(n)$ and $\log^{12} n$. In Figure 1 we have $R(n)$ for the composite n:s and in Figure 2 we have $R(n)$ for the prime numbers.

In Figure 3 we have the plot of $\log^{12} n$. In Figure 4 I have put the previous three plots together and we now see that $R(n)$ follows $\log^{12} n$.

When studying the plot of $\frac{R(n)}{\log^{12} n}$ in Figures 5 and 6 we see how the quotient converges to a constant. This leads us to the guess that $R(n)$ is bounded by $\log^{12} n$.

Note that I have removed the scale on the y-axis in Figure 4, since $R(n)$ is bounded by, but not identical to $\log^{12} n$.

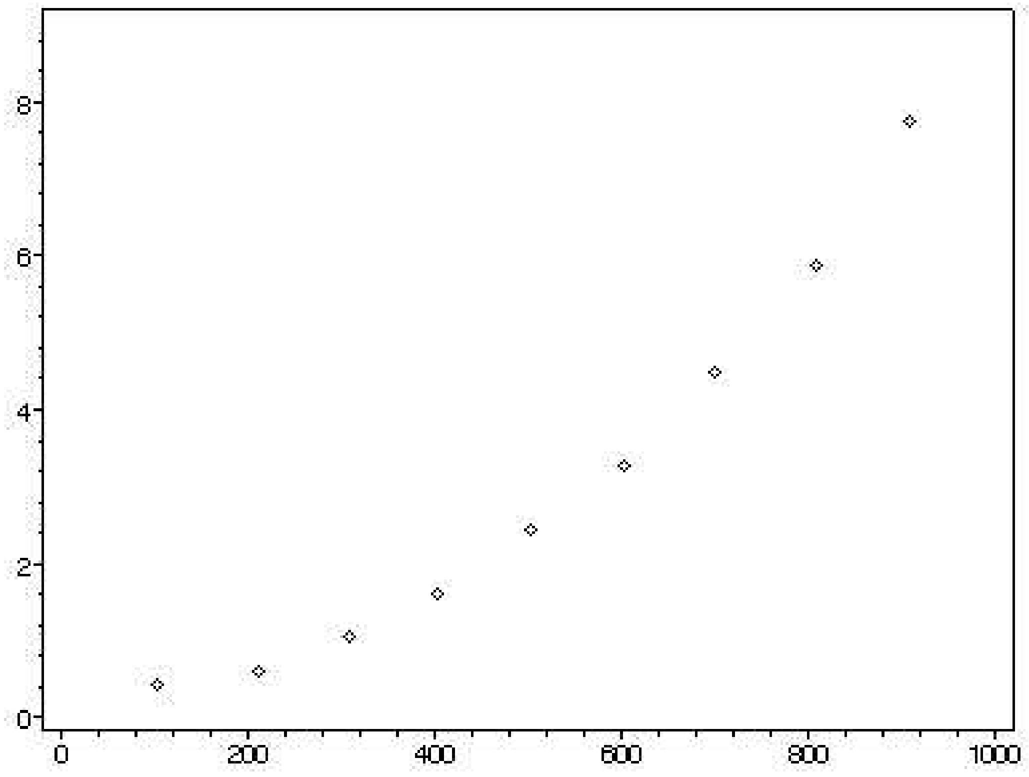

Figure 1: $R(n)$ for composite n:s.

