Primality Testing

Theory, Complexity, and Applications

Riley Worthington

Whitman College

May 11, 2018

Abstract

Primality testing is the problem of deciding whether a given number n is prime. Efficient primality tests are needed for generating keys used in many modern cryptographic systems. Until recently, no such algorithm was known that was general, deterministic, unconditional, and polynomial time. With the publication of *PRIMES is in P* in 2002, computer scientists Agrawal, Kayal, and Saxena showed that an algorithm with all four properties does indeed exist. In this paper we present a background of primality testing, as well as an exposition of the AKS algorithm and its proof of correctness.

Contents

1 Introduction

We say an integer $n \geq 2$ is **prime** if its only positive divisors are 1 and itself. Euclid, in his book *Elements* (circa 300 BC) was the first to record such a definition, albeit with slightly more cryptic wording:

A prime number is that which is measured by a unit alone.

Euclid also proved that there are infinitely many primes. Another Greek mathematician, Eratosthenes, proposed an early method for finding them. His technique, called the Sieve of Eratosthenes, was to first form a list of all the integers from 2 up to a certain n.

$$2,\ 3,\ 4,\ 5,\ 6,\ 7,\ 8,\ 9,\ 10,\ 11\ldots,\ n$$

One could then proceed by finding the smallest number in the list (2 to begin with) and crossing out all of its multiples (4, 6, 8, etc.).

$$2,\ 3,\ \cancel{4},\ 5,\ \cancel{6},\ 7,\ \cancel{8},\ 9,\ \cancel{10},\ 11\ldots,\ n$$

The process is repeated with the next uncrossed number in the list, 3.

$$2,\ 3,\ \cancel{4},\ 5,\ \cancel{6},\ 7,\ \cancel{8},\ \cancel{9},\ \cancel{10},\ 11\ldots,\ n$$

The numbers that remain uncrossed at the end are the primes.

Many of the further advances in the subject are owed to the groundwork in number theory done in the 17th and 18th centuries by mathematicians Fermat,

Euler, and Mersenne. Numbers of the form $2^n - 1$ are called Mersenne numbers after the latter, and 50 of them are known to be primes as of the writing of this paper. In fact, the largest known prime as discovered in December 2017 is a Mersenne prime: $2^{77,232,917} - 1$.

Unfortunately, Eratosthenes' sieve method would take longer than the age of the sun to verify the above number. Though it is simple and easy to carry out by hand, it is only useful for relatively small inputs. Due to its time and space requirements, it becomes wildly inefficient as the numbers grow. Indeed, before the advent of computers, no one was trying to find large primes anyway. Regardless of impracticality, there simply wasn't a reason to.

It would later turn out that in addition to their great importance to number theory and other areas of mathematics, prime numbers have practical applications as well. Many cryptographic systems that we use daily to keep information secure on the internet require large prime numbers to operate. Thus with modern cryptography came the need for *efficient* primality tests. Many algorithms have been proposed, but almost all of them fail to have at least one of the following desired characteristics:

General. An algorithm that is *general* works for all numbers. Algorithms that are not general only work on numbers of a certain form, such as the Lucas-Lehmer test for Mersenne numbers.

Deterministic. A *deterministic* algorithm gives a definitive result every time it is run. The opposite of deterministic is *probabilistic*, which gives an answer with some probability of being correct. For example, the Miller-Rabin test can correctly identify a number as composite over 75% of the time. Such tests rely on multiple runs to gain a greater certainty of the result.

Unconditional. An *unconditional* algorithm is one whose correctness does not depend on any unproven hypotheses. For example, there are *conditional* primality tests that are correct only if the Extended Riemann Hypothesis is true.

Polynomial Time. A *polynomial time* algorithm is one with computational complexity that is a polynomial function of the input size. For primality testing, we measure the input size as the number of bits needed to represent the number. Therefore a polynomial time algorithm will have complexity that is a polynomial function of $\log_2 n$.

In computer science, a distinction is made between problems that can be solved in polynomial time in a deterministic way and those that cannot. The former is referred to as class **P** complexity. For quite some time, it was unknown whether or not the problem of determining a number to be prime was in **P**. Finally, in 2002 it was shown by Indian computer scientists Agrawal, Kayal, and Saxena that an algorithm fulfilling all four properties is in fact possible.

Their algorithm, AKS, is the focus of Chapter 6. We preclude its presentation with a few examples of primality tests that satisfy some, but not all of the above properties, including both deterministic and probabalistic approaches. We provide the lower level number theory algorithms that allow these tests to function. Finally, we conclude with a discussion of the theoretical implications versus the practical usage of these tests. All algorithms mentioned in this paper been implemented in Python, with code made available at https://github.com/worthirj/primality-testing.

2 Preliminaries

We assume a knowledge of basic number theory and abstract algebra. The following sections cover a brief review of these topics and introduce some of the notation used throughout the paper.

2.1 Review of Number Theory

Important definitions and results in number theory are listed here for reference. Proofs are not given, see for example *An Introduction to Higher Mathematics* by Patrick Keef and David Guichard.

Definition 2.1. An integer $p > 1$ is **prime** if it has only two positive divisors, 1 and itself. An integer $a > 1$ with more than two positive divisors is **composite**.

Definition 2.2. Let a and b be integers. The **greatest common divisor** of a and b, denoted $\gcd(a, b)$, is the largest positive integer n for which $n \mid a$ and $n \mid b$.

Theorem 2.3. Let a and b be nonzero integers. Then there exist integers x and y such that $\gcd(a, b) = ax + by$.

Definition 2.4. Two integers a and b are **relatively prime** if and only if $\gcd(a, b) = 1$.

Definition 2.5. Let a, b, and n be integers. Then a is **congruent** to b modulo n if and only if $n \mid a - b$. We denote this as $a \equiv b \pmod{n}$.

Definition 2.6. Let a and n be integers. The **multiplicative inverse** of a modulo n is an integer b such that $ab \equiv 1 \pmod{n}$.

Theorem 2.7. An integer a has a multiplicative inverse modulo n if and only if a and n are relatively prime. The set $\mathbb{U}_n \subseteq \mathbb{Z}_n$ denotes the set of elements in $\mathbb{Z}_n$ that are relatively prime to n, i.e. have multiplicative inverses.

Definition 2.8. Given $r \in \mathbb{N}$ and an integer n with $\gcd(n, r) = 1$, we define the **order** of n modulo r as the smallest number k such that $n^k \equiv 1 \pmod{r}$, and denote it as $\text{ord}_r(n)$.

Note that as a consequence of Theorem 2.7, $\text{ord}_r(n)$ is defined if and only if $\gcd(n, r) = 1$. As an example of what can happen if n and r are not relatively prime, take $n = 4$ and $r = 6$. Listing out the powers of 4 (mod 6), we have

k	$4^k \pmod{6}$
1	4
2	4
3	4
4	4
5	4
6	4

No value of $k > 0$ can cause $4^k \equiv 1 \pmod{6}$ so 4 does not have an order modulo 6.

Theorem 2.9 (Fundamental Theorem of Arithmetic). Every integer $n > 1$ is either a prime number or can be factored into the product of prime numbers. Such factorizations are unique.

We typically use the notation $n = {p_1}^{e_1}{p_2}^{e_2} \cdots {p_k}^{e_k}$ to denote the prime factorization of n, where $p_1, p_2, \ldots, p_k$ are distinct primes and the exponents $e_1, e_2, \ldots, e_k$ are greater than 0.

Definition 2.10. Let $n > 1$ be an integer. **Euler's Phi Function**, denoted $\phi(n)$, is the number of integers less than n and greater than or equal to 0 that are relatively prime to n. Note that $|\mathbb{U}_n| = \phi(n)$. Letting $n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k}$ be the prime factorization of n, this quantity is given by the formula

$$\phi(n) = n \prod_{i=1}^{k} \left(1 - \frac{1}{p_i}\right).$$

For example, $\phi(9) = 6$ since 1, 2, 4, 5, 7, and 8 are all relatively prime to 9. Not included are 3 and 6 since they share the common factor of 3.

Note that if p is a prime, $\phi(p) = p - 1$ since all integers less than p are relatively prime to p.

Theorem 2.11 (Euler's Theorem). If n and a are positive integers and $\gcd(a, n) = 1$, then $a^{\phi(n)} \equiv 1 \pmod{n}$.

Theorem 2.12 (Fermat's Little Theorem). Let a be an integer and p be a prime. Then

a) $a^{p-1} \equiv 1 \pmod{p}$ if a and p are relatively prime,

b) $a^p \equiv a \pmod{p}$ for all a.

2.2 Review of Abstract Algebra

We assume a knowledge of the basic definitions of groups, rings, fields and their properties. Results that will be referenced later are cited here. We also develop a bit of theory about polynomials that will be used to prove the correctness of the AKS algorithm. For a more complete treatment, see any standard algebra textbook.

Theorem 2.13 (Binomial Theorem). Let R be a commutative ring with identity, n an integer, and $a, b \in R$. Then

$$(a+b)^n = \sum_{k=0}^{n} \binom{n}{k} a^k b^{n-k}.$$

Theorem 2.14 (The Freshman's Dream). Let R be a commutative ring of prime characteristic p. Let $a, b \in R$. Then

$$(a+b)^p = a^p + b^p.$$

Proof. By the binomial theorem, we have

$$(a+b)^p = \sum_{k=0}^{p} \binom{p}{k} a^k b^{p-k}.$$

Since $\binom{p}{k} = \frac{p!}{k!(p-k)!}$, we see that p divides the numerator but not the denominator for $1 \leq k \leq p-1$ because p is a prime and k, $p-k$ are both less than p. Hence the coefficients on all the terms are congruent to 0 modulo p except the first and last, which are congruent to 1, giving us $(a+b)^p \equiv a^p + b^p \pmod{p}$. Therefore $(a+b)^p = a^p + b^p$ in R. □

2.2.1 Polynomial Rings

The idea of polynomials from calculus can be abstracted to any ring.

