An Exposition of the AKS Polynomial Time Primality Testing Algorithm

Benjamin Linowitz

A THESIS

in

Mathematics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Master of Arts 2006

Supervisor of Thesis

Graduate Group Chairman

Contents

Abstract

After providing a brief survey of primality testing algorithms, I present a thorough analysis of the unconditional deterministic polynomial time algorithm for determining whether a given integer is prime or composite proposed by Agrawal, Kayal and Saxena in their paper "Primes is in P" [aks].

1 Introduction

Among the various branches of mathematics, number theory, which can broadly be defined as the study of the integers and their properties, stands out as one of the oldest and largest. One of the reasons that number theory has maintained its appeal over the last few thousand years is that its problems can often be stated in extremely simple terms, enticing professional and amateur mathematicians alike. This is not to say that all problems are easily provable. Consider, for example, Goldbach's conjecture.

In 1742 a Prussian mathematician named Christian Goldbach wrote a letter to Leonhard Euler conjecturing that every integer greater than 5 can be written as the sum of 3 primes. During the eighteenth century 1 was considered to be a prime, so the today's version of the conjecture is: "Every even integer greater than 2 can be written as the sum of 2 primes." Despite the problem's simple wording, Goldbach's conjecture remains unsolved to this day.[1]

[1] Although a proof of Goldbach's conjecture continues to elude the mathematical community, the past few decades have witnessed substantial progress. In 1966 Chen Jingrun proved that every even

One particular subfield of number theory is notorious for producing problems capable of baffling even the most seasoned mathematicians: prime number theory. Although any schoolchild can multiply two integers and determine their product, determining the prime factorization of a given integer has actually been an active area of mathematical research for over 2300 years![2] When studying prime numbers, one question immediately presents itself: Given an integer n, how can one tell if n is prime?

If our sole mission is to determine the primality of n, without having to take the number of steps in the algorithm into consideration, then one could simply try to divide n by every integer m less than $\sqrt{n}$.[3] On a practical level however, trial division is a bit too naive. As the value of n increases, the number of divisions needed to be performed grows exponentially.

Testing primality via trial division is an example of a *deterministic* algorithm; that is, given an integer n as input, the algorithm will always return the same output. Not all algorithms are deterministic however. Some are *probabilistic*, meaning that the

integer of sufficient size can be written as the sum of either two primes or a prime and a product of two primes (see [chen] for the proof). In 1975 Montgomery and Vaughan proved (see [vau]) the existence of constants c, C such that for sufficiently large N, the number of even integers less than N not expressible as the sum of two primes is at most $C \cdot N^{1-c}$. In 2005 Roger Heath-Brown showed (see [bro]) that all but finitely many even integers can be expressed as the sum of two primes and exactly 13 powers of 2.

[2]The doubtful reader is invited to factor the following 212 digit number, for which security company RSA is offering $30,000:

$$\begin{gathered}
7403756347956171282804679609742957314259 31 \\
8888923128908493623263897276503402826627 68 \\
9199641962511784399589433050212758537011 89 \\
6809828673317327310893090055250511687706 32 \\
9907239638078671008609696253793465056379 6359
\end{gathered}$$

[3]We only have to try integers less than $\sqrt{n}$ as $\forall m > \sqrt{n}$ such that m divides n, $\frac{n}{m} < \sqrt{n}$ and also divides n.

probability of the algorithm mistakenly declaring a composite integer to be prime is greater than zero.[4] Note that given a probabilistic algorithm, it is quite easy to make the probability of incorrect output arbitrarily small.

Suppose that the algorithm fails to return the correct output with probability $0 < p < 1$ and let $\epsilon > 0$. Choose an integer n_0 such that $p^{n_0} < \epsilon$. Now run the algorithm n_0 times. If the algorithm outputs composite during any of these n_0 trials, then n is certainly not prime. Moreover, the probability that the algorithm will mistakenly declare n to be prime all n_0 times is equal to $p^{n_0} < \epsilon$. In this way one can take an extremely fast probabilistic primality test and reduce its error rate to the point that it is actually less than the probability of the computer running the algorithm failing.

In 2002, Manindra Agrawal, Nitin Saxena and Neeraj Kayal developed a new deterministic primality test. What made it different from previous deterministic primality tests is that their test runs fairly rapidly. Specifically, given x bits of input, the number of steps needed by their test is bounded by some polynomial in x. The goal of this paper is to thoroughly analyze Agrawal, Neeraj and Saxena's algorithm, and prove that it does indeed run in polynomial time. After providing a brief survey of primality tests, I define the AKS algorithm (Section 6), prove its correctness (Section 7), and determine its complexity (Section 8).

The astute reader may be thinking: "If other primality testing algorithms are so fast, why has the AKS algorithm become so famous?" Truth be told, the AKS

[4]In practice, being a probabilistic algorithm means that the algorithm uses a pseudo-random number generator.

algorithm has become as famous as it has largely because it is heartening to mathematicians and computer scientists. It shows that there are still open problems whose proofs are not overly complex and tedious but rather short and elegant. In the words of P. Leyland: "One reason for the excitement within the mathematical community is that not only does this algorithm settle a long-standing problem, it also does so in a brilliantly simple manner. Everyone is now wondering what else has been similarly overlooked."[cra]

In a similar spirit, I have striven to make this paper accessible to as broad an audience as possible. That in mind, I have included an Appendix containing proofs of all results not covered in a first course in abstract algebra (at the level of groups, rings and fields).

2 A Survey of Primality Testing

Like many areas of mathematics, the history of primality testing begins with the ancient Greeks. Around 250 BC Eratosthenes developed a novel way for computing all of the prime numbers less than any fixed integer. His algorithm will serve as our survey's starting point.

2.1 The Sieve of Eratosthenes

Given an integer n, Eratosthenes sought an easy way to compute all of the prime numbers less than n. His idea was to first list all of the integers between 2 and n. The first integer on the list, 2, is certainly prime. We therefore circle 2. Every multiple of 2 is assuredly not a prime however (as 2 divides each of them), and so we draw an "X" through all multiples of 2. We now label the next integer in our list that has not been crossed out a prime and circle it (in our case the next integer is 3). Continuing in this manner yields a list with all primes less than n circled.

One of Eratosthenes' important realizations was that by the time one reaches $\sqrt{n}$, all integers not crossed off are prime. Suppose, for example, that there was some composite integer $m > \sqrt{n}$ that has not yet been crossed out. Let $m = a \cdot b$. Then one of a, b is less than $\sqrt{n}$ and has m as a multiple. But in this case, m would have an "X" through it. This proves Eratosthenes' assertion.

We provide a simple implementation of the Sieve of Eratosthenes, written in pseudocode.

Algorithm 2.1.

```
Input:  n ∈ ℕ
0:  m[1..n] integer array;
1:  for i = 1 to n do  //initialize an array storing first n integers
2:    m[i]← i;
3:  j ← 2;
4:  while j^2 < n do
5:    if m[j] ≠ 0 then
6:          t ← 2 · j;
7:          while t ≤ n do //Set all multiples of m[j] to 0
8:                  m[t] ← 0; t ← t + j;
9:    j ← j + 1;
10:  for i = 2 to n do
11:  if m[i] ≠ 0 then return m[i] "is prime";
```

It goes without saying that for large value of n, the Sieve of Eratosthenes requires a lot of memory. A second disadvantage is that proving primality can require up to $\sqrt{n}$ cycles. The sieve does however, have one major advantage: it requires virtually no multiplication and division.

2.2 Wilson's Characterization

Our next primality test is based on a theorem by John Wilson, a student of Edward Waring, in 1770:

Theorem 2.2. *Let $n \in \mathbb{N}$. Then n is prime if and only if $(n-1)! \equiv -1 \pmod{n}$.*

Proof.

"$\Rightarrow$": Suppose n is prime. Then every integer in the interval $[2, ..., n-2]$ is relatively prime to n and has a unique inverse modulo n. Therefore

$$\prod_{2 \leq j \leq n-2} j \equiv 1 \pmod{n}.$$

these congruences The result follows as $(n-1) \equiv -1 \pmod{n}$.