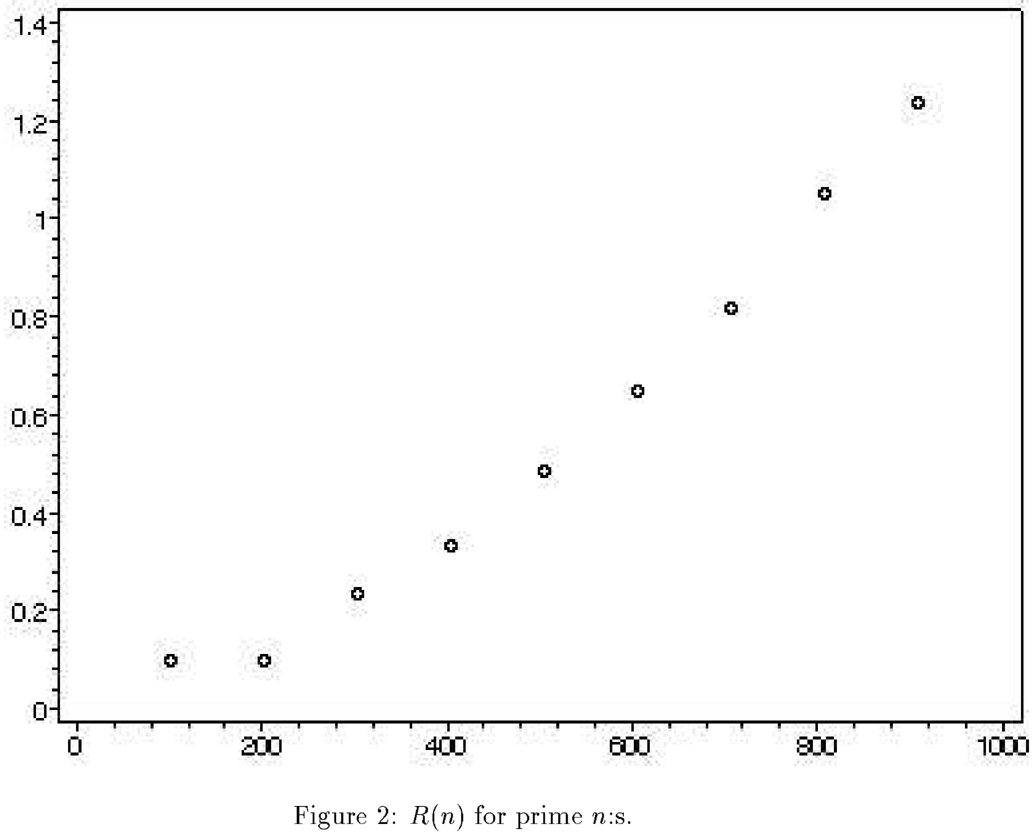

Figure 2: $R(n)$ for prime n:s.

