An Algorithm that Decides PRIMES in Polynomial Time

Kevin Clark

June 3, 2011

Contents

1 Introduction

Prime numbers have been studied by mathematicians since the time of Pythagoras. They play an important role in many areas of mathematics, especially in number theory. One important question arising from the study of prime number is if the primality of a number can be determined efficiently. This has practical as well as theoretical value because modern cryptography relies on generating large prime numbers.

Since the advent of complexity theory, which allowed the idea of "efficiency" to be formulated precisely, this problem has been studied intensively. Primality was known to be in NP (solvable in nondeterministic polynomial time) since 1974 but it remained an open question until 2003 whether it was in P (solvable in polynomial time). A simple algorithm for deciding if n is prime called the Sieve of Erasthones tests if n can be divided by any natural number less than or equal to $\sqrt{n}$. This algorithm runs in exponential time on its input, giving rise to the question of whether a faster algorithm for deciding primes exists. A series of breakthroughs brought algorithms closer to deterministically deciding primality in polynomial time. A probabilistic algorithm by Miller and Rabin [7] relying on Fermat's Little Theorem can find prime numbers in polynomial time given a probability that the algorithm produces incorrect output. In 1983, Adleman, Pomerance, and Rumely [1] gave a deterministic algorithm that runs in $\mathcal{O}\big(\log n^{\log\log\log n}\big)$ time, which is better than exponential time (although still not polynomial). There also exists algorithms that run in polynomial time assuming the Extended Riemann Hypothesis. This paper describes the first discovered polynomial-time algorithm for deciding if a number is prime given by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena [2] (referred to in this paper as the AKS primality test) and the ideas behind it. It also gives an analysis of the run time of the algorithm which proves that it does polynomial time.

2 Definitions

The following definitions formalize the problem of testing primality in polynomial time.

Definition 2.1. PRIMES $= \{n \in \mathbb{N} \mid n$ has no divisors other than 1 and itself$\}$

Definition 2.2. P is the class of languages that are decidable in polynomial time on a deterministic single-tape Turing machine.

In order to analyze the run time of the algorithm, we use big-O notation, which provides an upper bound on how the run time asymptotically scales with the input.

Definition 2.3. A funtion $f(n)$ is $\mathcal{O}\big(t(n)\big)$ if there exist positive constants M, n_0 such that

$$|f(n)| \leq M\,|t(n)| \tag{1}$$

for all $n \geq n_0$

Note that a number of size n can be represented with $\lceil \log n \rceil$ symbols. This means to prove primes is in P we need to show PRIMES can be determined in time polynomial of $\log n$ because time complexity is measured in terms of the size of an algorithm's input (in this case the number of digits of n). That is, out algorithm's run time must be $\mathcal{O}\big(\log^k n\big)$ for some $k \in \mathbb{N}$. Using big O notation for the analysis of the algorithm can be cumbersome because the algorithm's run time is expressed in terms of $\log n$ instead of n. The following notation alleviates this somewhat.

Definition 2.4. A function $f(n)$ is $\tilde{\mathcal{O}}\big(t(n)\big)$ if

$$f(n) = \mathcal{O}\big(t(n)\big) \cdot poly(\log t(n)) \tag{2}$$

Notice that

$$\tilde{\mathcal{O}}\big(\log^k n\big) = \mathcal{O}\big(\log^k n\big) \cdot poly(\log \log^k n) = \mathcal{O}\big(\log^k n\big) \cdot poly(\log k \log n) = \mathcal{O}\big(\log^{k+\epsilon} n\big) \tag{3}$$

For all $\epsilon > 0$.This means that if an algorithm runs in time $\tilde{\mathcal{O}}\big(\log^k n\big)$ it runs in time polynomial of $\log n$.

We will also need the following definitions from number theory for the algorithm:

Definition 2.5. Given $r \in \mathbb{Z}$, $n \in \mathbb{Z}$ with $gcd(n, r) = 1$, *the order of n modulo r*, denoted $o_r(n)$ is the smallest integer $k > 0$ such that

$$n^k = 1 \pmod{r} \tag{4}$$

Definition 2.6. *Euler's totient function*, $\varphi(r)$, defined for $r \in \mathbb{N}$, is the number of positive integers less than or equal to r that are coprime to r.