"$\Leftarrow$": Suppose now that n is composite. Then $\{1, 2, ..., n-1\}$ contains all prime factors of n, which implies that $(n-1)! \equiv 0 \pmod{n}$. □

Because of this characterization of primes, we can determine the primality of an integer n by calculating $(n-1)! \pmod{n}$. Unfortunately, this computation requires $n-1$ multiplications, making it horribly inefficient. these congruences

2.3 Fermat's Primality Test

Recall that if p is prime, then $|(\mathbb{Z}/p\mathbb{Z})^*| = \varphi(p) = p-1$. Group theory tells us that the order of any element of a group divides the order of the group, so every element

$a \in (\mathbb{Z}/p\mathbb{Z})^*$ has an order dividing $p-1$. This fact was phrased in the form of a theorem by Pierre de Fermat during the middle of the seventeenth century.

Theorem 2.3. *(Fermat's Little Theorem)* $\forall 0 < a \leq p-1, a^{p-1} \equiv 1 \pmod{p}$

Out of Fermat's characterization of primes arises a natural primality test: Choose k integers a_i at random from the interval $[2, ..., n-1]$ and test to make sure that for every $i \leq k$, $a_i^{n-1} \equiv 1 \pmod{n}$.If the congruence does not hold for any of the a_i's, then we can be certain that n is composite. If, on the other hand, the congruence holds for all of the a_i's, then we call n a *probable prime*, as there is a small probability that we would have found n to be composite had we chosen different a_i's from $[2, ..., n-1]$. Clearly this primality test will not be deterministic but rather probabilistic.

In the introduction, we described polynomial time algorithms as those in which the number of steps needed is bounded by a polynomial in the number of bits of input. We will show in Section 4.3 that modular exponentiation is polynomial. That is, the number of steps needed to calculate $a^{n-1} \pmod{n}$ is bounded by a polynomial in $log_2(n)$. This shows that Fermat's primality test is a polynomial time algorithm.

Curious readers may be wondering whether or not it is possible for a composite integer to "pass" Fermat's primality test. It turns out that such numbers do exist and our called *Carmichael numbers*.

Definition 2.4. Let $n \in \mathbb{N}$. We call n a *Carmichael number* if $gcd(b, n) = 1 \Rightarrow b^{n-1} \equiv 1 \pmod{n}$

Even though Carmichael numbers are rare (there are only 585,355 less than 10^{17}),

there are enough of them to cause Fermat's primality test to be seldomly used.[5]

In recent years several sophisticated variations of Fermat's primality test have arisen. We will end our brief survey by discussing one of these congruences these variants, the Solovay-Strassen primality test.

2.4 The Solovay-Strassen Primality Test

Let p be a prime integer and $(\mathbb{Z}/p\mathbb{Z})^*$ be the group of invertible integers modulo p. We know that this group is cyclic, so let g be a generator of $(\mathbb{Z}/p\mathbb{Z})^*$ and consider the set of powers of g: $\{g, g^2, ..., g^{p-1}\}$. This set contains every element of $(\mathbb{Z}/p\mathbb{Z})^*$, and because half of the exponents are even, we know that there are $\frac{p-1}{2}$ squares modulo p (we call these squares the *quadratic residues* of p). Suppose that $a = g^{2i}$ is a quadratic residue modulo p. Then $a^{\frac{p-1}{2}} \equiv g^{2i^{\frac{p-1}{2}}} = g^{p-1\cdot i} \equiv 1 \pmod{p}$. Now suppose that $b = g^j$ is not a quadratic residue modulo p (notice that this implies j is odd). Because $b^{p-1} \equiv g^{p-1\cdot j} \equiv 1 \pmod{p}$ and $b^{\frac{p-1}{2}\cdot 2} = b^{p-1}$, it must be the case that $b^{\frac{p-1}{2}} \equiv -1 \pmod{p}$ (because there are only two square roots of 1 modulo p, 1 and -1). We now state this result in the form of a theorem:

Theorem 2.5. *(Euler) Let p be prime and $1 \leq a < p$. Then*

$$(a|p) \equiv a^{\frac{p-1}{2}} \pmod{p}$$

[5] In 1994 it was shown that there are infinitely many Carmichael numbers. See [gran] for the proof.

$$\textit{where } (a|p) = \begin{cases} 1 & \textit{if } a \textit{ is a quadratic residue of } p \\ -1 & \textit{if } a \textit{ is not a quadratic residue of } p \\ 0 & \textit{if } p \textit{ divides } a \end{cases} \cdot$$

Note that the converse of this theorem is false. For example, let $n = 365$ and $a = 7$. Then $(7|365) = -1$ (this can be checked by direct computation) and $7^{162} \equiv -1 \pmod{365}$, but $365 = 5 \cdot 73$ is composite. Integers like 7, for which the converse of our theorem is false have a special name: *E-liars* for n (the "E" is for Euler). Thus, 7 is an E-liar for 365.

In 1977 Robert Solovay and Volker Strassen used Euler's theorem in order to develop a primality test. Their idea was to randomly choose k integers less than n and ensure that each satisfied the above equality. It turns out that roughly half of the elements of $(\mathbb{Z}/n\mathbb{Z})^*$ are E-liars for n (see [diet] for a readable proof of this), which means that each iteration of Solovay and Strassen's test has a probability of error of $\frac{1}{2}$. If we run the test k times however, the percentage of error becomes 2^{-k}, which can be made arbitrarily small.

3 Notation

As usual, $\mathbb{N}$ refers to the natural numbers, $\mathbb{Z}$ refers to the integers, and $\mathbb{R}$ refers to the reals. If $x \in \mathbb{R}$, then $\lceil x \rceil$ denotes the smallest integer greater than x, and $\lfloor x \rfloor$ denotes the greatest integer less than x. Given $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, 2, ..., n\}$. Let a, b be elements of some ring R. Then we write $b|a$ to mean b divides a in R, i.e., $\exists c \in R$ such that $a = b \cdot c$.

Let $\mathbb{Z}/n\mathbb{Z}$ denote the ring of integers modulo n, $(\mathbb{Z}/n\mathbb{Z})^*$ the abelian group consisting of the units in $\mathbb{Z}/n\mathbb{Z}$, and $\mathbb{F}_{p^f}$ the finite field with p^f elements (in the case that p is prime). Let $f(x)$ be an irreducible polynomial in $\mathbb{F}_p[x]$. Then we write $f(x) \equiv g(x) \pmod{h(x), p}$ to represent the equation $f(x) - g(x) = 0$ in the ring $(\mathbb{Z}/p\mathbb{Z})[x]/(h(x))$.

We will often write $(a, b) = c$ as shorthand for $gcd(a, b) = c$. Suppose $r, p \in \mathbb{Z}$ with $(r, p) = 1$. Then $o_r(p)$ denotes the smallest non-negative integer k such that $p^k \equiv 1 \pmod{r}$. Also, given $r \in \mathbb{N}$, $\varphi(r)$ is equal to the number of integers less than r that are relatively prime to r. It follows from Lagrange's Theorem[6] that $o_r(p)|\varphi(r)$.

[6]Recall that Lagrange's Theorem states that if H is a subgroup of G, then the order of H divides that order of G.

4 Asymptotic Notation and Runtime Analysis

Throughout this paper, all logarithms will be to the base 2 and denoted $log(n)$ rather than $log_2(n)$ or $lg(n)$. Additionally, all of our runtime analysis will be done will use "Big-O" notation, which is defined as follows:

$O(f) = \{g|\exists c \exists n_0 \forall n > n_0 : |g(n)| \leq c \cdot |f(n)|\}$
$O^{\sim}(f) = \{g|\exists c \exists n_0 \exists k \forall n > n_0 : |g(n)| \leq c \cdot |f(n)| log^k(|f(n)|)\}$
$\Omega(f) = \{g|\exists c \exists n_0 \forall n > n_0 : |g(n)| \geq c \cdot |f(n)|\}$
$\Theta(f) = O(f) \cap \Omega(f)$

Fact 4.1. *If* $g_1(n) = O(f_1(n))$ *and* $g_2(n) = O(f_2(n)$*, then*

(i) $g_1(n) + g_2(n) = O(max\{f_1(n), f_2(n)\})$

(ii) $g_1(n) \cdot g_2(n) = O(f_1(n) \cdot f_2(n))$

An implication of part (i) of the above fact is that complex algorithms can often have have a single step whose complexity dominates the other steps, meaning that the overall complexity of an algorithm is sometimes simply the complexity of a single step of the algorithm. Indeed, this is the case with the AKS Primality Testing Algorithm.