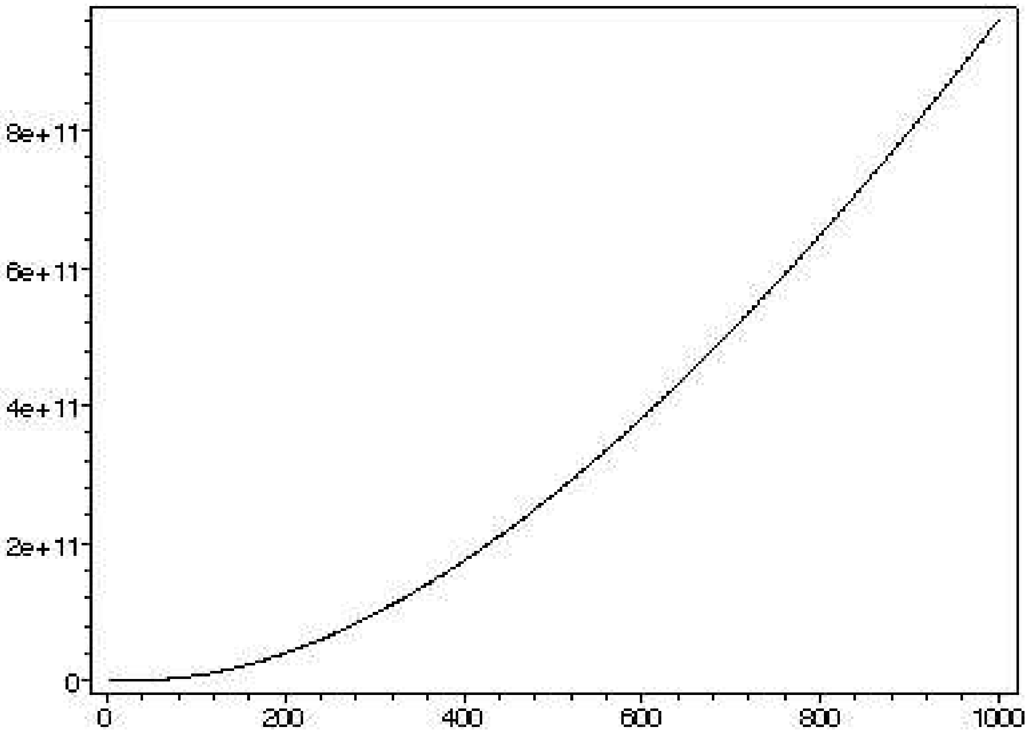

Figure 3: $\log^{12} n$.

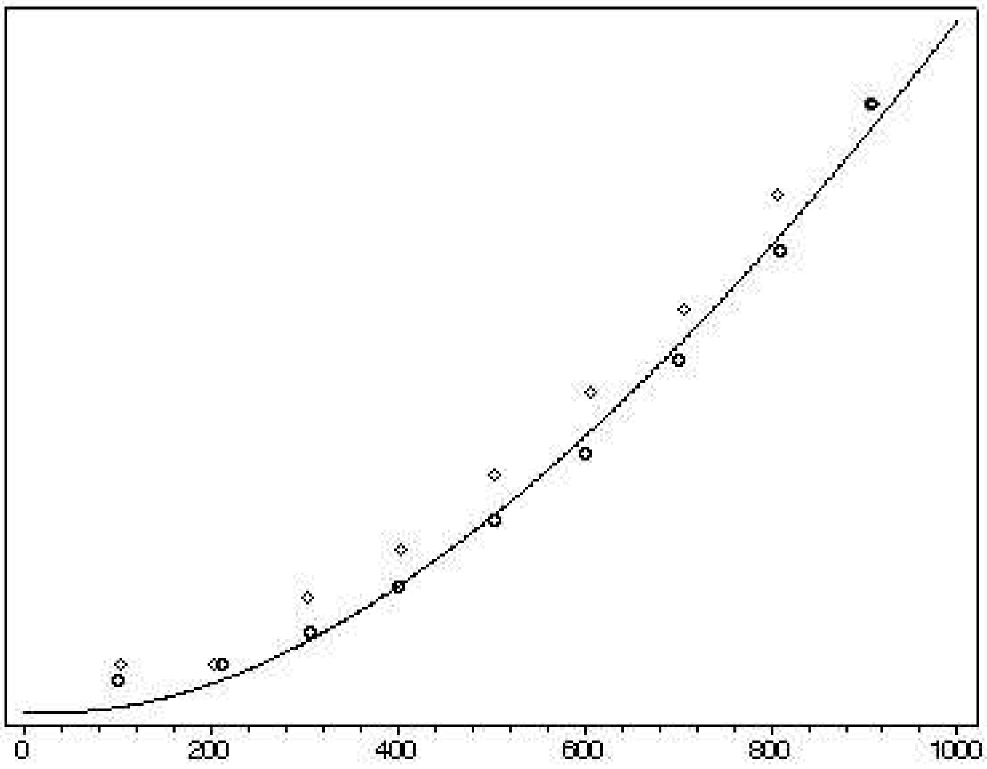

Figure 4: $\log^{12} n$ and $R(n)$ for primes ($\circ$) and composites ($\diamond$).