Theorem 2.15. Let R be a ring and let $R[X]$ denote the set of all sequences of elements in R $(a_0, a_1, \dots)$ such that $a_i = 0$ for all but a finite number of indices i. Then $R[X]$ is a ring with addition and multiplication defined by

$$\begin{aligned}(a_0, a_1, \dots) + (b_0, b_1, \dots) &= (a_0 + b_0, a_1 + b_1, \dots) \\ (a_0, a_1, \dots) \cdot (b_0, b_1, \dots) &= (c_0, c_1, \dots),\end{aligned}$$

where $c_n = \sum_{i=0}^{n} a_{n-i} b_i$. If R is commutative then so is $R[X]$.

For convenience, we write $(a_0, a_1, \ldots, a_d)$ for the sequence $(a_0, a_1, \ldots)$ if $a_d \neq 0$ and $a_i = 0$ for all $i > d$. That is, the sequences $(0, 2, 5, 0, 0, \ldots)$ and $(0, 2, 5)$ denote the same polynomial.

We already have some intuition about $R[X]$ from calculus. We can assign the formal symbol X as a placeholder to write the elements of $R[X]$ as we are used to seeing them. That is,

$$(a_0, a_1, a_2, a_3, \ldots, a_d) = a_d X^d + \cdots + a_3 X^3 + a_2 X^2 + a_1 X + a_0.$$

If $a_i = 0$, the convention is to omit the X^i term. For example, we write $(1, 2, 0, 3, 0, 4) = 4X^5 + 3X^3 + 2X + 1$.

As in calculus, the highest power of X that appears in f is the **degree** of the polynomial and is denoted $\deg(f)$. For a polynomial $f \neq 0$ with degree d, we call a_d the **leading coefficient** of f. A polynomial $f \neq 0$ is **monic** if the leading coefficient is 1. For example, $f(X) = X^5 + 2X^4 + 3X^2 + 7X + 2$ is monic and has degree 5.

Example. $\mathbb{Z}[X]$ is the ring of polynomials with integer coefficients. $\mathbb{Z}_n[X]$ is the ring of polynomials with integers in $\mathbb{Z}_n$ (operations on coefficients are carried out modulo n).

As Theorem 2.15 suggests, we can think of addition and multiplication of two polynomials as a series of operations on their coefficients. When doing these operations on a computer, it is straightforward to represent a polynomial as an array of its coefficients. We can also develop a concept of division and modular arithmetic for polynomials analogous to the integers.

Theorem 2.16. Let R be a ring with identity, and let $f, g \in R[X]$ be nonzero polynomials such that the leading coefficient of g is a unit in R. Then there

exist unique polynomials $q, r \in R[X]$ such that

$$f = qg + r \quad \text{with } \deg(r) < \deg(g).$$

Definition 2.17. For $f, h \in R[X]$, we say that h **divides** f if $f = h \cdot q$ for some $q \in R[X]$. If $0 < \deg(h) < \deg(f)$, then h is called a **proper divisor** of f.

Definition 2.18. Let $h \in R[X]$ be a polynomial whose leading coefficient is a unit. For $f, g \in R[X]$ we say that f and g are **congruent** modulo h, and write $f \equiv g \pmod{h}$, if $f - g$ is divisible by h.

Example. We can visualize division with remainder through polynomial long division, a process essentially the same as that done in elementary school with integers. For example:

$$\begin{array}{rr|l} & & 6X - 2 \\ \hline X^2 - 1 & 6X^3 - 2X^2 + X + 3 & \\ & -6X^3 \qquad + 6X & \\ \hline & -2X^2 + 7X + 3 & \\ & 2X^2 \qquad - 2 & \\ \hline & 7X + 1 & \end{array}$$

We get that $(6X^3 - 2X^2 + X + 3) \,/\, (X^2 - 1) = (6X - 2)(X^2 - 1) + (7X + 1)$, and write

$$(6X^3 - 2X^2 + X + 3) \equiv (7X + 1) \pmod{X^2 - 1}.$$

2.2.2 Quotient Rings of Polynomials

We are now ready to develop the notion of quotient rings of polynomials which is a central component of the AKS algorithm. As with the integers taken

modulo some m, we can take the elements of $R[X]$ modulo some polynomial h and form a ring from the remainders.

Theorem 2.19. Let R be a ring with identity and $h \in R[X]$ be a nonzero polynomial with unit leading coefficient. Let $d = \deg(h)$. Define $R[X]/h(X)$ as the set of all polynomials in $R[X]$ with degree strictly less than d, with the operations $+$ and $\cdot$ taking place modulo $h(X)$. Then $R[X]/h(X)$ is also a ring with identity.

The elements of $R[X]/h(X)$ can be thought of as the elements in $R[X]$ modulo $h(X)$. Let $f, g \in R[X]$. Then if $f \equiv g \pmod{h(X)}$, we say $f = g$ in the ring $R[X]/h(X)$ (under the map $f \mapsto f \pmod{h(X)}$). If $R = \mathbb{Z}_n$, then two polynomials are equal in $\mathbb{Z}_n[X]/h(X)$ if they are equivalent modulo $h(X)$, n.

Example. Consider $R = \mathbb{Z}_5[X]/(X^2 - 1)$, the ring of polynomials with coefficients in $\mathbb{Z}_5$ modulo $X^2 - 1$. Any element of $\mathbb{Z}[X]$ can be naturally mapped to R by the mapping $f \mapsto f \pmod{X^2 - 1, 5}$. For example, $X^3 + 5X^2 + 2X + 1 \mapsto 3X + 1$.

We now turn our attention to polynomials over fields, beginning with a definition.

Definition 2.20. Let F be a field, and let F^* denote $F - \{0\}$. A nonzero polynomial $f \in F[X]$ is **irreducible** if it does not have a proper divisor. That is, if $f = g \cdot h$ for some $g, h \in F[X]$ it follows that $g \in F^*$ or $\deg(g) = \deg(f)$.

Theorem 2.21 (Unique Factorization for Polynomials). Let F be a field. Then every nonzero polynomial $f \in F[X]$ can be written as a product of $a \cdot h_1 \cdots h_s$, $s \geq 0$, where $a \in F^*$ and $h_1, \ldots, h_s$ are monic irreducible polynomials in $F[X]$ of degree greater than 0. This product representation is unique up to the order of the factors.

Theorem 2.22. Let F be a field, and let $h \in F[X]$ be a monic irreducible polynomial over F. Then $F[X]/h(X)$ is also a field.

Finally, we can also extend the concept of roots of polynomials to fields.

Definition 2.23. Let F be a field, and let $f \in F[X]$. An element a is a **root**, or **zero** of f if $f(a) = 0$, where $f(a)$ means to substitute a for X in the expression for $f(X)$.

Theorem 2.24. A polynomial of degree n over a field has at most n zeros.

To establish an important property of the roots, we recall another familiar calculus concept, the derivative.

Definition 2.25. Let $f(X) = a_n X^n + a_{n-1} X^{n-1} + \cdots + a_1 X + a_0$ be an element of $F[X]$. The **derivative** of $f(X)$, denoted $f'(X)$, is the polynomial $n a_n X^{n-1} + (n-1) a_{n-1} X^{n-2} + \cdots + a_1$ in $F[X]$.

Theorem 2.26. A polynomial $f(X)$ over a field F has a repeated root if and only if $f(X)$ and $f'(X)$ have a common factor of degree ≥ 1 in $F[X]$.

Theorem 2.27. Let F be a field and let $h(X)$ be irreducible in $F[X]$ with root a. Then the extension field $F(a) = \{x + ya \mid x, y \in F\}$ is isomorphic to $F[X]/h(x)$ under the mapping $\psi : a \mapsto X$.

2.2.3 Cyclotomic Polynomials

The following definitions illustrate a specific type of polynomial that will be used in the AKS proof of correctness. For a more in-depth examination, see [6] or [9].

Definition 2.28. Let n be a positive integer. An **$\mathbf{n^{th}}$ root of unity** is a number z satisfying the equation $z^n = 1$. An n^{th} root of unity is **primitive** if it is not a k^{th} root of unity for some $k < n$, that is, $z^k \neq 1$ for all $k = 1, 2, \ldots, n-1$.

Definition 2.29. For any positive integer n, let $\omega_1, \omega_2, \ldots, \omega_{\phi(n)}$ denote the primitive nth roots of unity. The **$\mathbf{n^{th}}$ cyclotomic polynomial** in $\mathbb{C}[X]$ is the polynomial $\Phi_n(X) = (X - \omega_1)(X - \omega_2) \cdots (X - \omega_{\phi(n)})$.

Note that the roots of $\Phi_n(X)$ are precisely the $\phi(n)$ primitive n^{th} roots of unity, making it monic with degree $\phi(n)$.

Theorem 2.30. $\Phi_n(X)$ has integer coefficients and is a divisor of $X^n - 1$.

We also require the following result about cyclotomic polynomials over finite fields (see [9]).

Theorem 2.31. Let $\Phi_r(X)$ be the r^{th} cyclotomic polynomial over the finite field $\mathbb{F}_p$. Then $\Phi_r(X)$ divides $X^r - 1$ and factors into irreducible factors of degree $\text{ord}_r(p)$.

3 Algorithms in Number Theory

With the mathematical foundations in place, we turn to the computational side of things. Much of our further work in primality testing and its cryptographic applications requires making frequent computations of the greatest common divisor, multiplicative inverses, and powers of integers modulo n. We present the algorithms here that we will be using to compute these quantities. First, we give a definition of computational complexity.

Definition 3.1 (Big-O Notation). Let $f(n)$ and $g(n)$ be functions of n taking on positive values. We say f is Big-O of g and write

$$f(n) \in O(g(n))$$

if there are positive constants c and C such that

$$f(n) \leq cg(n) \qquad \text{for all} \quad n \geq C.$$

Big-O notation can give us an idea for how quickly the time and space required for a given algorithm grow with the input size n.

This paper deals only with algorithms that take integer inputs. Since we are running them on computers, we measure the input size as the number of bits needed to represent the integer in binary.