Given a nonnegative integer n, let $||n||$ denote the number of bits in the binary representation of n. Then $||n|| = \lceil log(n+1) \rceil$(recall that all logarithms are base 2).

Definition 4.2. We say that an algorithm is *polynomial* if given input n, all computations are performed in $O(||n||^k)$ bit operations for some $k \geq 0$, where a bit operation refers to the number of steps needed to perform an arithmetical operation on a natural number given in binary representation.

We now show the complexity of basic operations on numbers.

4.1 Basic Operations

We begin this section by showing that many arithmetical operations can be executed in polynomial time. Although our proofs will use naive algorithms and therefore produce crude estimates, we will provide the reader with more refined estimates by referring to advanced methods developed during the past few decades.

Lemma 4.3. *Let $n, m \in \mathbb{N}$. Then*

(i) Computing $m + n$ takes $O(||n|| + ||m||) = O(log(n) + log(m))$ bit operations.

(ii) Computing $m \cdot n$ takes $O(||n|| \cdot ||m||) = O(log(n) \cdot log(m))$ bit operations.

(iii) Computing the quotient n div m and the remainder n mod m takes $O((||n|| - ||m|| + 1) \cdot ||m||)$ bit operations.

Proof. In each case the result follows from grade school algorithms, adapted to binary notation. □

Addition and subtraction can therefore be computed in linear time, while multiplication and division can be computed in quadratic time, which is still polynomial. These bounds suffice to prove that the AKS primality testing algorithm is polynomial, however much faster methods are known.

Fact 4.4. *Let $m, n \in \mathbb{N}$ with at most k bits each. Then*

(i) m and n can be multiplied with $O(k(log(k))(loglogk)) = O^{\sim}(k)$ bit operations.

(ii) *n div m and n mod m can be computed using $O(k(log(k))(loglogk)) = O^{\sim}(k)$ bit operations.*

(iii) *Multiplication of two polynomials of degree d with coefficients at most m bits in size can be done in $O^{\sim}(d \cdot m)$ bit operations.*

Proof. See [ss] for a proof of (i), and [vz] for proofs of (ii) and (iii). □

Armed with these estimates, we are ready to prove the complexity of several important algorithms, all of which are used in the AKS algorithm. For a more thorough treatment, the reader is encouraged to consult Dietzfelbinger [diet], whose treatment we will be following.

4.2 The Euclidean Algorithm

The Euclidean Algorithm, which appeared in Euclid's Elements [euc] around 300 BC, is one of the oldest algorithms known and provides an extremely efficient way of computing the greatest common divisor of two elements in a Euclidean Domain (though we will be using $\mathbb{Z}$). The implementation that we will be studying is:

Algorithm 4.5.

`Input:` $m, n \in \mathbb{Z}$

`0:` a, b `integer;`

`1:` `if` $|n| \geq |m|$

`2:` `then` $a \leftarrow |n|;\ b \leftarrow |m|;$

3: else $b \leftarrow |m|$; $a \leftarrow |n|$;

4: while $b > 0$ repeat

5: $(a, b) \leftarrow (b, a \text{ mod } b)$; //i.e., $a_i = b_{i-1}, b_i = a_{i-1} \text{ mod } b_{i-1}$

6: return a;

Let (a_i, b_i) denote the values stored in variables (a, b) after the loop in lines 4,5 has been executed for the i^{th} time. Then $(m, n) = (a_i, b_i)$ (which implies that the algorithm returns (m, n)), and the numbers $b_1, b_2, ...$ form a strictly decreasing sequence.[7] We are now ready to determine the algorithm's complexity.

Lemma 4.6. *The complexity of the above algorithm is* $O(log(n) \cdot log(m))$.

Proof. We begin by showing that the **while** loop in lines $4, 5$ of the algorithm runs at most $2 \cdot min\{||n||, ||m||\}$ times. Recall that this loop ends when the strictly decreasing sequence $b_1, b_2, ...$ reaches 0. Consider the following two cases:

Case 1: $b_{i+1} > \frac{1}{2} b_i = \frac{1}{2} a_{i+1}$. Then $b_{i+2} = a_{i+1} - b_{i+1} < \frac{1}{2} b_i$.

Case 2: $b_{i+1} \leq \frac{1}{2} b_i = \frac{1}{2} a_{i+1}$. Then $b_{i+2} = (a_{i+1} \text{ mod } b_{i+1}) < b_{i+1} \leq \frac{1}{2} b_i$.

We therefore see that for every two executions of the loop, the bit length of the variable b is reduced by 1. It follows that after $2 \cdot min\{||n||, ||m||\}$ many executions of the **while** loop, b= 0, stopping the loop.

Using elementary methods for dividing allows us to compute $a_i \text{ mod } b_i$ in $O((||a_i|| -$

[7] These results are elementary and can be found in any Abstract Algebra textbook, for example [df].

$||b_i|| + 1) \cdot ||b_i||)$ many bit operations. Since $b_i = a_{i+1}$ for $0 \leq i \leq t$ we have:

$$(||a_i|| - ||b_i|| + 1) \cdot ||b_i|| = ||a_i|| \cdot ||b_i|| - ||a_{i+1}|| \cdot ||b_i|| + ||b_i|| \leq (||a_i|| - ||a_{i+1}||)||b_0|| + ||b_i||.$$

Then we can bound the number of bit operations of the algorithm by

$$\sum_{0 \leq i \leq t} O((||a_i|| - ||b_i|| + 1)||b_i||)) = O(\sum_{0 \leq i \leq t} ((||a_i|| - ||a_{i+1}||)||b_0|| + ||b_i||)))$$

$$= O(||a_0|| \cdot ||b_0|| + t \cdot ||b_0||) = O(||n|| \cdot ||m||).$$

□

The next calculation we wish to analyze is Fast Modular Exponentiation. Suppose we wish to calculate $(x+a)^n \pmod{X^r - 1, n}$. Were we to proceed in the most naive way, carrying out $n - 1$ multiplications and divisions by $(x^r - 1), n$, the number of bit operations would not be polynomial but instead exponential. A simple trick often employed in order to speed up this calculation is Fast Modular Exponentiation, sometimes referred to as repeated "squaring".

4.3 Fast Modular Exponentiation

Let $n = 2^{a_1} + 2^{a_2} + \cdots + 2^{a_l}$ where $a_1 > a_2 > \cdots > a_l$. Define $f_0 := (x + a), f_{i+1}(x) = f_i(x)^2 \pmod{x^r - 1, n}$. Then $f_{a_j}(x) = (x + a)^{a_j}$. If we further define $g_1(x) := f_{a_1}(x)$

and $g_k(x) \equiv g_{k-1}(x)f_k(x) \pmod{x^r - 1, n}$, then we see that

$$g_l(x) \equiv (x+a)^{2^{a_1}+\cdots+2^{a_l}} = (x+a)^n \pmod{x^r - 1, n}.$$

We have therefore computed $(x+a)^n \pmod{x^r-1, n}$ in $a_1 + l \leq 2log(n)$ steps, where a step consists of multiplying two polynomials of degree less than r with coefficients in $\mathbb{Z}/n\mathbb{Z}$. This leads to a total runtime of $O^{\sim}(r \cdot log^2(n))$.

4.4 A Perfect Power Test

Suppose we are given an integer n and want to determine whether or not n is prime. Before inputing n into a complex (and possibly memory intensive) primality proving algorithm, it is often convenient to ensure that n is not a perfect power, i.e., $\forall a, b > 1$, $n \neq a^b$.

In order to test whether or not n is a perfect power, we will be conducting a binary search of the integers $\{1, 2, ..., n\}$ for a number a number m such that $n = m^b$ for some $b > 1$. Let $b > 1$ and suppose that if a solution m to $m^b = n$ exists, then it must lie in some interval $[c_i, d_i]$. When $i = 0$ we may take $[c_0, d_0] = [1, n]$. To define $[c_{i+1}, d_{i+1}]$, consider $\alpha := \lfloor \frac{(c_i+d_i)}{2} \rfloor$. If $\alpha^b = n$ then we're done. If $\alpha^b > n$, let $[c_{i+1}, d_{i+1}] = [c_i, \alpha]$; otherwise $\alpha^b < n$ and we let $[c_{i+1}, d_{i+1}] = [\alpha, d_i]$. We continue in this manner until $|c_i - d_i| \leq 1$. We then increase the value stored in variable b and start the loop again. Performing this loop for all $b \leq log(n)$ completes our algorithm. A pseudocode implementation of this algorithm follows:

Algorithm 4.7.