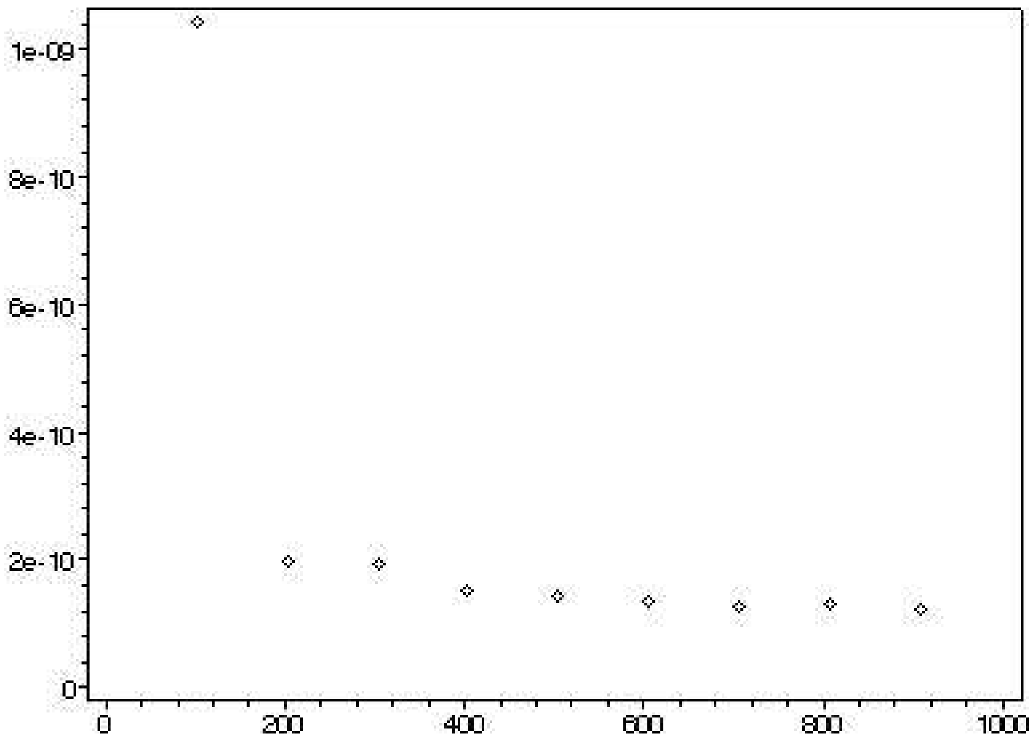

Figure 5: $\frac{R(n)}{\log^{12} n}$ for composite n:s.