Notation

For an integer $n \geq 1$, let $||n|| = \lfloor \log_2(n) \rfloor + 1$ be the number of bits in the binary representation of n. Define $||0|| = 1$.

Since cryptographic applications require we deal with very large integers (hundreds of digits), we require algorithms that are at least polynomial time or better in the size of the input: $O(||n||^a)) = O((\log_2 n)^a)$ for some a. This means that the complexity we seek is a polynomial function of $\log_2 n$. In the following sections we will build up a bank of algorithms that will allow us to efficiently compute quantities on very large inputs.

We will also occasionally use the following rather technical notation.

Definition 3.2 (Soft-O Notation). Let $f(n)$ and $g(n)$ be functions of n taking on positive values. We say f is Soft-O of g and write

$$f(n) \in O^{\sim}(g(n))$$

if there exists k such that

$$f(n) \in O(g(n) \cdot \log^k g(n)).$$

This allows us to simplify Big-O notation by ignoring logarithmic factors, which do not contribute to the overall worst case run-time as much as super-logarithmic functions do. For example, the complexity

$$t(n) \in O((\log n)(\log\log n)(\log\log\log n))$$

can be simplified to $t(n) \in O^{\sim}(\log n)$.

To calculate the complexity of our algorithms, we use the following assumptions about basic math operations.

Cost of Basic Arithmetic

Let n and m be natural numbers. By [3] we can assume the following:

a) Adding or subtracting n and m takes $O(||n|| + ||m||) = O(\log n + \log m)$ bit operations.

b) Multiplying n and m takes $O(||n|| \cdot ||m||) = O(\log n \cdot \log m)$ operations.

c) Computing the quotient and remainder of n modulo m takes $O((||n|| - ||m|| + 1) \cdot ||m||) = O(\log n \cdot \log m)$ bit operations.[1]

Therefore addition and subtraction are linear in the binary length of the input numbers, and multiplication and division are no worse than quadratic.

3.1 GCD and Inverses

Given natural numbers a and b, what is their greatest common divisor? The following algorithm recursively returns the answer.

Algorithm 3.1 The Euclidean Algorithm

Input: Positive integers a and b, with $a \geq b$.

1: **if** $b = 0$ **then**
2: **return** a.
3: **else**
4: Set $a = b$, $b = a \mod b$, go back to line 1.
5: **end if**

Recall that Theorem 2.3 states that we can write $\gcd(a, b) = ax + by$ for some integers x and y. If $\gcd(a, b) = 1$, then a has a multiplicative inverse modulo b, namely x. If we keep track of these coefficients along the way, we can extend the Euclidean Algorithm to efficiently calculate inverses as well.

Time Complexity

By [2], with $a \geq b$ the Euclidean Algorithm takes no more than $2\log_2(b) + 2$ iterations to compute $\gcd(a, b)$[2].

Adding in the cost of each operation, by [3] the Euclidean Algorithm and its extended version are both $O(||a|| \cdot ||b||)$.

[1] The complexites in (b) and (c) assume the basic methods and can be improved to $O^{\sim}(B)$, where $B = \max\{||n||, ||m||\}$ using more advanced algorithms.

[2] In fact, it has been proven that the Euclidean Algorithm finishes in no more than $1.45\log_2(b) + 1.68$ iterations.

3.2 Raising Integers to Powers

Another common operation is raising an integer to a power, modulo another number. That is, computing $g^a \pmod{N}$.

Since the numbers are potentially very large, the naive approach of multiplying g by itself a times, then computing the remainder mod N is much too slow, and the numbers get too big. Instead, we use a more clever method: exponentiation by squaring.

We can compute the values g, g^2, g^4, g^8, ... by squaring and immediately reducing modulo N to keep the numbers small. We then look at the binary expansion of the exponent a to determine which powers of g we need to multiply together. For example, if $a = 27 = 16 + 8 + 2 + 1$, we multiply

$$g^{16} \cdot g^8 \cdot g^2 \cdot g \pmod{N}$$

to obtain the final result. The algorithm is as follows.

Algorithm 3.2 The Fast Powering Algorithm

Input: Positive integers g, a, and N.

1: Let $u = a$ and $s = g \mod N$.
2: Let $c = 1$.
3: **while** $u \geq 1$ **do**
4: **if** u is odd, **then** let $c = (c \cdot s) \mod N$.
5: Let $s = s \cdot s \mod N$.
6: Let $u = \lfloor u/2 \rfloor$.
7: **end while**
8: **return** c

Time Complexity

By [2], the Fast Powering Algorithm takes at most $2\log_2(a)$ multiplications modulo N to compute g^a. Therefore even for very large a, for example $a \approx 2^{1000}$, the computer only has to do around 2000 multiplications, an incredible savings over the brute force approach.

By [3], the algorithm has complexity $O(||a||\cdot||N||^2)$ using naive bit operations and $O^{\sim}(||a|| \cdot ||N||)$ using more advanced methods.

3.3 Testing For Perfect Powers

Consider the following definition:

Definition 3.3. An integer n is a **perfect power** if and only if there exist $a, b \in \mathbb{N}$ with $b > 1$ such that $n = a^b$.

Examples of perfect powers include $8 = 2^3$ and $15625 = 5^6$. At first glance, checking whether an integer n is a perfect power seems to involve a lot of trial and error exponentiation. However, we can do this in a remarkably efficient way by performing a binary search for the base that makes raising to the b power closest to n.

Algorithm 3.3 Perfect Power Test

Input: Integer $n \geq 2$.

Let $b = 2$.
while $2^b \leq n$ **do**
    Let $a = 1$, $c = n$.
    while $c - a \geq 2$ **do**
        Let $m = \lfloor (a + c)/2 \rfloor$.
        Let $p = \min\{m^b, n + 1\}$. (Truncated fast powering)[3]
        if $p = n$, **return** "Perfect Power" ($p = m^b$)
        if $p < n$ then let $a = m$, **else** let $c = m$.
    end while
    Let $b = b + 1$.
end while
return "Not a perfect power"

Time Complexity

By [3], the Perfect Power test is $O(||n||^4 \log ||n||)$ for naive bit operations and $O^{\sim}(||n||^3)$ for advanced methods.

[3] To compute m^b, we use Algorithm 3.2, stopping immediately after we obtain a result greater than $n + 1$.

4 Motivation

What is the point of being able to tell if a multi-hundred digit number is prime? Certainly, it can be an interesting problem in and of itself. However, it turns out such a tool is a necessity for supporting modern computer security systems that we rely on daily.

4.1 Public Key Cryptography

Perhaps the most common practical use for prime numbers today is in public key cryptography, which provides a framework for secure communication in the presence of an adversary.

In general, a public key cryptosystem is one in which a user holds two quantities: a public key used for encryption (known to everyone) and a private key used for decryption (known only to them). The public key enables other people to send encrypted information to the user, and the private key ensures that only the recipient will be able to successfully decrypt and read the original message.

Public key systems are based on mathematical problems that are hard to solve on the surface, but become much easier when another piece of information is known (i.e., the private key). It is important to note that by "hard", we mean "unable to be computed in any reasonable amount of time" (say, the age of the sun) by modern computers. We will illustrate this concept with one widely used system, RSA.

4.2 The RSA Cryptosystem

RSA is a public key cryptosystem that was named after and first published by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. RSA is used to protect a variety of communication done over the internet including through web browsers and email.

Consider two people, Bob and Alice, who want to communicate securely. The basic setup is as follows.

Bob: Choose two large primes, p and q. Keep these secret. Let $N = pq$, and choose encryption exponent e such that $\gcd(e, (p-1)(q-1)) = 1$. Publish $(N,\ e)$ as the public key.

Alice: Convert plaintext to integer m such that $1 \leq m < p, q$. Compute $c \equiv m^e \pmod{N}$ using Algorithm 3.2. Send ciphertext c to Bob.

Bob: Knowing p and q, solve the congruence $ed \equiv 1 \pmod{(p-1)(q-1)}$ for d using the Extended Euclidean Algorithm. Then compute $c^d \pmod{N} = m$ to convert back to plaintext.

We will first prove that Bob's decryption actually does yield the original message m.

Proof. We want to show $c^d = m^{ed} \equiv m \pmod{pq}$. Note that $\gcd(m, pq) = 1$ and $\phi(pq) = (p-1)(q-1)$. Since $ed \equiv 1 \pmod{(p-1)(q-1)}$, there is an integer k such that $ed = 1 + k(p-1)(q-1)$. We then have

$$\begin{aligned} m^{ed} &= m^{1+k(p-1)(q-1)} \\ &= m \cdot \left(m^{(p-1)(q-1)}\right)^k \\ &\equiv m \cdot 1^k \pmod{pq} && \text{by Euler's Theorem} \\ &= m. \end{aligned}$$

The message m is recovered. □

Why is this setup secure? Suppose an adversary had access to the communication channel over which this information was being broadcasted. They would then know the values N, e, and c, and face the challenge of solving the congruence $c \equiv m^e \pmod{N}$ for m. This task of taking e^{th} roots modulo N, also called the RSA problem, is conjectured to be computationally hard. That is, there is currently no algorithm to solve it in a reasonable amount of time given sufficiently large numbers.

Another way an adversary could gain access would be to factor N to find p and q. This would allow them to decrypt the message in exactly the same way Bob does. However, factoring is also believed to be a "hard" problem.

There are a myriad of other number theoretic intricacies that would potentially allow an adversary to gain more information that this paper will not cover. At the most basic level, assuming large enough primes are used, Alice and Bob can safely communicate without anyone else uncovering their messages because of the "one-way" nature of these mathematical problems.

As computing power has increased, so too has what is considered "large enough" for the key N to be unfactorable. For example, from RSA's conception in 1977 through the 1980's, 512 bits (or 155 decimal digits) was the recommended key size. By the 90's, numbers of this size had become too easy for computers to factor, so the recommendation increased to 1024 bits. Currently as of 2018, the recommended size is 2048 bits, or 617 decimal digits. Even these are likely to become breakable in the not-so-distant future. Not surprisingly, larger keys have the side effect of making ordinary decryption slower as well. Choice of key size is thus a trade-off between performance and security.