Input: $n \in \mathbb{N}$

0: a, b, c, m integer

1: $b \leftarrow 2$

2: while $b \leq log(n)$ do //Loop 1

3: a=1;c=m;

4: while $c - a \geq 2$ do //Loop 2

5: $m \leftarrow (a + c)$ div 2;

6: $p \leftarrow min\{m^b, 1\}$;

7: if $p = n$ then return "n is a perfect power";

8: if $p < n$ then $a \leftarrow m$ else $c \leftarrow m$;

9: $b \leftarrow b + 1$;

10:return "n is not a perfect power."

Loop 1 will run at most $log(n)$ times. Also, it will take at most $log(n)$ iterations of loop 2 before $|c - a| \leq 1$. During each iteration of loop 2, we calculate $(a + c)$ **div** 2 and m^b, which can be done in $O^{\sim}(log(n))$ bit operations by Fact 4.4. The complexity of the entire algorithm is therefore $O^{\sim}(log^3(n))$.

5 Motivation

We describe a characterization of prime numbers that will be provide the conceptual foundation for our polynomial time algorithm:

Lemma 5.1. *Let $a \in \mathbb{Z}, n \in \mathbb{N}$, such that $(a, n) = 1$. Then*

$$n \text{ is prime } \iff (x + a)^n \equiv x^n + a \pmod{n}.$$

Proof. By the binomial theorem we have:

$$(x + a)^n = x^n + \sum_{0<i<n} \binom{n}{i} a^{n-i} x^i + a^n$$

"⇒": If n is prime, then for $0 \leq i \leq n-1$, $\binom{n}{i}$ is divisible by n. Then within the ring $(\mathbb{Z}/n\mathbb{Z})[x]$, we have $(x + a)^n = x^n + a^n$. By Fermat's Little Theorem, $a^n = a$ within $(\mathbb{Z}/n\mathbb{Z})$, hence $(x + a)^n = x^n + a \pmod{n}$ as required.

"⇐": If n is composite, then let q be a prime divisor of n with $q^s \parallel n$. The coefficient of x^{n-q} in the binomial expansion of $(x + a)^n$ is $\frac{n(n-1)\cdots(n-q+1)}{q!} \cdot a^q$. The numerator of $\frac{n(n-1)\cdots(n-q+1)}{q!}$ is divisible by q^s but not q^{s+1}, the denominator is divisible by q, hence $\frac{n(n-1)\cdots(n-q+1)}{q!} \neq 0$ in $\mathbb{Z}/n\mathbb{Z}$. Additionally, $(a, n) = 1 \Rightarrow (a, q) = 1 \Rightarrow (a, q^s) = 1 \Rightarrow (a^q, q^s) = 1 \Rightarrow \frac{n(n-1)\cdots(n-q+1)}{q!} \cdot a^q \neq 0$ in $\mathbb{Z}/n\mathbb{Z}$. Therefore $(x + a)^n \neq x^n + a$ in $(\mathbb{Z}/n\mathbb{Z})[x]$. □

The above identity suggests a simple method for testing the primality of an integer n: choose an integer a such that $(a, n) = 1$ and calculate

$$f(x) := (x + a)^n - (x^n + a)$$

within the ring $(\mathbb{Z}/n\mathbb{Z})[x]$. If $f(x) \equiv 0$, then n is prime, otherwise it is not. Although this is certainly a valid primality test, it is horribly inefficient as it involves the computation of n coefficients. The simplest method for reducing the number of coefficients that need to be computed is to evaluate f(x) modulo n *and* modulo some polynomial of small degree, say $x^r - 1$.

Although it is clear that all primes p satisfy $(x+a)^p-(x^p+a) \equiv 0 \in (\mathbb{Z}/p\mathbb{Z})[x]/(x^r-1)$ for all values of a and r, it may be the case that some composite integer n satisfies the same identity for a few values of a and r. It turns out that for a judiciously chosen r, if the above identity is satisfied for several values of a, then n can be shown to be a prime power. The number of a's and the appropriate value of r are bounded by $log(n)$, meaning that we have just described a deterministic polynomial time primality testing algorithm.

6 The Algorithm

The remainder of this paper will be spent proving the correctness and complexity of the following primality testing algorithm:

Algorithm 5.1

Input: n ≥ 1

STEP 1. If $\exists a, b > 1 \in \mathbb{N}$ such that $n = a^b$, then Output COMPOSITE;

STEP 2. Find the minimal $r \in \mathbb{N}$ such that $o_r(n) > log^2(n)$;

STEP 3. For a = 1 to r do

if $1 < (a, n) < n$, then Output COMPOSITE;

STEP 4. if $r \geq n$, then Output PRIME[8];

STEP 5. For a = 1 to $\lfloor \sqrt{\phi(r)} \cdot log(n) \rfloor$ do

if $(x + a)^n \not\equiv x^n + a \pmod{x^r - 1, n}$, then Output COMPOSITE;

STEP 6. Output PRIME;

[8]We will show in Lemma 2 of the correctness proof that $r \leq log^5(n)$. Therefore, this step is only relevant for $n \leq 5,690,034$. On a practical level however, the most efficient way to determine the primality of an integer $n \leq 5,690,034$ would be to simply consult a database of small primes (there are only 392,928 primes less than 5,690,034).

7 The Correctness Proof

Theorem 7.1. *The above algorithm outputs PRIME if and only if n is prime.*

Proof. Suppose n is prime. Then n is certainly not of the form a^b for any $a, b > 1$, so STEP 1 will not output COMPOSITE. Since n is prime, we also know that $\forall x \in \mathbb{N}$, $(n, x) = 1$ or n. Hence STEP 3 will not output composite either. We have already seen that if n is prime, then $(x + a)^n \equiv x^n + a \pmod{n}$, so STEP 5 will not output COMPOSITE. Therefore the algorithm will output PRIME.

Now suppose that the above algorithm outputs PRIME. If the algorithm returns PRIME during STEP 4, then we know that $\forall m < n$, $(m, n) = 1$ (this was checked in STEP 3), meaning n is prime.[9] The remaining case, in which the algorithm returns PRIME during STEP 6, will take considerably more effort and require some extra machinery.

We begin with two lemmas that will bound the size of the r found in STEP 2. This bound will be important when we determine the algorithm's overall complexity.

Lemma 7.2. *There exists an integer $r \in \mathbb{N}$ with the following properties:*

(i) $r \leq max\{3, \lceil log^5(n) \rceil\}$

(ii) $o_r(n) > log^2(n)$

(iii) $(r, n) = 1$

[9] i.e., for $n > 1, \varphi(n) = n - 1 \iff n$ is prime.

Proof. If $n = 2$, then $r = 3$ satisfies the requisite conditions, so may assume that $n > 2$. Then $\lceil log^5(n) \rceil > 10$, meaning that Lemma 9.1 of the Appendix implies that

$$lcm([\lceil log^5(n) \rceil]) > 2^{\lceil log^5(n) \rceil}.$$

Now define

$$N := n \cdot \prod_{i=1}^{\lfloor log^2(n) \rfloor} (n^i - 1).$$

Let r be the smallest integer not dividing N. Then condition (2) is obviously satisfied as r is not a divisor of $(n^i - 1)$ for $i \leq \lfloor log^2(n) \rfloor$ (i.e., $n^i \not\equiv 1 \pmod{r}$). To see that condition (1) is satisfied as well, note that

$$N \leq n^{1+2+\cdots+log^2(n)} = n^{\frac{1}{2} \cdot (log^4(n) + log^2(n))} < n^{log^4(n)} = 2^{log^5(n)}.$$

Thus, by lemma 9.1, $N < lcm([\lceil log^5(n) \rceil])$. Therefore $\exists r_0 \leq \lceil log^5(n) \rceil$ such that $r_0 \not| N$ and as r was chosen to be minimal, we have $r < \lceil log^5(n) \rceil$ as well.

We now prove (iii). It is clear that $(r, n) < r$, as otherwise r would divide n and hence N. Thus $\frac{r}{(r,n)}$ is another integer less than $max\{3, \lceil log^5(n) \rceil\}$ not dividing N. Because r was chosen to be minimal, it must be the case that $\frac{r}{(r,n)} = r$, i.e., $(r, n) = 1$.