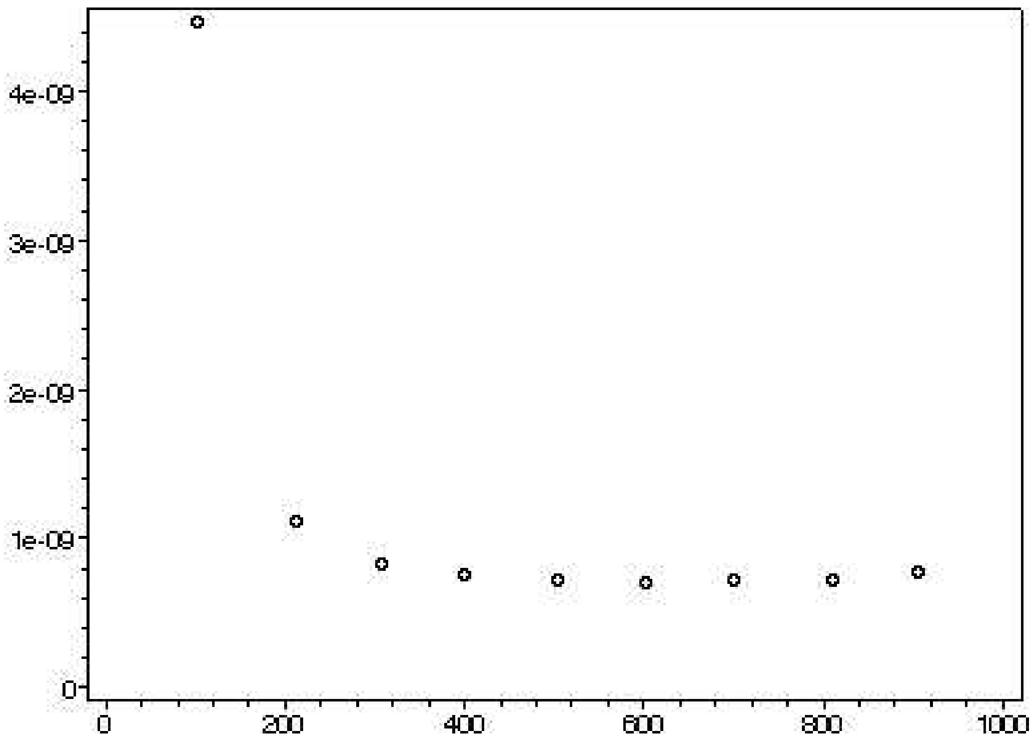

Figure 6: $\frac{R(n)}{\log^{12} n}$ for prime n:s.

8.3 Improvements by Lenstra

The AKS algorithm may still be improved to work even faster. On the internet the algorithm and possible improvements are discussed, and on the homepage [Phil] most of the improvements are presented.

For example Henrik W. Lenstra has found a way to decrease some constants in the run time [Ber02].

In the AKS algorithm we looked for a small prime r, coprime to n, such that

1. $n^{(r-1)/q} \not\equiv 1 \pmod r$
2. $\binom{q+s-1}{s} \geq n^{\lfloor r \rfloor}$

where s is a large integer. But if we change the conditions for the r, the algorithm is conjectured to work roughly 60 000 times faster.

The new conditions for the r are presented in the following theorem.

Theorem 8.4 (AKS and Lenstra's theorem [Ber02]).
Let n and r be positive integers and let S be a finite set of integers. Assume that:

1. *n is not a perfect power.*
2. *n is a primitive root $\pmod r$.*
3. *$gcd(n, a_1 - a_2) = 1$ for all distinct $a_1, a_2 \in S$.*
4. *$\binom{\varphi(r)+|S|-1}{|S|} \geq n^{\lfloor r \rfloor}$.*
5. *$(x-a)^n \equiv x^n - a \pmod{x^r - 1, n} \forall a \in S$.*

Then n is a prime.

In the *for* loop in the AKS algorithm we checked $2\sqrt{r}\log n$ a:s. Also when looking for our r, we wanted the largest prime factor of $r-1$ to be $q \geq 4\sqrt{r}\log n$. These conditions are, from Theorem 8.4 unnecessarily strong.

A suitable r is conjectured to exist on the scale of $\log^2 n$, with $q = \frac{r-1}{2}$. The time spent checking the congruence from Theorem 8.4, condition 5, is $\mathcal{O}(rs\log^2 n)$.

Now to the interesting part of Lenstra's improvements. The minimum value of $rs\log^2 n$ is conjectured, from Theorem 8.4, to be $(0.0017\ldots + o(1))(\log^6 n)$. But in the original AKS algorithm, $rs\log^2 n$ is conjectured to be $(1024 + o(1))(\log^6 n)$.

So, the condition that $q \geq 4\sqrt{r}\log n$ would then no longer be necessary. It would also be unnecessary to check all the $2\sqrt{r}\log n$ a:s in the *for* loop.

This means that a slightly modified algorithm will work roughly 60 000 times faster. In theory it will still have time complexity $\widetilde{\mathcal{O}}(\log^{12} n)$, but in practice we may save some time.

8.4 Possible future improvements

We can prove that the algorithm has asymptotic time complexity $\widetilde{\mathcal{O}}(\log^{12} n)$. But most likely, the algorithm is even faster since a conjecture tells us that the algorithm will find a smaller suitable r. If so, it will have complexity $\widetilde{\mathcal{O}}(\log^6 n)$.