It is also worth mentioning that the security of RSA has not been proven theoretically; that is, we do not know for certain that no efficient algorithm can possibly exist to solve the RSA or factoring problems. It has not even been established that one must solve one of these problems to "crack" RSA, as there may be some clever alternate way of decoding messages that has yet to be discovered. The reason why we are confident in this system is simply

the fact that many people over a span of almost 50 years have tried to break it, and failed.

4.3 Prime Generation

In order for a system like RSA to run, one must first solve the *Prime Generation Problem* to generate the keys. An algorithm for this might look as follows:

1. Choose a random odd integer of the desired length.
2. Check if prime. If not, go back to step 1.

With this in mind, it is helpful to know how likely it is that a randomly chosen integer will be a prime. For this, we use the following famous theorem.

Theorem 4.1 (The Prime Number Theorem). Let N be an integer and let $\pi(N)$ equal the number of primes less than or equal to N. Then

$$\lim_{N\to\infty} \frac{\pi(N)}{N/\ln(N)} = 1.$$

Therefore, the probability that a randomly chosen integer N is prime is around $1/\ln(N)$.

If we wanted to make our RSA modulus N 2048 bits long, we would need to find two primes p and q of length approximately 1024 bits. We can use the prime number theorem to determine how many primes p are of this length, that is, satisfy $2^{1023} < p < 2^{1024}$.

$$\#\text{ of 1024 bit primes} = \pi(2^{1024}) - \pi(2^{1023}) \approx \frac{2^{1024}}{\ln 2^{1024}} - \frac{2^{1023}}{\ln 2^{1023}} \approx 2^{1013.53}$$

As can be seen, there are a lot to choose from.

5 Primality Testing

It may come as a surprise that it is no trivial task to determine whether or not a given number is prime. In this section, we examine why this is the case.

5.1 Brute Force

To begin to approach the idea of how we would create an efficient primality test, we start with the most basic, brute force approach. Given an integer input $n \geq 2$, we could determine its primality as follows:

Algorithm 5.1 Brute Force Primality Test

Input: Integer $n \geq 2$.

1: **for** $a = 2$ to $\lfloor \sqrt{n} \rfloor$:
2: **if** a divides n, **return** COMPOSITE.
3: **return** PRIME.

The brute force test is quite simple: run through every possible divisor of n. If one of them divides n, we know n is composite, otherwise n is prime. Note that we only need to check up to $\lfloor \sqrt{n} \rfloor$ since we will start getting repeats if we go higher.

We could improve this test by only checking the *primes* less than $\lfloor \sqrt{n} \rfloor$, rather than every integer. However, this would require we carry around a pre-made list, which is not feasible for extremely large values.

Why is this test inefficient? We see that in the worst case (when n is in fact a prime), this test takes approximately $\sqrt{n}$ computations. Doesn't that make it $O(\sqrt{n})$?

Recall that the number of bits needed to represent n is approximately $\log_2 n$. Seen in this light, a string of $\log_2 n$ 0's and 1's has caused the computer to do $\sqrt{n}$ units of work. Rewriting the complexity in terms of the number of bits $||n||$, we see that this test is actually $O(2^{||n||/2})$, which grows exponentially. In order to test a 2048 bit number this would require approximately 2^{1024} computations, an astronomical amount of work.

5.2 Probabilistic Tests

Brute force trial division was an example of a **deterministic** test. In this section we will see what it means to be a **probabilistic** test. Probabilistic tests, in addition to the input n, take another random integer a with $1 \leq a \leq n-1$ and return one of two outcomes:

- Composite
- Inconclusive.

The integer a is called a **witness** to the compositeness of n. The idea behind a probabilistic test is to test the number n with several values of a. If any of the tests come back composite, we can stop and be sure that n is not prime. If however, they all come back inconclusive, we can be more and more convinced that n is in fact a prime.

A good probabilistic test will have a high likelihood (ideally 50% or better) of returning composite if n is composite. One can see that after 50 'inconclusive' results, the probability[1] that n is actually composite is less than $(1/2)^{50} = 2^{-50}$. For most practical purposes, this gives us a very good chance of having a prime.

[1]This is an over-simplification, the actual probability can be calculated using conditional probabilities and Bayes' formula based on the distribution of the set of primes.

5.2.1 Fermat Test

We now turn to the question of how to construct such a test. It turns out that we already have a powerful tool to do so. Recall that Fermat's Little Theorem states that if an integer n is prime, then $a^n \equiv a \pmod{n}$ for all a.

Since Fermat's Little Theorem only works in one direction, the fact that $a^n \equiv a \pmod{n}$ for some a alone does not allow us to conclude that n is prime. For example, $2^{341} \equiv 2 \pmod{341}$ but $341 = 11 \cdot 31$. We can however take a probabilistic approach and use the Fast Powering Algorithm (Algorithm 3.2) to compute $a^n \pmod{n}$ for several random a's. If we get that $a^n \not\equiv a \pmod{n}$ for some a, we can conclude that n is composite. Otherwise, we become more and more sure that n is prime, stopping after some threshold k. The test looks as follows.

Algorithm 5.2 Fermat Test

Input: Integer $n \geq 2$.

1: **for** $i = 1$ to k:
2: Choose random $a \in [2, n-1]$.
3: **if** $a^n \not\equiv a \pmod{n}$, **return** COMPOSITE.
4: **return** PROBABLY PRIME.

Using Algorithm 3.2, each computation of a^n is $O^{\sim}(||n||^2)$, making the overall complexity of the Fermat Test $O^{\sim}(k \cdot ||n||^2)$, where k is the number of times we run the test.

The Fermat Test is quite fast and easy to implement. The test does however have a serious flaw: there exists a certain type of composite number that completely fools it. Consider the composite number 561, which factors as $3 \cdot 11 \cdot 17$. It can be shown that

$$a^{561} \equiv a \pmod{561} \quad \text{for all } a.$$

Though fairly sparse, it turns out that there are an infinite number of integers with this property, which are called Carmichael numbers.

If one were to run the Fermat Test on a Carmichael number c, the congruence $a^c \equiv a \pmod{c}$ would always be satisfied, causing the test to wrongly proclaim c as prime. We therefore need something a little stronger for our probabilistic primality test.

5.2.2 Miller-Rabin Test

The Miller-Rabin Test improves on the weaknesses of the Fermat Test. Based on theory formulated by Russian mathematician M. M. Artjuhov in 1966, Gary Miller created a deterministic, polynomial time primality test that relied on the unproven Extended Riemann Hypothesis in 1975. The test was later modified by Michael Rabin to yield the probabilistic, but unconditional Miller-Rabin test that we know today.

Their test is based on the following result:

Proposition 5.1. Let p be an odd prime and write

$$p - 1 = 2^k q \quad \text{with } q \text{ odd.}$$

Let a be any number not divisible by p. Then one of the following conditions is true:

i) $a^q \equiv 1 \pmod{p}$,

ii) One of $a^q,\ a^{2q},\ a^{4q}, \ldots,\ a^{2^{k-1}q}$ is congruent to $-1 \pmod{p}$.

Proof. By Fermat's Little Theorem, $a^{p-1} \equiv 1 \pmod{p}$. This tells us that looking at the list of numbers

$$a^q,\ a^{2q},\ a^{4q}, \ldots,\ a^{2^{k-1}q}, a^{2^k q},$$

the last number in the list (which is equal to a^{p-1}) is congruent to 1 modulo p. Since each number in the list is a square of the previous number, one of the following must occur:

i) The first number in the list is congruent to 1 modulo p.

ii) Some number in the list is not congruent to 1 modulo p, but becomes congruent to 1 when squared. The only number satisfying both

$$b \not\equiv 1 \pmod{p} \quad \text{and} \quad b^2 \equiv 1 \pmod{p}$$

is -1, so one of the numbers in the list is congruent to $-1 \pmod{p}$.

This completes the proof. □

This can be turned into a primality testing algorithm as follows.

Algorithm 5.3 Miller-Rabin Test

Input: Integer $n \geq 2$.

1: Choose random potential witness $a \in [2, n-2]$.
2: **if** n is even or $1 < \gcd(a, n) < n$, **return** COMPOSITE.
3: Set $a = a^q \pmod{n}$.
4: **if** $a \equiv 1 \pmod{n}$, **return** INCONCLUSIVE.
5: **for** $i = 0$ to $k - 1$:
6: **if** $a \equiv -1 \pmod{n}$, **return** INCONCLUSIVE.
7: Set $a = a^2 \pmod{n}$.
8: **return** COMPOSITE.

The correctness of the algorithm can be seen from the contrapositive of Proposition 5.1. Let n be an odd number and write $n - 1 = 2^k q$ with q odd. Let $a \in [2, n-2]$ with $\gcd(a, n) = 1$. If both

i) $a^q \not\equiv 1 \pmod{n}$, and

ii) $a^{2^i q} \not\equiv -1 \pmod{n}$ for all $i = 0, 1, 2, \ldots, k-1$,

then n is composite.

Time Complexity

As before, we use Algorithm 3.2 to compute a^q. By [3], one iteration of the Miller-Rabin test takes $O^{\sim}(||n||^2)$ time. Therefore if we were to test k different witnesses a, the time complexity is $O^{\sim}(k \cdot ||n||^2)$.

It can be shown that for an odd composite number n, at least 75% of the numbers a between 1 and $n-1$ are Miller-Rabin witnesses for n and will cause the algorithm to return COMPOSITE. Importantly, there is also no analogue to the Carmichael numbers of the Fermat Test. This means that after 50 'INCONCLUSIVE' runs of Miller-Rabin, the probability that n is actually composite is only about $(25\%)^{50} \approx 10^{-31}$.

Remark. The odds of getting a false prime after 50 rounds of Miller-Rabin ($\sim 10^{-31}$) are less than one millionth the odds of a cosmic ray flipping a bit to produce the wrong result in a deterministic primality test ($\sim 10^{-24}$). Therefore, for all intents and purposes there is no reason to use a slower deterministic test when generating primes to use in cryptosystems. The algorithms may be deterministic, but the computers they run on are not!