Definition 2.7. $LCM(m)$ denotes the lcm of first m numbers. That is, it is least number L such that $n \mid L$ for all integers n with $1 \leq n \leq m$.

3 Description of the Algorithm

Most primality testing algorithms are based on Fermat's Little Theorem, which states that if p is a prime number, then for any integer a, $a^p - a$ will be evenly divisible by p. We first prove the following Lemma.

Lemma 3.1. *Let p be a prime number and let $0 < i < p$. Then*

$$\binom{p}{i} = 0 \pmod{p}. \tag{5}$$

Proof.

$$\binom{p}{i} = \frac{p(p-1)(p-2)\dots(p-i+1)}{i!} \tag{6}$$

Because p is prime and $i < p$, $gcd(p, i!) = 1$. Since $\binom{p}{i}$ is an integer, this means $i!$ divides $(p-1)(p-2)\dots(p-i+1)$. Thus

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

is an integer, which concludes the proof. □

We are now ready to prove Fermat's Little Theorem.

Theorem 3.2. *(Fermat's Little Theorem) Let p be a prime number. Then $a^p = a \pmod p$ for all $a \in \mathbb{Z}$.*

Proof. Note that $(-a)^p = -a^p \pmod p$ for all $a \in \mathbb{Z}$, so it sufficient to prove the lemma for $a \in \mathbb{N}$. This can be proved by induction on a. For $a = 0$, the Lemma clearly holds. Now suppose $a \geq 0$ and $a^p = a \pmod p$. By the binomial theorem,

$$(a+1)^p = \sum_{k=0}^{p} \binom{p}{k} a^k \tag{8}$$

By Lemma 3.1, $\binom{p}{k} = 0 \pmod p$ for $0 < i < p$. Thus taking the above equation modulo p leaves only the first and last terms of the binomial expansion, so we have.

$$(a+1)^p = a^p + 1 = a + 1 \pmod p. \tag{9}$$

by the inductive hypothesis. □

The AKS algorithm relies on a generalization of Fermat's Little Theorem that gives an identity that holds for all primes but no composites. Because of this, it can be used to distinguish primes from composites.

Theorem 3.3. *Let $a, n \in \mathbb{Z}$ with $n \geq 2$ and $gcd(a, n) = 1$. Then n is prime if and only if*

$$(X+a)^n = X^n + a \pmod n \tag{10}$$

Notice that X is a free variable, so it is never substituted with a number. Congruence is established by expanding $(X+a)^n$ and examining the coefficients for the different powers of X. More formally, this says $(X+a)^n = X^n + a$ on the ring of polynomials $\mathbb{Z}_n[x]$. We shall first prove that primality is a sufficient condition for the equivilance using Fermat's Little Theorem.

Lemma 3.4. *Let $a, n \in \mathbb{Z}$ with $n \geq 2$ and $gcd(a, n) = 1$. Then if n is prime*

$$(X+a)^n = X^n + a \pmod n \tag{11}$$

Proof. By the binomial theorem

$$(X+a)^n = \sum_{k=0}^{n} \binom{n}{k} x^k \tag{12}$$

Suppose n is prime. Then $\binom{n}{k} = 0 \pmod{n}$ for $0 < i < p$ by Lemma 3.1. This only leaves the first and last terms in the binomial expansion, so we have

$$(X+a)^n = X^n + a^n = X^n + a \pmod{p} \tag{13}$$

by Fermat's Little Theorem. □

Now for the other direction:

Lemma 3.5. *Let $a, n \in \mathbb{Z}$ with $n \geq 2$ and $gcd(a, n) = 1$. Then if n is composite*

$$(X+a)^n \neq X^n + a \pmod{n} \tag{14}$$

Proof. Suppose n is composite. Let q be a prime factor of n and k be the integer such that $q^k \mid n$ but $q^{k+1} \nmid n$. We claim that

$$q^k \nmid \binom{n}{q}. \tag{15}$$

To see this, suppose $q^k \mid \binom{n}{q}$. Then

$$\binom{n}{q} = \frac{n(n-1)(n-2)\dots(n-q+1)}{q!} = mq^k \tag{16}$$

for some integer $m \geq 1$ so we have

$$n = \frac{m(q-1)!q^{k+1}}{(n-1)(n-2)\dots(n-q+1)} \tag{17}$$

Let j be an integer with $0 < j < q$ and suppose $q \mid (n-j)$. Since $q \mid n$, we must have $q \mid j$, which is impossible, so $q \nmid (n-j)$ for all $0 < j < q$. Thus because q is prime and n is an integer, we get

$$\frac{m(q-1)!q^{k+1}}{(n-1)(n-2)\dots(n-q+1)} \tag{18}$$

is an integer. But this is a contradiction since it means $q^{k+1} \mid n$.