Conjecture 8.5 (co-Sophie Germain prime distribution [HL22]).
The number of co-Sophie Germain primes in $[1 \ldots x]$ *is* $\mathcal{O}(\frac{Dx}{\log x})$*, where* D *is the twin prime constant (approximately 0.6601618158.. according to [PG])*

If this conjecture some day is proved true, we can be sure that the *while* loop in the algorithm will find a suitable r of size $\mathcal{O}(\log^2 n)$.
If the reader is interested in details of the complexity analysis under these new conditions, I recommend [AKS02].

Since the conjecture has been verified for r:s up to 10^{10} we may assume that the r is most likely smaller than we have assumed earlier. If so, the algorithm has complexity

$$\widetilde{\mathcal{O}}(\sqrt{r}\log^2 n) \text{ (\textit{the while loop})} + \widetilde{\mathcal{O}}(r^{\frac{3}{2}}\log^3 n) \text{ (\textit{the for loop})} = \widetilde{\mathcal{O}}(\log^6 n).$$

This means that it is, most likely, possible to improve the *while* loop to work even faster.

Is it also possible to improve the *for* loop? Well, it might be.

As we stated in chapter 6, the *for* loop iterates $2\sqrt{r}\log n$ times to generate a group G large enough. The upper limit for a, and then also the complexity, may be decreased if a smaller number of polynomials $(x-a)^n$ generates a group large enough.

The following conjecture was presented in [AKS02].

Conjecture 8.6. *[BP01]*
If r *does not divide* n*, and if*

$$(x-1)^n \equiv (x^n - 1) \pmod{x^r - 1, n} \tag{14}$$

then either n *is prime or* $n^2 \equiv 1 \pmod r$.

Note that r needs to be a prime. Otherwise the conjecture is incorrect, as seen in the following example.

Example 8.7.
33 does not divide 121 and

$$(x-1)^{121} \equiv (x^{121} - 1) \pmod{x^{33} - 1, 121}$$

and

$$121^2 \equiv 14641 \equiv 22 \pmod{33}$$

but 121 is not a prime.

If Conjecture 8.6 is true, we may modify the algorithm to first look for an r that does not divide n^2-1. The r will be found in $[2, 4\log n]$. The algorithm will then check congruence 14. If Fast Fourier Multiplication is used, the *for* loop will have complexity $\mathcal{O}(r\log^2 n)$. And then the algorithm will have complexity

$$\widetilde{\mathcal{O}}(\log^3 n).$$

A An implementation in Maple 5

Here a present a suggestion how to implement the algorithm in Maple 5. If you want to test this implementation on your computer, you may download it from my homepage http://www.josef.nu/. Click on "josef" - "my education" - "mathematics", and you will find it.

```
# APowerB determines whether n is on the form a^b, b > 1, or not
APowerB := proc(n) local ans, a, b;
ans := false;
for a from 2 by 1 while a <= n do
for b from 2 by 1 while a^b <= n do
if( a^b = n ) then ans := true fi;
od;
od;
RETURN(ans)
end :

# LastForLoop evaluates the congruence
# (x - a)^n = x^n - a mod n

LastForLoop := proc(n, r) local a, ans2;

ans2 := true;

for a from 1 by 1 to floor(2 * sqrt(r) * log[2](n)) do
if(
modpol((x - a)^n, (x^r) - 1, x, n) mod n
<>
modpol((x^n) - a, (x^r) - 1, x, n) mod n
) then ans2 := false fi;
od;

RETURN(ans2);
end :

# largestPrimeFactor gives us the largest
# prime factor of a given number

largestPrimeFactor := proc(r) local qPrime, qP;

for qP from 1 by 1 to r do
if((r mod qP = 0) and (isprime(qP))) then qPrime := qP fi;
od;

RETURN(qPrime);
end :
```

```
# FindTheR gives us the r

FindTheR := proc(n, ans3) local r, q, ans4;
ans4 := ans3;

for r from 2 by 1 while (r < n) do
if((not (gcd(n, r) = 1))) then ans4 := false fi;

if(isprime(r)) then
q := largestPrimeFactor(r - 1);

#if we find the suitable r, we break the last for - loop
if ((q >= 4 * sqrt(r) * log[2](n))
and
(((n^((r - 1)/q)) mod r = 1) = false)) then break fi;
fi;

od;

RETURN(r, ans4)
end :

# josefAKS is my version of the AKS algorithm.
# it is based on the above defined functions :
# APowerB, FindTheR and LastForLoop.

#load the library "modpol"
readlib(modpol) :
#define the function with some local variables
josefAKS := proc(n) local a, b, r, answer;
#we first assume that n is prime
answer := true :
#if n is a power of a number, n is a composite
if(APowerB(n)) then answer := false; fi;
#Search for a suitable r
if(answer) then (r, answer) := FindTheR(n, answer) : fi;
#if n has still not been found to be a composite we will test the congruence
if(answer) then answer := LastForLoop(n, r) : fi :

RETURN(answer)

#to use this implementation you write
# > josefAKS(n);
#where n is the input number.
```