□

Because $o_r(n) > 1$, n must have some prime divisor p such that $o_r(p) > 1$. STEP 3 did not output COMPOSITE, so we know that $(n, r) = (p, r) = 1$. Therefore

$p, n \in (\mathbb{Z}/r\mathbb{Z})^*$. Additionally, we know that $p > r$ as otherwise STEP 3 or STEP 4 would have returned a decision regarding the primality of n. Throughout the remainder of this paper, we will make the following assumptions:

Hypothesis 7.3.

(i) Natural number n has p as a prime divisor.

(ii) r is less than $\lceil log^5(n) \rceil$ and $o_r(n) > log^2(n)$

(iii) $(r, n) = 1$

(iv) $l := \lfloor \sqrt{\phi(r)} \cdot log(n) \rfloor$.

We now focus our attention on STEP 5 of the algorithm. During STEP 5, the algorithm checks whether or not $(x + a)^n \equiv (x^n + a) \pmod{x^r - 1, n}$ holds for all $1 \leq a \leq l$. We begin our analysis by giving this property a name.

Definition 7.4. Let $f \in \mathbb{Z}[x]$ and $m \in \mathbb{N}$. Then m is said to be *introspective*[10] for f if:

$$f(x)^m \equiv f(x^m) \pmod{x^r - 1, p}$$

In Lemma 5.1 we saw that given a prime number p, and an integer $a \in \mathbb{Z}$ such that $(a, p) = 1$, p is introspective for the function $(x + a)$. We now use this fact in order

[10] We are able to work modulo p rather than modulo n because

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

to prove a somewhat surprising result; if we assume both n and p to be introspective for some linear polynomial $(x+a)$, then $\frac{n}{p}$ is introspective for $(x+a)$ as well.

Lemma 7.5. *Let $n \in \mathbb{N}$ have prime divisor p and let $a \in \mathbb{N}$ with $0 \leq a \leq l$. If n, p are introspective $(x+a)$, then $\frac{n}{p}$ is introspective for $(x+a)$ as well.*

Proof. By Lemma 9.13 of the Appendix, $f(x)^p = f(x^p)\ \forall f \in (\mathbb{Z}/p\mathbb{Z})[x]$. Also, as p, n are introspective for $(x+a)$, we have:

$$(x^{\frac{n}{p}} + a)^p \equiv (x^n + a) \equiv (x+a)^n \equiv (x+a)^{p \cdot \frac{n}{p}} \pmod{x^r - 1, p}.$$

Now define $f = (x^{\frac{n}{p}} + a)$ and $g = (x+a)^{\frac{n}{p}}$. We have just shown that $f^p = g^p$ in the ring $(\mathbb{Z}/p\mathbb{Z})[x]/(x^r - 1)$. Then $f^p + (-g)^p = 0$. Applying Lemma 9.12 from the Appendix on $f, (-g)$, we see that $(f-g)^p = 0$. Let $h := (f-g)$. It remains to show that $h = 0$ within $(\mathbb{Z}/p\mathbb{Z})[x]/(x^r - 1)$. By Lemma 9.11, $(r,p) = 1 \Rightarrow (x^r - 1)$ factors into distinct irreducibles $h_i(x)$ over $(\mathbb{Z}/p\mathbb{Z})[x]$, i.e., $x^r - 1 = \prod_i h_i(x)$. Therefore, by the Chinese Remainder Theorem we see that

$$h^p \in \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(x^r - 1)} = \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(\prod_i h_i(x))} = \prod_i \frac{(\mathbb{Z}/p\mathbb{Z})[x]}{(h_i(x))}.$$

Because $(x^r - 1) | h^p$, we see that each irreducible factor $h_i(x)$ of $(x^r - 1)$ divides h. Thus $(x^r - 1) | h$, completing the proof. □

We now show that introspective numbers are closed under multiplication and that the set of functions for which a given integer is introspective is closed under multiplication:

Lemma 7.6. *Let $f, g \in \mathbb{Z}[x]$, $s, t \in \mathbb{N}$. Then the following hold:*

(i) If s and t are introspective for f, then $s \cdot t$ is introspective for f.

(ii) If s is introspective for f and g, then s is introspective for $f \cdot g$.

Proof. (i): Because t is introspective for f, we have:

$$f(x)^{s \cdot t} = (f(x^t))^s \pmod{x^r - 1, p}.$$

Also, s is introspective for f, so:

$$f(x)^s \equiv f(x^s) \pmod{x^r - 1, p}$$

By substituting y^t for x in this identity, we see that:

$$f(y^t)^s \equiv f(y^{s \cdot t}) \pmod{y^{r \cdot t} - 1, p}.$$

Because $(y^r - 1)$ divides $(y^{r \cdot t} - 1) = (y^r - 1)(y^{r \cdot (t-1)} + y^{r \cdot (t-2)} + \cdots + y^r + 1)$, the above equation holds modulo $(y^r - 1, p)$, i.e., $f(y^t)^s \equiv f(y^{s \cdot t}) \pmod{y^r - 1, p}$. Putting these equations together yields the required result:

$$f(x)^{s \cdot t} = f(x^{s \cdot t}) \pmod{x^r - 1, p}.$$

(ii): Since s is introspective for f and g, we have:

$$[f(x) \cdot g(x)]^s = f(x)^s \cdot g(x)^s \equiv f(x^s) \cdot g(x^s) = (f \cdot g)(x^s) \pmod{x^r - 1, p}$$

□

Lemmas 7.5 and 7.6 show that every element of the set $I = \{(\frac{n}{p})^i \cdot p^j : i, j \geq 0\}$ is introspective for every polynomial in the set $P = \{\prod_{a=0}^{l} (x+a)^{e_a} : e_a \geq 0\}$. We now define two groups based on these sets.

Proposition 7.7. *$I_r = \{i \pmod r : i \in I\}$ is a multiplicative subgroup of $(\mathbb{Z}/r\mathbb{Z})^*$.*

Proof. First notice that I_r is closed under multiplication:

$$((\tfrac{n}{p})^{i_1} \cdot p^{j_1}) \pmod r \times ((\tfrac{n}{p})^{i_2} \cdot p^{j_2}) \pmod r = ((\tfrac{n}{p})^{i_1+i_2} \cdot p^{j_1+j_2}) \pmod r.$$

We have already seen that $(n, r) = (p, r) = 1$, so both n, p are units in $(\mathbb{Z}/r\mathbb{Z})$. Because I_r is generated by $n \pmod r$ and $p \pmod r$, we have $I_r \subset (\mathbb{Z}/r\mathbb{Z})^*$. □

Let $|I_r| = t$. Then $o_r(n) > log^2(n)$ implies that $t > log^2(n)$.

Let $\Phi_r(x)$ be the r^{th} cyclotomic polynomial over $\mathbb{F}_p$. Then $\Phi_r(x)|(x^r - 1)$ and by Lemma 9.11, $\Phi_r(x)$ factors into distinct irreducibles of degree $o_r(p)$. Let F be the splitting field of $x^r - 1$ over $\mathbb{F}_p$ and let $\zeta \in F$ be a primitive r^{th} root of unity with minimum polynomial $h(x) \in \mathbb{F}_p[x]$. Then $h(x)$ is an irreducible factor of $x^r - 1$ over $\mathbb{F}_p[x]$ of order $o_r(p)$. This implies that[11] $F = \mathbb{F}_p(\zeta) \cong \mathbb{F}_p[x]/(h(x))$.

Recall that $P = \{\prod_{a=0}^{l} (x+a)^{e_a} : e_a \geq 0\}$. Now define $G = \{f \pmod{h(x), p} : f \in P\}$. This group is clearly generated by the linear polynomials $x, x+1, x+2, ..., x+l$ within the field F.

Let $k = o_r(p)$. We have already seen that $k =$deg$(h(x))$, implying that $|F| = p^k$.

[11]Note that the map $\Psi : \mathbb{F}_p[x]/(h(x)) \longmapsto \mathbb{F}_p(\zeta)$ which sends $g(x)$ to $g(\zeta)$ is an isomorphism.