Now suppose the coefficient of X^q in the expansion of $(X-1)^n$ is divisible by n. Then

$$\binom{n}{q} a^{n-1} = mn \tag{19}$$

for some $m \in \mathbb{Z}$ so we have

$$\binom{n}{q} a^{n-1}/q^k = mn/q^k \tag{20}$$

This must be an integer since $q \mid n$. Since $gcd(a, n) = 1$ and $q \mid n$, $gcd(a, q) = 1$ so clearly

$$gcd(a^{n-q}, q^k) = 1. \tag{21}$$

Hence q^k must divide $\binom{n}{q}$, but this contradicts (15). This means the coefficient for X^q in $(X - a)^n$ is not divisible by n so it is nonzero modulo n. This proves $(X + a)^n \neq X^n + a \pmod{n}$. □

Theorem 3.3 gives a simple way of testing the primality of n: we choose an a and test whether (10) is satisfied. Unfortunately, this does not yield an algorithm that runs in polynomial time because the left hand side of (10), $(X + a)^n$, has $n + 1$ coefficients. This is exponential in $\log n$, so such as algorithm would run in exponential time. The idea behind AKS is to reduce the number of coefficients by evaluating both sides modulo a polynomial of the form $X^r - 1$ where r is small instead of modulo n.

$$(X + a)^n = X^n + a \pmod{n, X^r - 1} \tag{22}$$

The remainder $(X + a)^n \pmod{n, X^r - 1}$ has $r + 1$ terms, so it is possible to compute in polynomial time for small enough r. Furthermore, if n is prime, it clearly satisfies this for all values of a and r by Theorem 3.3. However, the converse is not true: for each a and r there exist composite numbers that can satisfy this, so testing this for one set of values of a and r is not sufficient to test for primality. Thus, AKS tests (22) on enough values of a to guarantee n is prime. We will show that for an appropriately chosen r bounded by a polynomial in $\log n$, if (22) is satisfied for a particular number of values of a also bounded by a polynomial in $\log n$, then n must be prime. Using this, we get a deterministic primality test algorithm that runs in polynomial time.

The AKS Primality Test

On input n, where n is a positive integer

1. If ($n = a^b$ for $a \in \mathbb{N}$ and $b > 1$), output COMPOSITE.
2. Find the smallest r such that $o_r(n) > \log^2 n$.
3. If $1 < gcd(a, n) < n$ for some $a \leq r$, output COMPOSITE.
4. If $n \leq r$, output PRIME.
5. For $a = 1$ to $\left\lfloor \sqrt{\phi(r)} \log n \right\rfloor$
6. If ($(X + a)^n \neq X^n + a \pmod{X^r - 1, n}$), output COMPOSITE.
7. Output PRIME.

Step 2 can be done by trying succesive values of r until we find one such that $n^k \neq 1 \pmod{r}$ for all $k \leq \log^2 n$. Step 3 can be done by computing $gcd(a, n)$ for all $a \leq r$ and returning COMPOSITE if $1 < gcd(a, n) < n$ for some a.

4 Correctness of the Algorithm

Lemma 4.1. *If n is prime, the algorithm returns PRIME.*

Proof. Since n is prime, the algorithm will not return COMPOSITE for steps 1 and 3 because these imply n has a divisor other than itself. Furthermore, by Theorem 3.3 step 6 can never output COMPOSITE. Therefore the algorithm will determine n is prime in step 6. □

Agrawal, Kayal, and Saxena [2] also proved the converse of this lemma which leads to the correctness of the algorithm. The proof is much more involved and will be omitted from this paper in favor of proving the algorithm runs in polynomial time.

Theorem 4.2. *The algorithm returns PRIME if and only if n is prime.*

5 Asymptotic Analysis of the Algorithm

A key part of determining the run time of the algorithm is getting a bound on r. This is given in Lemma 5.2. The Lemma is also important in proving the correctness of the algorithm because it proves that we can always find an apppriate r. In our proof of the Lemma, we need the following fact about the LCM of the first m numbers given by M. Nair [6].