6 The AKS Primality Test

The primality testing algorithm presented by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena in their 2002 paper *PRIMES is in P* was the first to have all four characteristics of general, deterministic, unconditional, and polynomial time. Subsequently, improvements were made to the runtime and the paper was re-published in 2004. This chapter will outline the general concepts, walk through the updated algorithm step by step, and prove its correctness. Drawing on the methods in Chapter 3, we will also demonstrate the time complexity of the algorithm, and culminate with a discussion of its theoretical and practical significance.

6.1 The Idea

The AKS test is based on a generalization of Fermat's Little Theorem to polynomials.

Lemma 6.1. Let $n \geq 2$ be an integer, and $a < n$ be an integer with $\gcd(a, n) = 1$. Then n is prime if and only if

$$(X + a)^n \equiv X^n + a \pmod{n}.$$

Note that a and n are real numbers in this expression but the letter X is not. Rather, it is a formal symbol that is part of a polynomial as described in Chapter 2. Equivalence is defined as having the same coefficients in the polynomial ring $\mathbb{Z}_n[X]$.

Proof. From the Binomial Theorem, we have

$$(X+a)^n = \sum_{i=0}^{n} \binom{n}{i} a^{n-i} X^i \tag{6.1}$$

It follows that the coefficient of X^i in $(X+a)^n$ is $\binom{n}{i}a^{n-i}$.

Suppose n is prime. Then for $0 < i < n$, $\binom{n}{i} = \frac{n!}{i!(n-i)!} \equiv 0 \pmod{n}$, since n divides the numerator but does not divide the denominator (i and $n-i$ are both less than n). For $i = 0$, we get the term a^n since $X^0 = 1$ and $\binom{n}{0} = 1$. For $i = n$, we get the term X^n since $a^{n-n} = 1$ and $\binom{n}{n} = 1$. It follows that $(X+a)^n \equiv a^n + 0 + 0 + \cdots + 0 + X^n \pmod{n}$. By Fermat's Little Theorem, $a^n \equiv a \pmod{n}$ so we are left with $(X+a)^n \equiv X^n + a \pmod{n}$, as desired.

Suppose n is composite. Consider a prime q that is a factor of n and let k be the power of q in the prime factorization of n. Note that $1 < q < n$, and q^k divides n but q^{k+1} does not.

Consider the coefficient of X^q in equation (6.1):

$$\binom{n}{q} \cdot a^{n-q} = \frac{n!}{q!(n-q)!} \cdot a^{n-q} = \frac{n(n-1)\cdots(n-q+1)}{q!} \cdot a^{n-q}$$

Looking at the right-hand side of this equation, note that q^k divides n, but all other factors in the numerator are relatively prime to q. It follows that the numerator is divisible by q^k but not q^{k+1}. The factor of q in the denominator cancels with one of the q's in the numerator, leaving the entire fraction not divisible by q^k.

Since $\gcd(a, n) = 1$, we have q^k and a^{n-q} relatively prime. Since q^k does not divide either term in the coefficient $\binom{n}{q}a^{n-q}$, n certainly cannot either since q^k divides n. Hence, the coefficient of X^q is nonzero (mod n). Recall that $1 < q < n$, therefore $(X+a)^n \not\equiv X^n + a \pmod{n}$. □

We will illustrate this result with a few examples.

Example. Test whether 7 is prime.

Choose $a = 1$ for convenience.
Evaluate $(x+1)^7 \pmod 7$.

$$\begin{aligned}(x+1)^7 &= x^7 + 7x^6 + 21x^5 + 35x^4 + 35x^3 + 21x^2 + 7x + 1 \\ &\equiv x^7 + 1 \pmod 7\end{aligned}$$

The congruence in Lemma 6.1 is satisfied, therefore 7 is prime.

Example. Test whether 6 is prime.

Evaluate $(x+1)^6 \pmod 6$.

$$\begin{aligned}(x+1)^6 &= x^6 + 6x^5 + 15x^4 + 20x^3 + 15x^2 + 6x + 1 \\ &\equiv x^6 + 3x^4 + 2x^3 + 3x^2 + 1 \pmod 6 \\ &\not\equiv x^6 + 1\end{aligned}$$

The congruence in Lemma 6.1 is not satisfied, therefore 6 is composite. Observe that the nonzero coefficients correspond to precisely the powers of x that share prime factors with 6.

Note that Lemma 6.1 provides a test for primality on its own. However, computing the coefficients on $(X+a)^n$ is exponential in the size of n, making the test extremely slow for larger inputs. In fact, brute force trial division is faster.

The main breakthrough of the AKS test comes in evaluating the polynomials on both sides of the equation in Lemma 6.1 modulo another polynomial, $X^r - 1$, and checking several different a's. That is, testing the congruence

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

This lowers the degree of the polynomials and reduces the number of coefficients needing to be evaluated, thus improving the efficiency.

We will show in the correctness proof that the appropriate r is bounded above and only requires us to check a's up to a certain point to decide about the primality of n, reducing the complexity of the algorithm into polynomial time.

6.2 The Algorithm

Notation

We use log to denote the base 2 logarithm.

We use $\phi(n)$ to denote Euler's Phi Function as defined previously.

In the correctness proof, we use $\mathbb{F}_p$ to refer to the finite field $\mathbb{Z}/p\mathbb{Z}$.

Algorithm 6.1 The AKS Primality Test

Input: Integer $n \geq 2$.

1: **if** $n = a^b$ for $a \in \mathbb{N}$ and $b > 1$, **return** COMPOSITE.
2: Find the smallest r such that $\text{ord}_r(n) \not\leq \log^2 n$.
3: **if** $1 < \gcd(a, n) < n$ for some $a \leq r$, **return** COMPOSITE.
4: **if** $n \leq r$, **return** PRIME.
5: **for** $a = 1$ to $\lfloor \sqrt{\phi(r)} \log n \rfloor$:
6: **if** $(X + a)^n \not\equiv X^n + a \pmod{X^r - 1,\ n}$, **return** COMPOSITE.
7: **return** PRIME.

Line 1 checks if n is a power of an integer where the exponent is greater than 1. We can use the Perfect Power Test (Algorithm 3.3) to verify this.

Line 2 looks for the smallest value of r such that there is no k satisfying the equation $n^k \equiv 1 \pmod{r}$ for some $1 \leq k \leq \log^2 n$. We do this by trying successive values of r starting at $r = 2$ and testing if $n^k \not\equiv 1 \pmod{r}$ for every $1 \leq k \leq \log^2 n$. If none of these are congruent to 1, we have found our r, otherwise we increment r by 1 and try again.

Note that there are two options here, either $\text{ord}_r(n) > \log^2 n$ or $\gcd(r, n) > 1$. The first case means that the smallest k satisfying $n^k \equiv 1 \pmod{r}$ is greater than $\log^2 n$. The second case means $n^k \equiv 1 \pmod{r}$ does not have a solution for any value of k.

Line 3 checks if n shares a common nontrivial factor with any number $a \leq r$. This step is necessary because we don't yet know if $\text{ord}_r(n)$ is defined. We compute the GCD using the Euclidean Algorithm (Algorithm 3.1).

Line 4 checks whether or not we have looked at the GCD of all numbers less than n in the previous step. If we have, we will have found a nontrivial factor if there was one in line 3 and returned COMPOSITE there.

Lines 5 and 6 verify a series of congruences for successive values of a. It is clear that these congruences are satisfied mod n if and only if n is prime from Lemma 6.1. Our subsequent analysis will focus on proving that they are satisfied mod $X^r + 1$ as well, and that it suffices to check values of a up to $\lfloor \sqrt{\phi(r)} \log n \rfloor$ to determine the primality of n.

Line 7 returns PRIME if the algorithm still has not decided.

6.3 Proof of Correctness

The correctness of the algorithm depends on the following theorem.

Theorem 6.2 (Main Theorem). Given an input number $n > 2$, the algorithm returns PRIME if and only if n is prime.

We will establish this result through a series of propositions. Proposition 6.3 gives the '$\leftarrow$' direction of the proof. We split the other direction into two cases. The case of returning PRIME in line 4 is covered by Proposition 6.4. The rest of the section is dedicated to proving the second case, returning PRIME in line 7 (Proposition 6.5).

Proposition 6.3. If n is prime, the algorithm returns PRIME.

Proof. Suppose n is prime. Clearly the statements in lines 1 and 3 can never be true and thus will never return COMPOSITE. By Lemma 6.1, the `for` loop in lines 5-6 will never return COMPOSITE either, since $(X + a)^n \equiv X^n + a \pmod{n}$ for all a. Therefore the algorithm will return PRIME in either line 4 or line 7. □

We will now prove the converse, that is, if the algorithm returns PRIME then n is prime. There are two ways the algorithm can return PRIME: either in line 4 or in line 7. We will consider the line 4 case first.

Proposition 6.4. If the algorithm returns PRIME in line 4, then n is prime.

Proof. Suppose the algorithm returns PRIME in line 4. Then $n \leq r$, so we have looked at $\gcd(a, n)$ for all $1 \leq a \leq n$ in line 3. Thus if n had been composite we would have found a nontrivial factor in this step. Therefore n must be prime. □

For the rest of the section we will assume that the algorithm returns PRIME in line 7.

Proposition 6.5. If the algorithm returns PRIME in line 7, then n is prime.

We have some more work to do before we are able to prove Proposition 6.5. First, we will establish a bound on the number r found in line 3. We will use the following result:

Lemma 6.6. Let $\mathrm{LCM}(m)$ denote the least common multiple of the first m numbers. For $m \geq 7$:

$$\mathrm{LCM}(m) \ \geq 2^m \ .$$

For a proof, see [10].

Lemma 6.7. For all $n \geq 2$, there exists an $r \leq \max\left\{3, \ \left\lceil \log^5 n \right\rceil\right\}$ such that either $\mathrm{ord}_r(n) > \log^2 n$ or $\gcd(r, n) > 1$, in which case $\mathrm{ord}_r(n)$ is undefined.