We can actually use this fact in order to provide an alternate proof of Lemma 7.5 in the case that $f(x) \in G$:

Lemma 7.8. *Let $f(x) \in G$.*

$$\textit{If } f(x)^n = f(x^n) \textit{ and } f(x)^p = f(x^p) \textit{ then } f(x)^{\frac{n}{p}} = f(x^{\frac{n}{p}}).$$

Proof. $|F| = p^k \Longrightarrow \forall\ \beta \in F,\ \beta^u = \beta^v$ whenever $u \equiv v \pmod{p^k - 1}$. Suppose that v is introspective for f. Then $f(\zeta)^u = f(\zeta)^v = f(\zeta^v) = f(\zeta^u)$. Introspective elements are closed under multiplication, so $f(x)^{n \cdot p^{k-1}} = f(x^{n \cdot p^{k-1}})$. Then the result holds as

$$n \cdot p^{k-1} \equiv \frac{n}{p} \pmod{p^k - 1}.$$

□

The next two lemmas provide us with an upper and lower bound for $|G|$. The first of these lemmas is due to Hendrik Lenstra Jr.[len], which not only avoided having to use a deep theorem from analytic number theory (see [fou] for the result), but lowered the complexity of the algorithm as well. Before stating the proof however, we ask the reader to recall that in Proposition 7.7 we defined a multiplicative group I_r (with $|I_r| = t$) and found it to be a subgroup of $(\mathbb{Z}/r\mathbb{Z})^*$.

Lemma 7.9. $|G| \geq \binom{t+l}{t-1}$

Proof. Note that because $h(x)$ is a factor of $\Phi_r(x)$, x is a primitive r^{th} root of unity in F. We now show that if $f, g \in P$ are distinct polynomials with degrees less than

t, then they map to distinct elements in G.

Suppose that $f(x) = g(x)$ in F. Let $m \in I$. Then m is introspective for f and g, so $f(x^m) = g(x^m)$ within F. Then x^m is a root of $j(z) = f(z) - g(z)$ for every $m \in I_r$. By Prop 7.2, $(m, r) = 1$, so each such x^m is a primitive r^{th} root of unity. Hence there are $|I_r| = t$ distinct roots of $j(z)$ in F. But the degree of $j(z) < t$ by the choice of f and g. This contradiction (a polynomial cannot have more roots in a field than its degree) implies that $f(x) \neq g(x)$ in F.

Notice that $i \neq j$ in $\mathbb{F}_p$ whenever $1 \leq i, j \leq l$ since $l = \lfloor \sqrt{\phi(r)} log(n) \rfloor < \sqrt{r} log(n) < r < p$. Then by above, $x, x+1, ..., x+l$ are all distinct in F. Since the degree of $h(x)$ is greater than 1, all of these linear polynomials are nonzero in F. Therefore there are at least $l + 1$ distinct polynomials of degree 1 in G. By Lemma 9.5 there are at least $\binom{l+s}{s}$ polynomials of degree s in G. Then by Lemma 9.6 the order of G is at least $\sum_{s=0}^{t-1} \binom{l+s}{s} = \binom{t+l}{t-1}$.

□

Lemma 7.10. *If n is not a power of p, then $|G| \leq n^{\sqrt{t}}$*

Proof. Consider the following subset of I:

$$\hat{I} = \{(\frac{n}{p})^i \cdot p^j : 0 \leq i, j \leq \lfloor \sqrt{t} \rfloor\}$$

If n is not a power of p, then $|\hat{I}| \geq (1 + \lfloor \sqrt{t} \rfloor)^2 > t$. Since $|I_r| = t$, there are at least two elements of $\hat{I}$ that are equivalent modulo r. Label these elements m_1, m_2 where

$m_1 > m_2$. Then

$$x^{m_1} \equiv x^{m_2} \pmod{x^r - 1}$$

Let $f(x) \in P$. Then because m_1, m_2 are introspective,

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

Thus $f(x)^{m_1} = f(x)^{m_2}$ in the field F. Therefore the polynomial $Q(y) = y^{m_1} - y^{m_2}$ has at least $|G|$ roots in F (since $f(x) \in P$ was arbitrary). Then because $\frac{n}{p}\lfloor\sqrt{t}\rfloor p^{\lfloor\sqrt{t}\rfloor}$ is the largest element of $\hat{I}$,

$$deg(Q(y)) = m_1 \leq (\tfrac{n}{p} \cdot p)^{\lfloor\sqrt{t}\rfloor} = n^{\lfloor\sqrt{t}\rfloor}.$$

It follows that $|G| \leq n^{\sqrt{t}}$. □

We are now ready to finish our correctness proof of the algorithm.

Lemma 7.11. *If the algorithm returns PRIME then n is prime.*

Proof. Assume that the algorithm returns prime. Recall that $|I_r| = t$ and is generated by n and p. Therefore $t \geq o_r(n) > log^2(n) \Rightarrow t^2 > t \cdot log^2(n) \Rightarrow t > \lfloor\sqrt{t}log(n)\rfloor$. Then Lemma 7.9 shows that for $l = \lfloor\sqrt{\varphi(r)}log(n)\rfloor$:

$$|G| \geq \binom{t+l}{t-1} \geq \binom{l+1+\lfloor\sqrt{t}log(n)\rfloor}{\lfloor\sqrt{t}log(n)\rfloor} \qquad (\text{since } t > \lfloor\sqrt{t}log(n)\rfloor)$$

$$\geq \binom{2\lfloor\sqrt{t}log(n)\rfloor+1}{\lfloor\sqrt{t}log(n)\rfloor} \qquad (\text{since } I_r \subset Z_r^* \Rightarrow t \leq \varphi(r) \Rightarrow l = \lfloor\sqrt{\varphi(r)}log(n)\rfloor \geq \lfloor\sqrt{t}log(n)\rfloor)$$

$$> 2^{\lfloor\sqrt{t}log(n)\rfloor+1} \geq 2^{\sqrt{t}log(n)} = n^{\sqrt{t}} \qquad \text{(by Lemma 9.2 of the Appendix)}$$

Then by Lemma 7.10, $|G| \leq n^{\sqrt{t}}$ if p is not a power of p. Therefore it must be the case that $n = p^k$ for some $k > 0$. But STEP 1 did not output COMPOSITE, so $k = 1$, proving that n is indeed prime. □

This completes our proof of Theorem 7.1 □

8 Runtime Analysis

STEP 1

We have shown in Section 4.4 that STEP 1 will take at most $O^{\sim}(log^3(n))$ bit operations.

STEP 2

In this step the algorithm finds the least r such that $o_r(n) > log^2(n)$. By Lemma 4.2, there exists such an r less than $\lceil log^5(n) \rceil$. The easiest way to find such an r is simply to calculate $n^k \pmod r$ for $k = 1, 2, ..., log^2(n)$. This involves $O(log^2(n))$ multiplications modulo r for each r, so STEP 2 takes $O^{\sim}(log^7(n))$ bit operations.

STEP 3

In this step the algorithm computes (a, n) for $a = 1, ..., r$ in order to determine whether $(a, n) > 1$ for some $a \leq r$. Computing each gcd takes $O^{\sim}(log^2(n))$ bit operations using the Euclidean Algorithm, resulting in a total of $O^{\sim}(log^7(n))$ bit operations taken during STEP 3.

STEP 5

During this step the algorithm determines whether the congruence

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

holds for $a = 1, 2, 3, ..., \lfloor \sqrt{\varphi(r) \cdot log(n)} \rfloor$. Given $a \leq \lfloor \sqrt{\varphi(r) \cdot log(n)} \rfloor$, we may calculate $(x + a)^n$ in the ring $\mathbb{Z}/n\mathbb{Z}$ as reducing modulo $x^r - 1$ is trivial (simply replace x^s

by x^{s-r}). In order to calculate $(x+a)^n$, we must perform $O(log(n))$ multiplications of polynomials of degree less than r with coefficients of size $O(log(n))$ (as the coefficients are in written modulo n; recall that all polynomials are reduced modulo $x^r - 1$ during Fast Modular Exponentiation). Each congruence therefore takes $O^{\sim}(log^7(n))$ bit operations to verify. STEP 5 therefore takes $O^{\sim}(\sqrt{\varphi(r)}log(n)log^7(n)) = O^{\sim}(\sqrt{\varphi(r)}log^8(n)) = O^{\sim}(log^{\frac{21}{2}}(n))$ bit operations. The complexity of STEP 5 clearly dominates the complexity of the other steps, so the overall complexity of the algorithm is $O^{\sim}(log^{10.5}(n))$, which is indeed polynomial.