Lemma 5.1. *For $m \geq 7$*

$$LCM(m) \geq 2^m \tag{23}$$

We can now prove r is bounded above by a number that is polynomial in $\log n$:

Lemma 5.2. *There an exists an $r \leq \max\left\{3, \lceil \log^5 n \rceil\right\}$ such that*

$$o_r(n) > \log^2 n. \tag{24}$$

Proof. When $n = 2$, $r = 3$ satisfies the conditions because $o_3(2) = 2$ ($2^2 = 1 \pmod 3$). Fix $n > 2$. Let $B = \lceil log^5 n \rceil$. For $n > 2$, $B > 10$ so Lemma 5.1 applies to B. Note that for any integer $m > 0$, the largest $k > 0$ such that $m^k \leq B \Rightarrow k \leq \log_m B$ is $\lfloor \log B \rfloor$. Now let r be the smallest number that does not divide

$$n^{\lfloor \log B \rfloor} \cdot \prod_{j=1}^{\lfloor \log^2 n \rfloor} (n^j - 1) \tag{25}$$

We claim that

(1) $o_r(n) > \log^2 n$.

(2) $r \leq \lceil \log^5 n \rceil$.

For (1), note that $gcd(r, n)$ cannot be divisible by all the prime divisors of r since otherwise r will divide $n^{\lfloor \log B \rfloor}$ and hence the above product by the observation above. This means that $r/gcd(r, n)$ will also not divide the above product. Since r is the least number not diving the above product, we must have $gcd(r, n) = 1$. Furthermore, since r does not divide the above

product, it must not divide $(n^j - 1)$ for $1 \leq j \leq \lfloor \log^2 n \rfloor$. For such j, $n^j \neq 1 \pmod{r}$. This gives us $o_r(n) > \log^2 n$.

For (2), note that can bound the above product by

$$n^{\lfloor \log B \rfloor} \cdot \prod_{j=1}^{\lfloor \log^2 n \rfloor} (n^j - 1) < n^{\lfloor \log B \rfloor} \cdot n^{(\frac{1}{2}(\log^2 n - 1))^{\lfloor \log^2 n \rfloor}} \leq$$
$$n^{\lfloor \log B \rfloor + \frac{1}{2}(\log^2 n)(\log^2 n - 1)} \leq n^{log^4 n} \leq 2^{log^5 n} \leq 2^B \tag{26}$$

□

By Lemma 5.1, $LCM(B)$, the least common multiple of the first B numbers, is at least 2^B. Since r is the least number that does not divide the product, the product must not be divisible by any number smaller than r. Hence the product is at least as big as the lcm of the first r numbers. Therefore we have

$$LCM(r) \leq n^{\lfloor \log B \rfloor} \cdot \prod_{j=1}^{\lfloor \log^2 n \rfloor} (n^j - 1) < 2^B \leq LCM(B) \tag{27}$$

so $r \leq B = \lceil \log^5 n \rceil$.

In order to analyze the run time of the algorithm, we also need the following results:

(1) Let a, b be integers with $|a|, |b| \leq n$. Then $b \mod a$ can be computed in $\mathcal{O}\big(\log^2 n\big)$ time.

(2) Let a, b be integers with $|a|, |b| \leq n$. Then ab can be computed in $\tilde{\mathcal{O}}\big(\log n\big)$ time.

(3) Let $p(x), q(x)$ be polynomials with degree less than or equal to r and coefficients with absolute value less than or equal to n. Then $p(x)q(x)$ can be computed in $\tilde{\mathcal{O}}\big(r \log n\big)$ time.

(4) Let $n \geq 2$ be an integer. Then there is algorithm that tests if there are integers $a \geq 2$ and $b \geq 2$ such that $n = a^b$ in $\tilde{\mathcal{O}}\big(log^3 n\big)$ time.

Result (1) comes from a simple recursive division algorithm given in [3].

Result (2) comes from the Fast Fourier Transform algorithm (given in [3]), which allows d-digit numbers to be multiplied in $\mathcal{O}\big(d \log d\big)$ time. Since a n can be expressed with $\lceil \log n \rceil$ digits, the multiplication runs in.

$$\mathcal{O}\big(d \log d\big) = \mathcal{O}\big(\log n \log \log n\big) = \mathcal{O}\big(\log n\big) \cdot poly(\log n) = \tilde{\mathcal{O}}\big(\log n\big) \tag{28}$$

time for $|a|, |b| \leq n$.