Proof. Suppose $n = 2$. Then $r = 3$ satisfies $3 \leq \max\left\{3,\ \left\lceil \log^5(2) \right\rceil\right\}$. Furthermore, $2^2 \equiv 1 \pmod 3$ giving $\mathrm{ord}_3(2) = 2$, hence $\mathrm{ord}_3(2) > \log^2(2) = 1^2 = 1$.

With the trivial case established, assume $n > 2$. Let $R = \{r_1, r_2, \ldots, r_t\}$ be the set of numbers such that either $\mathrm{ord}_{r_i}(n) \leq \log^2 n$ or r_i divides n. Then each of these numbers must divide the product

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

Either r_i divides n or $n^k \equiv 1 \pmod{r_i}$ where $k = \mathrm{ord}_{r_i}(n) \leq \log^2 n$, implying that r_i divides $(n^k - 1)$ in the $i = k$ term.

Furthermore, observe that

$$\begin{aligned}
n \cdot \prod_{i=1}^{\lfloor \log^2 n \rfloor} \left(n^i - 1\right) &= n \cdot (n-1) \cdot (n^2 - 1) \cdots (n^{\lfloor \log^2 n \rfloor} - 1) \\
&< n \cdot n \cdot n^2 \cdot n^3 \cdots n^{\lfloor \log^2 n \rfloor} \\
&< n^{\lfloor \log^2 n \rfloor} \cdot n^{\lfloor \log^2 n \rfloor} \cdots n^{\lfloor \log^2 n \rfloor} \qquad (\lfloor \log_2 n \rfloor \text{ times}) \\
&\leq n^{\log^2 n \ \cdot \ \log^2 n} \\
&= n^{\log^4 n} \\
&= \left(2^{\log n}\right)^{\log^4 n} \\
&= 2^{\log^5 n},
\end{aligned}$$

giving the product Π a strict upper bound of $2^{\log^5 n}$. Since $\lceil \log^5 n \rceil > 10$ for $n > 2$, Lemma 6.6 applies, giving us the inequality

$$\mathrm{LCM}\left(\lceil \log^5 n \rceil\right) \ \geq \ 2^{\lceil \log^5 n \rceil} > \Pi.$$

Since $\mathrm{LCM}\big(\lceil \log^5 n \rceil\big)$ is strictly greater than the product Π, there is some number $s \leq \lceil \log^5 n \rceil$ that divides $\mathrm{LCM}\big(\lceil \log^5 n \rceil\big)$ but does not divide Π[1]. Since

[1]This is because if all s for which $1 \leq s \leq \lceil \log^5 n \rceil$ divided Π, we would have $\Pi \geq \mathrm{LCM}\left(\lceil \log^5 n \rceil\right)$ which is not the case.

every element of R divides Π, it follows that $s \notin R$.

If $\gcd(s, n) = 1$ then $\text{ord}_s(n)$ is defined, and since $s \notin R$, $\text{ord}_s(n) > \log^2 n$ and the first case is satisfied.

If $\gcd(s, n) > 1$, then the second case is satisfied. □

With this bound on r in mind, we proceed with the proof. We will also use the following lemma.

Lemma 6.8. If $\text{ord}_r(n) > 1$, then there exists a prime divisor p of n such that $\text{ord}_r(p) > 1$.

Proof. Suppose $\text{ord}_r(n) > 1$. Let $n = \prod_{i=1}^{t} p_i^{e_i}$ be the prime factorization of n.

Suppose $\text{ord}_r(p_i) = 1$ for all $1 \leq i \leq t$. Then $p_i \equiv 1 \pmod{r}$ for all i, so $n = \prod_{i=1}^{t} p_i^{e_i} \equiv \prod_{i=1}^{t} (1)^{e_i} \equiv 1 \pmod{r}$ and hence $\text{ord}_r(n) = 1$, a contradiction. Therefore there is a prime divisor p_i of n such that $\text{ord}_r(p_i) > 1$ for some $1 \leq i \leq t$. □

In the case that $n = 2$, the algorithm will return PRIME in line 4[2]. Suppose $n > 2$. Since we have gotten all the way to line 7 in the algorithm, there are a number of assertions we can make about n and r.

For one, we know that $n > r$ and $\gcd(n, r) = 1$ since the algorithm did not return in lines 3 or 4.

It follows that $\text{ord}_r(n)$ is defined and greater than $\log^2 n$ by the way r was chosen in line 2. Since $\log^2 n > 1$ when $n > 2$, by Lemma 6.8 there exists a prime divisor p of n such that $\text{ord}_r(p) > 1$.

Furthermore, from getting to line 7 we can conclude that $p > r$ since if $p \leq r$, either $p < n$ and the algorithm would have returned COMPOSITE in line 3 or $p = n$ and the algorithm would have returned PRIME in line 4.

It follows that $p, n \not\equiv 0 \pmod{r}$ and hence $p, n \in \mathbb{Z}_r^*$. The numbers p and r will be fixed from this point forward. Our goal for the rest of the section is to

[2]The algorithm will choose $r = 2$ since $\log^2(2) = 1$ and $2^k \not\equiv 1 \pmod{2}$ for every $1 \leq k \leq 1$. Then $n \leq r$, causing line 4 to return PRIME.

show that $p = n$, in order to prove that n is prime.

Given what we know thus far, we will now analyze the polynomial congruences. Let $\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor$. Note that the `for` loop in lines 5 and 6 checks ℓ equations. Since the algorithm does not return COMPOSITE in this step, we have

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

for all a, $0 \leq a \leq \ell$. Note that we have included $a = 0$ since the equation is trivially satisfied in this case.

Since n divides $((X + a)^n - (X^n + a))$, we have that p also divides $((X + a)^n - (X^n + a))$ since p divides n. It follows that

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

for $0 \leq a \leq \ell$.

We use the following language to describe this property.

Definition 6.9. Given a polynomial $f(X)$ and number $m \in \mathbb{N}$, we say that m is **introspective** for $f(X)$ if

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

where r and p are fixed numbers as before.

Under this definition, n and p are both introspective for $X + a$ when $0 \leq a \leq \ell$. We make the following claim.

Claim: $\frac{n}{p}$ is introspective for $X + a$ when $0 \leq a \leq \ell$.

Proof. Note that $\mathbb{Z}_p[X]$ is a ring of characteristic p. By Theorem 2.14, we have

$$(f + g)^p \equiv f^p + g^p \pmod{p}. \tag{6.2}$$

In order to show $\frac{n}{p}$ is introspective for $X + a$, we must show that

$$(X + a)^{n/p} - (X^{n/p} + a) \equiv 0 \pmod{X^r - 1, p}.$$

Dividing the left-hand-side by $X^r - 1$ using polynomial long division, we obtain

$$(X + a)^{n/p} - (X^{n/p} + a) = (X^r - 1) \cdot Q(X) + R(X),$$

with polynomial quotient $Q(X)$ and remainder $R(X)$, which we will call Q and R to simplify notation. We wish to show that $R \equiv 0 \pmod{X^r - 1}$ in order to prove that the left-hand-side is divisible by $X^r - 1$.

Raising both sides of the equation to the p power and simplifying using equation 6.2, we then have

$$\begin{aligned}
\left[(X + a)^{n/p} - (X^{n/p} + a)\right]^p &\equiv [(X^r - 1) \cdot Q + R]^p \pmod{p} \\
(X + a)^n - (X^{n/p} + a)^p &\equiv (X^r - 1)^p \cdot Q^p + R^p \pmod{p} \\
(X + a)^n - (X^n + a) &\equiv (X^r - 1)^p \cdot Q^p + R^p \pmod{p}
\end{aligned}$$

Since $(X + a)^n \equiv X^n + a \pmod{X^r - 1, p}$ for all $0 \leq a \leq \ell$, we have $(X^r - 1)^p Q^p + R^p \equiv 0 \pmod{X^r - 1, p}$. Since $X^r - 1$ clearly divides $(X^r - 1)^p$, it must also divide R^p.

Furthermore, $X^r - 1$ can be broken into irreducible factors

$$X^r - 1 = (X - a_1)(X - a_2) \cdots (X - a_r).$$

Since the derivative $(X^r - 1)' = rX^{r-1}$ does not share any factors of degree ≥ 1 with $X^r - 1$, each a_i is distinct (Theorem 2.26). It follows that for each a_i, $X - a_i$ divides $X^r - 1$ and hence also divides R^p. Since $X - a_i$ is irreducible, it follows that $X - a_i$ divides R for every $1 \leq i \leq r$.

Therefore $X^r - 1$ divides R and hence $R \equiv 0 \pmod{X^r - 1}$. We then have $(X + a)^{n/p} - (X^{n/p} + a) \equiv 0 \pmod{X^r - 1, p}$ for all $0 \leq a \leq \ell$, showing $\frac{n}{p}$ is introspective for $X + a$ when $0 \leq a \leq \ell$. □

We will now prove some properties of introspective numbers and polynomials in order to define two groups upon which the rest of the proof will be based. We begin with the fact that introspective numbers are closed under multiplication.

Lemma 6.10. Let m and m' be introspective numbers for polynomial $f(X)$. Then $m \cdot m'$ is also introspective for $f(X)$.

Proof. Since m is introspective for $f(X)$, we have

$$[f(X)]^{m \cdot m'} \equiv [f(X^m)]^{m'} \pmod{X^r - 1, p}.$$

Since m' is also introspective for $f(X)$, we can replace X by X^m in the introspection identity to obtain

$$\begin{aligned} [f(X^m)]^{m'} &\equiv f(X^{m \cdot m'}) \pmod{X^{m \cdot r} - 1, p} \\ &\equiv f(X^{m \cdot m'}) \pmod{X^r - 1, p} \text{ (since } X^r - 1 \text{ divides } X^{m \cdot r} - 1). \end{aligned}$$

Putting these equations together gives us

$$[f(X)]^{m \cdot m'} \equiv f(X^{m \cdot m'}) \pmod{X^r - 1, p},$$

as desired. □

The set of polynomials for which a number m is introspective is also closed under multiplication.