References

[AKS02]
Manindra Agrawal, Neeraj Kayal and Nitin Saxena.
PRIMES is in P, (2002).
http://www.cse.iitk.ac.in/news/primality.pdf.

[Ber98]
Daniel J. Bernstein.
Detecting Perfect Powers In Essentially Linear Time, (1998).
http://cr.yp.to/papers/powers-ams.pdf.

[Ber02]
Daniel J. Bernstein.
An exposition to the Agrawal-Kayal-Saxena primality-proving theorem, (2002).
http://cr.yp.to/papers/aks.pdf.

[Atr]
Mohan Atreya.
Introduction to Cryptography.
http://www.rsasecurity.com/products/bsafe/whitepapers/IntroToCrypto.pdf.

[JJ98]
Gareth A. Jones and J. Mary Jones.
Elementary Number Theory, (1998).
Chapter 2, Prime Numbers, pages 25-26.
Chapter 4, Congruence with a Prime-power Modulus, pages 70-71.

[Fra99]
John B. Fraleigh.
Abstract Algebra, (1999), sixth edition.
Introduction to Rings and Fields, page 271.
Extension Fields, page 419.

[CP01]
Richard Crandall and Carl Pomerance.
Prime Numbers - A Computational Perspective, (2001),
Analytic number theory, pages 36-39.
Probable primes and witnesses, pages 124-130.

[Apo97]
T. M. Apostol.
Introduction to Analytic Number Theory, Springer-Verlag (1997).

[BH96]
R. C. Baker and G. Harman.
The Brun-Titchmarsh Theorem on Average, (1996).
In *Proceedings of a conference in Honor of Heini Halberstam, Volume 1*, pages 39-103.

[LN86]
R. Lidl and H Niederreiter.
Introduction to finite fields and their applications, Cambridge University Press (1986).

[Fou85]
E. Fouvry.
Theoreme de Brun-Titchmarsh; Application au theoreme de Fermat, (1985).
Invent. Math., 79:383-407.

[HL22]
G. H. Hardy and J. E. Littlewood.
Some problems of "Partitio Numerorum" III; On the expression of a number as a sum of primes, (1922).
Acta Mathematica., 44:1-70.

[BP01]
Rajat Bhattacharjee and Prashant Pandey.
Primality testing. Technical report, IIT Kanpur, (2001).
http://www.cse.iitk.ac.in/research/btp2001/primality.html.

[Phil]
AKS Primality proving analyses.
http://fatphil.asdf.org/maths/AKS/.

[Stan]
The Fast Fourier Transform,
http://sepwww.stanford.edu/sep/prof/fgdp/c1/paper_html/node4.html.

[nist]
Asymptotic Time Complexity,
http://www.nist.gov/dads/HTML/asymptoticTimeComplexity.html.

[PG]
The Prime Glossary: twin prime constant,
http://primes.utm.edu/glossary/page.php?sort=TwinPrimeConstant.