Result (3) also follows for the Fast Fourier Transform algorithm applied to polynomials, which allows the product of degree r polynomials to be computed with $\mathcal{O}\big(r \log r\big) = \tilde{\mathcal{O}}\big(r\big)$ multiplications. Each of these multiplications takes $\tilde{\mathcal{O}}\big(\log n\big)$ time by (2), giving a total time complexity of $\tilde{\mathcal{O}}\big(r \log n\big)$.

Result (4) comes from an algorithm for finding powers given by Smid [8].

Theorem 5.3. *The algorithm runs in* $\tilde{O}\left(\log^{21/2} n\right)$ *time.*

Proof. By result (4), step 1 of the algorithm takes $\tilde{\mathcal{O}}\big(log^3 n\big)$

For step 2, we try succesive values of r until we find one such that $n^k \neq 1 \pmod r$ for all $k \leq \log^2 n$. Thus each computation of $n^k \pmod r$ will take at most $\log^2 n$ multiplications modulo r (it is possible to do this with fewer multiplications, but this does not affect the result of the analysis). Since we are multiplying modulo r each product will have factors less than r. Hence by result (2), each multiplication takes $\tilde{\mathcal{O}}\big(\log r\big)$ time. Thus the computations on each r require $\tilde{\mathcal{O}}\big(\log r\big) \cdot \log^2 n = \tilde{\mathcal{O}}\big(\log r \log^2 n\big)$ time. By Lemma 5.2, we need to do the computations for r going up to $\lceil \log^5 n \rceil$. Hence we get the total run time of step 2 to be. $\lceil \log^5 n \rceil \cdot \tilde{\mathcal{O}}\big(\log r \log^2 n\big) = \tilde{\mathcal{O}}\big(\log \log^5 n \log^2 n \log^5 n\big) = \tilde{\mathcal{O}}\big(\log^7 n\big)$ time.

Step 3 requires computing the gcd of r numbers, where r is bounded by $\lceil \log^5 n \rceil$ from Lemma 5.2. Each gcd computations takes $\mathcal{O}\big(\log n\big)$ time [2]. Hence the total time complexity of this step is $\lceil \log^5 n \rceil \cdot \mathcal{O}\big(\log n\big) = \mathcal{O}\big(\log^6 n\big)$.

Step 4 simply compares r and n. We can do this by counting the number of digits in n and then seeing if r has that many or more. This takes time proportional to the number of digits in n, so it takes $\mathcal{O}\big(\log n\big)$ time.

Step 5 goes through all values of a from 1 to $\lfloor \sqrt{\phi(r)} \log n \rfloor$. Note that since $\phi(r) \leq r - 1$ for all r (if r is prime it is coprime to all positive integers less than or equal to it except itself), $\lfloor \sqrt{\phi(r)} \log n \rfloor = \mathcal{O}\big(\sqrt{r} \log n\big)$.

Step 6 computes $(X + a)^n$ and $X^n + a$ modulo $(X^r - 1, n)$. Although naively it takes n multiplications to compute $(X + a)^n \pmod{X^r - 1, n}$, we can do better by squaring $(X + a)$ modulo $(X^r - 1, n)$ $\lfloor \log n \rfloor$ times, which gives the values of $(X + a)^{2^j} \pmod{X^r - 1, n}$ for $1 \leq j \leq \lfloor \log n \rfloor$. We can then multiply the appropriate powers of $(X+a)^{2^j} \pmod{X^r - 1, n}$ to compute $(X + a)^n \pmod{X^r - 1, n}$. This method requires $\mathcal{O}\big(\log n\big)$ multiplications modulo $(X^r - 1, n)$: $\lfloor \log n \rfloor$ to compute the powers of $(X + a)$ and up to an another $\lfloor \log n \rfloor$ to get $(X + a)^n$. Since each product is taken modulo $(X^r - 1)$ every product has factors of degree less than or equal to r. Since each product is taken modulo n every product has factors with coefficients than or equal to n. Hence by result (3), each multiplication takes $\tilde{\mathcal{O}}\big(r \log n\big)$ time. This means computing $(X + a)^n \pmod{X^r - 1, n}$ takes a total of $\mathcal{O}\big(\log n\big) \cdot \tilde{\mathcal{O}}\big(r \log n\big) = \tilde{\mathcal{O}}\big(r \log^2 n\big)$ time. Since this is computed for every value of a in step 5, or $\mathcal{O}\big(\sqrt{r} \log n\big)$ times, the total time complexity of this step is $\mathcal{O}\big(\sqrt{r} \log n\big) \cdot \tilde{\mathcal{O}}\big(r \log^2 n\big) = \tilde{\mathcal{O}}\big(r^{\frac{3}{2}} \log^3 n\big)$ time. Since by Lemma 5.2, r is bounded by $\lceil \log^5 n \rceil$, this run time is equal to $\tilde{\mathcal{O}}\big((\log^5 n)^{\frac{3}{2}} \log^3 n\big) = \tilde{\mathcal{O}}\big(\log^{21/2} n\big)$. This time dominates all the other ones, so the total run time of the algorithm is $\tilde{\mathcal{O}}\big(\log^{21/2} n\big)$. □