9 Appendix

9.1 Results from Number Theory and Combinatorics

We begin with Nair's proof that the least common multiple of the first n integers is at least 2^n (See [nai] for original proof).

Lemma 9.1. *Let $n \in \mathbb{N}$ with $n \geq 7$. Then $d_n := lcm([n]) \geq 2^n$*

Proof. Direct calculation[12] shows that the Lemma holds when $n = 7$ or $n = 8$.We may therefore assume that $n \geq 9$.

Consider the integral

$$I(m,n) = \int_0^1 x^{m-1}(1-x)^{n-m}dx \qquad (1 \leq m \leq n).$$

We know that the binomial expansion of $(1-x)^{n-m} = \sum_{k=0}^{n-m}(-1)^k\binom{n-m}{j}x^k$, which gives us:

$$\int_0^1 x^{m-1}(1-x)^{n-m}dx = \int_0^1 x^{m-1}\sum_{k=o}^{n-m}(-1)^k x^k\binom{n-m}{k}dx = \sum_{k=0}^{n-m}(-1)^k\int_0^1 x^{m+k-1}dx.$$

Therefore $I(m,n) = \sum_{k=0}^{n-m}(-1)^k\binom{n-m}{k}\frac{1}{m+k}$. Because $0 \leq k \leq n-m$, $m+k$ divides d_n. Thus $I(m,n)\cdot d_n \in \mathbb{Z}$.

[12] If $n = 7$ then $d_7 = 420 > 2^7 = 128$. If $n = 8$ then $d_8 = 840 > 2^8 = 256$.

Alternatively, iterated integration by parts yields:

$$I(m,n) = \frac{(n-m)!}{n\cdot(n-1)\cdots m} = \frac{(n-m)!(m-1)!}{n!} = \frac{1}{m\binom{n}{m}}.$$

We have already seen that $I(m,n)\cdot d_n \in \mathbb{Z}$, so $m\binom{n}{m}|d_n$ for all $1 \leq m \leq n$. From this we may conclude that $n\binom{2n}{n}|d_{2n}|d_{2n+1}$ (since $d_{n-1}|d_n$) and that

$$(n+1)\binom{2n+1}{n+1} = (n+1)\binom{2n+1}{n} = (2n+1)\binom{2n}{n}|d_{2n+1}.$$

As $(n, 2n+1) = 1$, it follows that $n(2n+1)\binom{2n}{n}|d_{2n+1}$. Because $\binom{2n}{n}$ is the largest of the binomial coefficients occurring in the expansion of $(1+1)^{2n}$, $d_{2n+1} \geq n4^n$ (for $n \geq 1$). If $n \geq 2$ then $d_{2n+1} \geq 2\cdot 4^n = 2^{2n+1}$ and in the case that $n \geq 4$ we have $d_{2n+2} \geq d_{2n+1} \geq 4^{n+1}$. We have shown that $d_n \geq 2^n$ when $n \geq 9$, and since we have already calculated d_7 and d_8, we're done. □

Lemma 9.2. *$\forall n \in \mathbb{N}$, we have $\binom{2n+1}{n} > 2^{n+1}$*

Proof. By definition, $\binom{2n+1}{n} = \frac{(2n+1)\cdots(n+2)}{n!} = \prod_{i=1}^{n}\frac{2i+1}{i} \geq \prod_{i=1}^{n} 2 = 2^n$. We notice that $\prod_{i=1}^{3}\frac{(2i+1)}{i} = \frac{35}{2} > 2^4$, from which we conclude that $\binom{2n+1}{n} \geq 2^{n+1}$. □

We now focus our attention on proving a result from Combinatorics (This proof is due to Richard Stanley. See Stanley's Enumerative Combinatorics [stan] for more details). We begin with two definitions:

Definition 9.3. A *k-subset* of $[n]$ is a set of distinct elements of $[n]$ having cardinality

k. It is well known that the number of k-subsets of $[n]$ is equal to $\binom{n}{k}$.

Definition 9.4. A *k-multiset* of $[n]$ is a set of k elements of $[n]$, in which repetition is allowed.

For example, $\{1,2,3\} = \{2,1,3\}$ are equivalent as multisets of $[3]$, while $\{1,2,3\} \neq \{1,1,2,3\}$, as the former has cardinality 3, while the latter has cardinality 4.

Let $f(k,n)$ denote the number of k-multisets of $[n]$. We will now prove that $f(k,n) = \binom{n+k-1}{k}$.

Lemma 9.5. *The number of k-multisets of $[n]$ is equal to $\binom{n+k-1}{k}$.*

Proof. Let $1 \leq a_1 \leq a_2 \leq \cdots a_k \leq n+k-1$ be a k-subset of $[n+k-1]$. Define $b_i := a_i - i + 1$. Then $\{b_1, ..., b_k\}$ is a k-multiset on $[n]$. Conversely, given a k-multiset on $[n]$: $1 \leq b_1 \leq b_2 \leq \cdots b_k \leq n$, we can define $a_i = b_i + i - 1$ in order to get a k-subset of $[n+k-1]$. Therefore, the number of k-combinations of $[n]$ with repetition is simply the number of k-subsets of $[n+k-1]$, i.e., $\binom{n+k-1}{k}$. □

Lemma 9.6. *Let $l, t \in \mathbb{Z}$. Then $f(t-1, l+2) = \sum_{k=0}^{t-1} f(k, l+1)$.*

Proof.

Let A be the set of all $(t-1)$-multisets of $[l+2]$, and B be the set consisting of all i-multisets of $[l+1]$ for $i = 0, 1, ..., (t-1)$. Then $|A| = f(t-1, l+2)$ and $|B| = \sum_{k=0}^{t-1} f(k, l+1)$. We now exhibit a bijection between A and B.

Let $A_0 \in A$. If $(l+2) \notin A_0$ then A_0 is a $(t-1)$-multiset of $[l+1]$ and thus an element of B. If $(l+2) \in A_0$, then let a_{l+2} denote its multiplicity in A_0. Then

$A_0 - \{\underbrace{(l+2),(l+2),...,(l+2)}_{a_{l+2}times}\}$ is an element of B.

Now let B_0 be an element of B of cardinality k (i.e., B_0 is a k-multiset of $[l+1]$). If $k = (t-1)$ then $B_0 \in A$. If $k \neq (t-1)$, then $B_0 \cup \{\underbrace{(l+2),(l+2),...,(l+2)}_{(t-1-k)times}\}$ is an element of A.

These two operations are clearly inverses of each other, concluding our proof. $\square$

9.2 Cyclotomic Polynomials and Finite Fields

We now prove some results concerning finite fields and cyclotomic polynomials that will be necessary to show the correctness of the AKS Primality-Testing Algorithm (see Dummit and Foote [df] for a more complete treatment).

Let μ_n denote the group of n^{th} roots of unity over $\mathbb{Q}$. We know that as a group, $\mu_n \cong \mathbb{Z}/n\mathbb{Z}$ because of the map $a \mapsto \zeta_n^a$ where ζ_n is a fixed primitve n^{th} root of unity. Primitive roots of unity are given by residue classes relatively prime to n, meaning that there are $\phi(n)$ primitive n^{th} roots of unity.

Definition 9.7. The n^{th} *cyclotomic polynomial* $\Phi_n(x) \in \mathbb{C}[x]$ is the polynomial whose roots are precisely the $\varphi(n)$ primitive n^{th} roots of unity in $\mathbb{C}$:

$$\Phi_n(x) = \prod_{\zeta primitive \in \mu_n} (x - \zeta) = \prod_{(a,n)=1} (x - \zeta_n^a)$$

In particular note that the degree of $\Phi_n(x) = \varphi(n)$.

The roots of $(x^n - 1)$ are precisely the n^{th} roots of unity, so $x^n - 1 = \prod_{\zeta \in \mu_n} (x - \zeta)$. Grouping the factors $(x - \zeta)$ according to their order in μ_n yields (by Lagrange's Theorem):

$$x^n - 1 = \prod_{d|n} \prod_{\zeta primitive \in \mu_d} (x - \zeta) = \prod_{d|n} \Phi_d(x)$$

It turns out that for all $n \in \mathbb{N}$, $\Phi_n(x)$ has integer coefficients. In order to prove this however, we will need an important result regarding polynomial factorizations in $\mathbb{Z}[x]$.