Lemma 6.11. If $m \in \mathbb{N}$ is introspective for polynomials $f(X)$ and $g(X)$, then m is also introspective for $f(X) \cdot g(X)$.

Proof. Let $h(X) = f(X) \cdot g(X)$. We then have

$$\begin{aligned}
[h(X)]^m &= [f(X) \cdot g(X)]^m \\
&= [f(X)]^m \cdot [g(X)]^m \\
&\equiv f(X^m) \cdot g(X^m) \pmod{X^r - 1, p} \\
&\equiv h(X^m) \pmod{X^r - 1, p}.
\end{aligned}$$

Therefore m is introspective for $f(X) \cdot g(X)$. □

Since $\frac{n}{p}$ and p are both introspective for $(X + a)$ when $0 \leq a \leq \ell$, it follows by Lemmas 6.10 and 6.11 that every number in the set $I = \{(\frac{n}{p})^i \cdot p^j \mid i, j \geq 0\}$ is introspective for every polynomial in the set $P = \{\prod_{a=0}^{\ell}(X + a)^{e_a} \mid e_a \geq 0\}$. We are now ready to define the two groups.

For the first group, let I_r be the set of all residues of numbers in I modulo r. That is,

$$I_r = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \pmod{r} \;\middle|\; i, j \geq 0 \right\}.$$

Observe that I_r is generated by n and p modulo r, and since $\gcd(n, r) = \gcd(p, r) = 1$, I_r is a subgroup of $\mathbb{U}_r$. Let $|I_r| = t$ be the order of this group.

Recall that $\mathrm{ord}_r(n) > \log^2 n$, therefore $t > \log^2 n$. To illustrate this, pick for instance the elements

$$\begin{aligned}
\left(\frac{n}{p}\right)^0 \cdot p^0 &= 1 \\
\left(\frac{n}{p}\right)^1 \cdot p^1 &= n \\
\left(\frac{n}{p}\right)^2 \cdot p^2 &= n^2 \\
&\vdots \\
\left(\frac{n}{p}\right)^{\lfloor \log^2 n \rfloor} \cdot p^{\lfloor \log^2 n \rfloor} &= n^{\lfloor \log^2 n \rfloor}
\end{aligned}$$

and note that they are all distinct modulo r.

For the second group, we will require a bit of algebraic machinery. Let $\Phi_r(X)$ be the r^{th} cyclotomic polynomial over the finite field $\mathbb{F}_p$. Note that $\Phi_r(X)$ divides $X^r - 1$.

By [9], $\Phi_r(X)$ factors into irreducible factors of degree $\text{ord}_r(p)$. Let $h(X)$ be one of these irreducible factors. Recall that $\text{ord}_r(p) > 1$, therefore the degree of $h(X)$ is greater than one.

Let $F = \mathbb{F}_p[X]/h(X)$ be the field of polynomials with coefficients in $\mathbb{F}_p$ modulo $h(x)$.

We define the second group G as the set of all residues of polynomials in P modulo $h(X)$ and p. That is,

$$G = \left\{ \prod_{a=0}^{\ell} (X+a)^{e_a} \pmod{h(X), p} \;\middle|\; e_a \geq 0 \right\}.$$

Observe that G is generated by the elements $X,\ X+1,\ X+2, \ldots,\ X+\ell$ and is a subgroup of the multiplicative group of F (G is clearly closed under multiplication and contains 1).

We will now show that the size of G can be bounded. We will make use of the following formula for the sum of binomial coefficients.

Lemma 6.12. Let $m \in \mathbb{N}$. The following is true for all integers $k \geq 0$.

$$\sum_{i=0}^{k} \binom{m+i}{m} = \binom{m+k+1}{m+1}.$$

Proof. We have $\binom{m+0}{m} = 1 = \binom{m+0+1}{m+1}$, so the formula is true for $k = 0$. Suppose the formula is true for some k. Recall the following recursive formula for binomial coefficients, derived from Pascal's Triangle:

$$\binom{m}{k} = \binom{m-1}{k} + \binom{m-1}{k-1}. \tag{6.3}$$

Then

$$\begin{aligned}\sum_{i=0}^{k+1}\binom{m+i}{m} &= \sum_{i=0}^{k}\binom{m+i}{m} + \binom{m+k+1}{m} \\ &= \binom{m+k+1}{m+1} + \binom{m+k+1}{m} && \text{by inductive hypothesis} \\ &= \binom{m+(k+1)+1}{m+1}, && \text{by (6.3)}\end{aligned}$$

showing the formula to be true for $k+1$ as well. By induction, the result holds for all $k \geq 0$. □

The following lemma provides a lower bound on the size of G.

Lemma 6.13. $|G| \geq \binom{t+\ell}{t-1}$.

Proof. Since $h(X)$ is an irreducible factor of the cyclotomic polynomial $\Phi_r(X)$, its roots are primitive r^{th} roots of unity (see Definition 2.29). Suppose ω is one such root. Then ω is a primitive root in some extension field $\mathbb{F}_p(\omega)$, which is isomorphic to $\mathbb{F}_p[X]/h(X) = F$ by Theorem 2.27. The standard isomorphism maps ω to X, so X is a primitive r^{th} root of unity in F.

We will first show that any two distinct polynomials of degree less than t in the set $P = \{\prod_{a=0}^{\ell}(X+a)^{e_a} \mid e_a \geq 0\}$ map to different elements in G.

Let $f(X)$ and $g(X)$ be two distinct polynomials of degree less than t in P. Suppose toward a contradiction that $f(X) = g(X)$ in the field F (recall that G is a subgroup of F). This is equivalent to saying $f(X) \equiv g(X) \pmod{h(X), p}$.

Let $m \in I$. We then have $[f(X)]^m = [g(X)]^m$ in F. Since m is introspective for both f and g, we have

$$\begin{aligned} f(X^m) &\equiv g(X^m) \pmod{X^r - 1, p} \\ f(X^m) &\equiv g(X^m) \pmod{h(X), p} \text{ since } h(X) \text{ divides } X^r - 1, \end{aligned}$$

implying that $f(X^m) = g(X^m)$ in F. It follows that X^m is a root of the polynomial $Q(Y) = f(Y) - g(Y)$ for every $m \in I_r$ (recall that I_r is the set of

residues in I modulo r).

Since $\gcd(m, r) = 1$ (I_r is a subgroup of $\mathbb{U}_r$) and X is a primitive r^{th} root of unity in F, the order of X^m is $r/\gcd(m, r) = r$ showing that X^m is also a primitive r^{th} root of unity in F.

It follows that each X^m is distinct and since there are $|I_r| = t$ choices of m, there are t distinct roots of $Q(Y)$ in F. However, the degree of $Q(Y)$ is less than t since f and g were chosen to have degree less than t.

Since the number of roots must be less than or equal to the degree, this is a contradiction and therefore $f(X) \neq g(X)$ in F. Hence, any two distinct polynomials of degree less than t in the set P map to different elements in G.

Since $\text{ord}_r(n) > \log^2 n$, it follows that $r > \log^2 n$. We then have

$$\ell = \lfloor \sqrt{\phi(r)} \log n \rfloor < \sqrt{r} \log n < r < p,$$

hence $i \neq j$ in $\mathbb{F}_p$ for $1 \leq i \neq j \leq \ell$. Therefore the elements $X,\ X+1,\ X+2, \ldots,\ X+\ell$ are all distinct in F.

Additionally, since the degree of h is greater than one, $X + a \neq 0$ in F for every a, $0 \leq a \leq \ell$. Therefore there exist at least $\ell + 1$ distinct polynomials of degree one in G: namely $X,\ X+1,\ X+2, \ldots,\ X+\ell$.

We seek to find how many distinct polynomials of degree less than t are contained in G. This is equal to the sum

$$\sum_{k=0}^{t-1} (\#\text{ of distinct polynomials of degree } k \text{ in } G)\,.$$

In order to find the number of distinct polynomials of degree k, we look at the number of ways we can multiply k of the above factors of degree one (the generators of G). This can be visualized in the following way,

$$\underbrace{\_\_\ \$\ \ \$\ \_\_\ \_\_\ \_\_\ \$\ \_\_\ \cdots\ \_\_\ \_\_\ \_\_\ \$\ \_\_\ \_\_\ \$\ \_\_}_{\ell \text{ \$ signs inserted between } k \text{ spaces}}$$

where X terms go in the space(s) before the first \$ sign, $X+1$ terms go in the space(s) between the first and second \$ signs, and so on, up to $X+\ell$ terms going in the space(s) after the ℓ^{th} \$ sign. In the above example, there would be one X term, no $X+1$ terms, three $X+2$ terms, two $X+\ell-1$ terms, and one $X+\ell$ term (with other terms in the middle).

All together there are $k+\ell$ places where we can insert ℓ \$ signs. Hence, there are $\binom{k+\ell}{\ell}$ distinct ways of multiplying k factors.

Therefore (using Lemma 6.12), the total number of distinct polynomials of degree less than t in G is greater than or equal to

$$\sum_{k=0}^{t-1} \binom{k+\ell}{\ell} = \binom{t-1+\ell+1}{\ell+1} = \binom{t+\ell}{\ell+1} = \binom{t+\ell}{t+\ell-(\ell+1)} = \binom{t+\ell}{t-1}.$$

Since any two polynomials of degree less than t are distinct in G, it follows that $|G| \geq \binom{t+\ell}{t-1}$. □

Recall that p is a prime divisor of n. In the case that n is not a power of p, i.e. $n \neq p^k$ for any $k \geq 1$, the size of G can also be bounded above by the following lemma.

Lemma 6.14. If n is not a power of p, then $|G| \leq n^{\sqrt{t}}$.

Proof. Recall that t is the size of the group I_r, the set of numbers in I modulo r. Consider the following subset of I:

$$\hat{I} = \left\{ \left(\frac{n}{p}\right)^i \cdot p^j \;\middle|\; 0 \leq i,j \leq \lfloor\sqrt{t}\rfloor \right\},$$

where the exponents i and j are now only allowed to run over the integers less or equal to $\lfloor\sqrt{t}\rfloor$.