The run time of the algorithm depends on the bound on r from Lemma 5. The following Lemma proven by Fouvry [4] improves this bound on r and hence lowers the bound on the run time of the algorithm. Let $P(m)$ be the greatest prime divisor of m. Then

Lemma 5.4. *There exist constants* $c > 0$ *and* n_0 *such that for all* $x \geq n_0$

$$\left|\left\{q \mid q \text{ is prime, } q \leq x, \text{ and } P(q-1) > q^{\frac{2}{3}}\right\}\right| \geq c\frac{x}{\ln x} \tag{29}$$

This improves out bound on r from $\mathcal{O}\big(\log^5 n\big)$ to $\mathcal{O}\big(\log^3 n\big)$ [2]. This means Step 6 runs in time $\tilde{\mathcal{O}}\big(r^{\frac{3}{2}} \log^3 n\big) = \tilde{\mathcal{O}}\big((log^3 n)^{\frac{3}{2}} \log^3 n\big) = \tilde{\mathcal{O}}\big(log^{15/2} n\big)$.

Theorem 5.5. *The algorithm runs in* $\tilde{\mathcal{O}}\big(\log^{15/2} n\big)$ *time.*

The best case bound on r is $\mathcal{O}\big(\log^2 n\big)$, in which case the algorithm runs in time $\mathcal{O}\big(\log^6 n\big)$. Both of the following conjectures give this bound on r [2].

Conjecture 5.6. *(Artin's Conjecture). Let* $n \in \mathbb{N}$ *not be a perfect square. Then the number of primes* $q \leq n$ *such that* $o_q(n) = q - 1$ *is asymptotically* $A(n) \cdot m / \ln m$ *where* $A(n)$ *is Artin's constant with* $A(n) > 0.35$.

Conjecture 5.7. *(Sophie-Germain Prime Density Conjecture). The number of primes* $q \leq m$ *such that* $2q + 1$ *is also a prime is asymptotically* $\frac{2Cm}{\ln^2 m}$.

In 2003, Hendrik Lenstra and Carl Pomerance [5] gave a modified version the AKS primality which has a run time that is provably $\tilde{\mathcal{O}}\big(\log^6 n\big)$.

References

[1] L.M. Adleman, C. Pomerance, and R.S. Rumely. On distinguishing prime numbers from composite numbers. *The Annals of Mathematics*, 117(1):173–206, 1983.

[2] M. Agrawal, N. Kayal, and N. Saxena. Primes is in p. *The Annals of Mathematics*, 160(2):781–793, 2004.

[3] S. Dasgupta, C Papadimitriou, and U Vazirani. Algorithms, 2006.

[4] E. Fouvry. Theoreme de brun-titchmarsh; application au theoreme de fermat. *Inventiones Mathematicae*, 79(2):383–407, 1985.

[5] HW Lenstra and C. Pomerance. Primality testing with gaussian periods. *Lecture Notes in Computer Science*, pages 1–1, 2003.

[6] M. Nair. On chebyshev-type inequalities for primes. *The American Mathematical Monthly*, 89(2):126–129, 1982.

[7] M.O. Rabin. Probabilistic algorithm for testing primality. *Journal of Number Theory*, 12(1):128–138, 1980.

[8] M. Smid. Primality testing in polynomial time. 2003.