Definition 9.8. We call a polynomial $f(x) \in \mathbb{Z}[x]$ *primitive* if

$$f(x) = a_n x^n + \cdots a_1 x + a_0, \qquad gcd(a_n, a_{n-1}, ..., a_1, a_0) = 1.$$

Lemma 9.9. *(Gauss' Lemma) Let $p(x) \in \mathbb{Z}[x]$ be primitive.*

(i) If $q|f(x) \cdot g(x)$ where q is a prime integer and $f(x), g(x) \in \mathbb{Z}[x]$, then either $q|f(x)$ or $q|g(x)$ in $\mathbb{Z}[x]$.

(ii) If $p(x)$ is reducible in $\mathbb{Q}[x]$ then it is reducible in $\mathbb{Z}[x]$.

Proof. Consider the ring $\mathbb{F}_q[x]$. Then q divides $f \cdot g$ is equivalent to saying that $\overline{f \cdot g} = 0$ ($\overline{f \cdot g}$ is the image of $f \cdot g$ in $\mathbb{F}_q[x]$). Reduction modulo q is a homomorphism, so $\overline{f \cdot g} = \overline{f} \cdot \overline{g}$. But $\mathbb{F}_q[x]$ is an integral domain[13], so it must be the case that either $\overline{f} = 0$ or $\overline{g} = 0$. This proves (i).

To prove (ii), suppose that $f|p$ in $\mathbb{Q}[x]$, where f has integer coefficients. Then $\exists g \in \mathbb{Q}[x]$ such that $p = f \cdot g$. Clearing the denominator of the right hand side gives us $d \cdot p = f \cdot g$ where $d \in \mathbb{Z}$, $d \cdot p \in \mathbb{Z}[x]$. Let p_0 be some prime factor of d. Then by (i), p_0 divides f or g. But f was assumed to be primitive, so p_0 divides g. Then by induction, we see that $d|g$. Therefore $p(x) = f(x) \cdot \frac{h(x)}{d}$. □

We are now prepared to prove that $\Phi_n(x)$ has integer coefficients.

Proposition 9.10. $\Phi_n(x) \in \mathbb{Z}[x]$.

[13]Recall that $R[x]$ is an integral domain if and only if R is an integral domain.

Proof. We already know that $\Phi_n(x)$ is monic of degree $\varphi(n)$. We proceed by induction. In the case that $n = 1$, the result is clear. So let $n > 1$ and assume that $\Phi_d(x) \in \mathbb{Z}[x]$ for all $1 \leq d < n$. Then

$$x^n - 1 = \Phi_n(x)f(x), \qquad f(x) = \prod_{\substack{d|n \\ d<n}} \Phi_d(x).$$

$f(x)$ clearly divides $x^n - 1$ in $\mathbb{Q}(\zeta_n)[x]$, and because both polynomials have coefficients in $\mathbb{Q}$, we have that $f(x)$ divides $x^n - 1$ in $\mathbb{Q}[x]$[14]. Then by Gauss' Lemma, $f(x)$ divides $x^n - 1$ in $\mathbb{Z}[x]$, hence $\Phi_n(x) \in \mathbb{Z}[x]$. □

Lemma 9.11. *Let $n \in \mathbb{N}$ and q be prime with $(q, n) = 1$. Then $\Phi_n(x)$ factors into the product of $\frac{\varphi(n)}{o_n(q)}$ distinct monic irreducible polynomials in $\mathbb{F}_q[x]$.*

Proof. We begin by showing that $o_n(q)$ is that smallest integer such that $\zeta_n \in \mathbb{F}_{q^{o_n(q)}}$, where ζ_n is a primitive n^{th} root of unity. Indeed, $\zeta_n \in \mathbb{F}_{q^k}$ if and only if $\zeta_n^{q^k} = \zeta_n$ which is equivalent to the identity $q^k \equiv 1 \pmod{n}$. The smallest integer k for which this holds is obviously $k = o_n(q)$. Thus ζ_n lies in $\mathbb{F}_{q^{o_n(q)}}$ but no proper subfield thereof. Therefore the degree of the minimal polynomial of ζ_n over $\mathbb{F}_q$ is $o_n(q)$. As the degree of $\Phi_n(x)$ is $\varphi(n)$, it follows that Φ_n factors into $\frac{\varphi(n)}{o_n(q)}$ monic irreducible polynomials in $\mathbb{F}_q[x]$.

We now show that these polynomials are distinct. Let $\overline{\Phi_n}(x)$ be the image of $\Phi_n(x)$ in $\mathbb{F}_q[x]$, and let $\overline{\Phi_n'}(x)$ be its formal derivative. Suppose that $\overline{g}(x)$ is irreducible in

[14]Since the division algorithm is independent of field extensions in the sense that it is unique within polynomial rings of the for $F[x]$ where F is a field.

$\mathbb{F}_q[x]$ and $\overline{g}(x)^2|\overline{\Phi_n}(x)$. Then $gcd(\overline{\Phi_n'}(x), \overline{\Phi_n}(x)) \neq 1$. But $\overline{\Phi_n}(x)|(x^n-1)$, so $gcd((x^n-1), nx^{n-1}) \neq 1$. But $n \neq 0$ in $\mathbb{F}_q$, so $gcd((x^n-1), nx^{n-1}) = gcd((x^n-1), x^{n-1}) = 1$. Contradiction. Therefore $g(x)^2 \not| \overline{\Phi_n}(x)$, meaning that all of the irreducible factors of $\overline{\Phi_n}(x)$ are distinct. □

Lemma 9.12. *$(f+g)^p = f^p + g^p$ for all $f, g \in \mathbb{F}_p[x]$.*

Proof. By the Binomial Theorem, $(f+g)^p = f^p + \sum_{k=1}^{p-1} \binom{p}{k} f^k \cdot g^{p-k} + g^p$. Every term in the summation is divisible by p, and therefore equal to zero within the ring $\mathbb{F}_p[x]$. Therefore $(f+g)^p = f^p + g^p$. □

Lemma 9.13. *Let $f \in \mathbb{F}_p[x]$ for some prime p. Then $f(x)^p = f(x^p)$.*

Proof. Let $f(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. Then by Lemma 1 we have

$$f(x)^p = (a_n)^p (x^n)^p + (a_{n-1})^p (x^{n-1})^p + \cdots + (a_1)^p (x)^p + (a_0)^p.$$

By Fermat's Little Theorem, it follows that

$$f(x)^p = a_n (x^p)^n + a_{n-1}(x^p)^{n-1} + \cdots + a_1 x^p + a_0 = f(x^p).$$

□

References

[aks] M. Agrawal, N. Kayal and N. Saxena, PRIMES is in P, Ann. of Math. (2) 160 (2004), 781-793.

[bro] D. R. Heath-Brown and J.-C. Puchta, Integers represented as a sum of primes and powers of two, Asian J. Math. 6 (2002), 535-565.

[chen] J. R. Chen, On the representation of a large even integer as the sum of a prime and the product of at most two primes, Sci. Sinica 16 (1973), 157176.

[cra] R. Crandall and J. Papadopoulos, On the Implementation of AKS-Class Primality Tests, 18 Mar 2003, http://developer.apple.com/hardware/ve/pdf/aks3.pdf.

[diet] M. Dietzfelbinger, Primality Testing in Polynomial Time, Springer, 2004.

[df] D. Dummit and R. Foote, Abstract Algebra, 2nd ed, John Wiley and Sons, Inc., 1999.

[euc] Euclid, The Elements, 300 BC

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

[len] H. W. Lenstra, Jr. Primality testing with cyclotomic rings, Unpublished, August 2002.

[gran] Alford, Granville and Pomerance, There are infinitely many Carmichael numbers, Ann. of Math. 140:3 (1994), 703-722.

[nai] M. Nair, On Chebyshev-type inequalities for primes, Amer. Math. Monthly 89 (1982), 126-129.

[ss] A. Schnhage and V. Strassen, Schnele multiplikation grosser zahlen, Computing 7 (1971), 281-292.

[ss2] R. Solovay and V. Strassen, A fast Monte-Carlo test for primality, SIAM Journal on Computing 6:1 (1977), 84-85.

[stan] R. Stanley, Enumerative Combinatorics Volume I, Cambridge University Press, 1997.

[vau] H. L. Montgomery and R. C. Vaughan, The exceptional set in Goldbachs problem, Acta Arith. 27 (1975), 353370.

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