If n is not a power of p, then none of the powers $\left(\frac{n}{p}\right)^i$ and p^j overlap. Since there are $\lfloor\sqrt{t}\rfloor + 1$ choices of i and $\lfloor\sqrt{t}\rfloor + 1$ choices of j, it follows that the set $\hat{I}$ has $(\lfloor\sqrt{t}\rfloor + 1)^2 > t$ distinct elements.

Since $|I_r| = t$, at least two numbers in $\hat{I}$ must be equivalent modulo r. Let m_1 and m_2 be two such numbers with $m_1 > m_2$. We then have

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

Let $f(X) \in P$. Since m_1 and m_2 are both in I, they are introspective for any polynomial in P. This gives

$$\begin{aligned}
[f(X)]^{m_1} &\equiv f(X^{m_1}) \pmod{X^r - 1, p} \\
&\equiv f(X^{m_2}) \pmod{X^r - 1, p} \\
&\equiv [f(X)]^{m_2} \pmod{X^r - 1, p}.
\end{aligned}$$

Since $h(X)$ divides $X^r - 1$, this implies that

$$[f(X)]^{m_1} \equiv [f(X)]^{m_2} \pmod{h(X), p},$$

so $[f(X)]^{m_1} = [f(X)]^{m_2}$ in the field $F = \mathbb{F}_p[X]/h(X)$.

Let $g(X) \in G$. Since $G \subseteq F$ and $G \subseteq P$, the above equalities still hold and we have that $g(X)$ is a root of the polynomial $Q'(Y) = Y^{m_1} - Y^{m_2}$.

Since $g(X)$ was an arbitrary element in G (which are all distinct in F), the polynomial $Q'(Y)$ has at least $|G|$ distinct roots in F.

The degree of $Q'(Y)$ is $m_1 \leq (\frac{n}{p} \cdot p)^{\lfloor \sqrt{t} \rfloor} \leq n^{\sqrt{t}}$. Since the number of roots of any polynomial is less than or equal to the degree, $|G| \leq n^{\sqrt{t}}$. □

With these bounds on $|G|$, we are ready to prove Proposition 6.5. One final fact about binomial coefficients is required.

Lemma 6.15. Let $m > 1$. Then $\binom{2m+1}{m} > 2^{m+1}$.

Proof. Observe that

$$\begin{aligned}
\binom{2m+1}{m} &= \frac{(2m+1)!}{m!(m+1)!} \\
&= \left(\frac{2m+1}{m+1}\right)\left(\frac{2m}{m}\right)\cdots\left(\frac{m+2}{2}\right)\left(\frac{m+1}{1}\right) \\
&= (2m+1)\cdot\left(\frac{2m}{m}\right)\left(\frac{2m-1}{m-1}\right)\cdots\left(\frac{m+2}{2}\right) \\
&> 2^2\cdot\underbrace{(2)(2)\cdots(2)}_{m-1 \text{ terms}} \qquad \text{when } m>1 \\
&= 2^{m+1}.
\end{aligned}$$

□

Proposition 6.5 (Restated). If the algorithm returns PRIME in line 7, then n is prime.

Proof. By Lemma 6.13, for $t = |I_r|$ and $\ell = \lfloor\sqrt{\phi(r)}\log n\rfloor$ we have:

$$\begin{aligned}
|G| &\geq \binom{t+\ell}{t-1} \\
&\geq \binom{\ell+1+\lfloor\sqrt{t}\log n\rfloor}{\lfloor\sqrt{t}\log n\rfloor} && \text{(since } t>\sqrt{t}\log n\text{)}^3 \\
&\geq \binom{2\lfloor\sqrt{t}\log n\rfloor+1}{\lfloor\sqrt{t}\log n\rfloor} && \text{(since } \ell=\lfloor\sqrt{\phi(r)}\log n\rfloor \geq \lfloor\sqrt{t}\log n\rfloor\text{)}^4 \\
&> 2^{\lfloor\sqrt{t}\log n\rfloor+1} && \text{(since } \lfloor\sqrt{t}\log n\rfloor > \lfloor\log^2 n\rfloor \geq 1\text{, Lemma 6.15)} \\
&\geq n^{\sqrt{t}}.
\end{aligned}$$

By Lemma 6.14, $|G| \leq n^{\sqrt{t}}$ if n is not a power of p. Since the above inequalities show $|G| > n^{\sqrt{t}}$, it follows that $n = p^k$ for some $k \geq 1$. However,

[3] $t > \log^2 n \implies \sqrt{t} > \log n \implies t > \sqrt{t}\log n$

[4] n and p generate I_r and are relatively prime to r, so every element of I_r is relatively prime to r, hence $t \leq \phi(r)$.

if $k \geq 2$ the algorithm would have returned COMPOSITE in line 1, via the perfect power test.

Therefore $k = 1$ and $n = p$, showing n is prime. □

From Propositions 6.3, 6.4, and 6.5, the main theorem of correctness follows.

6.4 Time Complexity

We will use the fact that the basic operations of addition, multiplication, and division of two $||n||$ bit numbers can be carried out in $O^{\sim}(||n||)$ time using sophisticated methods [3]. The reader is encouraged to consult Chapter 3 for descriptions of the algorithms used for other number theory computations.

By [3], the multiplication of two polynomials of degree less than or equal to r takes $O^{\sim}(r)$ integer multiplications using sophisticated methods.

Notation

As before, let $||n|| = \lfloor \log_2(n) \rfloor + 1$ be the number of bits in the binary representation of n.

Use log to denote the base 2 logarithm.

Line 1: We use the Perfect Power test (Algorithm 3.3), which takes $O^{\sim}(||n||^3)$ time.

Line 2: To find the appropriate r, we start at $r = 2$ and check if $n^k \not\equiv 1 \pmod{r}$ for every $k \leq \log^2 n$. If it becomes equivalent to 1 we know the order is too small, so we increment r by 1 and start over. For a given r, it therefore takes at most $O(\log^2 n)$ multiplications modulo r, which is $O^{\sim}(\log^2 n \log r)$. By Lemma 6.7, the maximum number of r's we need to try is $O(\log^5 n)$, meaning the total time complexity of Line 2 is $O^{\sim}(\log^7 n) = O^{\sim}(||n||^7)$.

Line 3: In this step, we compute the GCD of r numbers. Using the Euclidean Algorithm, each calculation is $O(||n||^2)$, hence the time complexity of the step is $O(r \cdot ||n||^2) = O(||n||^7)$.

Line 4: The simple check of $n \leq r$ is $O(||n||)$.

Lines 5 and 6: In this step, we verify $\lfloor \sqrt{\phi(r)} \log n \rfloor$ equations. Each equation involves computing $(X+a)^n \pmod{X^r - 1}$ in $\mathbb{Z}_n[X]$ and comparing it to the polynomial $X^{n \bmod r} + a$. We use a variant of the Fast Powering algorithm (Algorithm 3.2) for polynomials: squaring, multiplying, and reducing modulo n and $X^r - 1$ each time.

Note that reduction modulo $X^r - 1$ is trivial, simply replacing X^s with X^{s-r} whenever an exponent s, $r \leq s \leq 2r-1$ appears. Using the square and multiply method, each equation thus requires $O(\log n)$ multiplications of polynomials of degree $\leq r$. Each such multiplication takes $O^{\sim}(r)$ integer multiplications modulo n. Since the size of the coefficients in bits is $O(\log n)$, each equation can be verified in $O^{\sim}(r \log^2 n)$ steps. Therefore the total complexity of lines 5 and 6 is $O^{\sim}(r\sqrt{\phi(r)} \log^3 n) = O^{\sim}(r^{3/2} \log^3 n) = O^{\sim}(\log^{21/2} n) = O^{\sim}(||n||^{10.5})$.

Since lines 5 and 6 have the greatest time complexity, they asymptotically dominate the other steps, making the overall time complexity of the algorithm $O^{\sim}(\log^{21/2} n) = O^{\sim}(||n||^{10.5})$. As advertised, this shows the algorithm to be polynomial time in the number of the bits of the input.

6.5 Theory vs. Practicality

The AKS test was a major theoretical breakthrough in the field of primality testing. It was the first algorithm of its kind to be deterministic, general, unconditional, and polynomial time.

However, it is important to note that "polynomial time" does not equate with being fast in practice. In reality, AKS is still far too slow to be of use in testing the large numbers needed for cryptographic systems. Additionally, it does not require many runs of a probabilistic test such as Miller-Rabin before the difference between that and a deterministic test is completely negligible (see remark at the end of section 5.2.2).

Instead, the AKS algorithm should be admired for what it is: a clever, surprisingly elegant approach to a problem that proved what people did not know to be possible, that PRIMES is in **P**. It also serves as a beacon of hope to mathematicians and computer scientists alike that other seemingly intractable problems may one day be solved.

Acknowledgements

This paper would not have been possible without the help of many people. Thank you to my advisor, Professor David Guichard, for helping me select an interesting topic and assisting with my research. Thank you to Brooke Taylor for proofreading my work. Thank you also to Professor Pat Keef for your guidance and inspiration throughout the senior project class.

References

[1] M. Agrawal, N. Kayal, N. Saxena, *PRIMES is in P*, Annals of Mathematics 160, 781-793, 2004.

[2] J. Hoffstein, J. Pipher, J.H. Silverman, *An Introduction to Mathematical Cryptography*, Undergraduate Texts in Mathematics, Springer, Second Edition, 2014.

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

[4] P. Keef, D. Guichard, *An Introduction to Higher Mathematics*, Whitman College, 2010.

[5] T. Hungerford, *Algebra*, Graduate Texts in Mathematics, Springer, 1974.

[6] J. Gallian, *Contemporary Abstract Algebra*, Seventh Edition, Brooks/Cole, 2010.

[7] H. Wilf, *Algorithms and Complexity*, University of Pennsylvania, Internet Edition, 1994.

[8] R. Mollin, *A Brief History of Factoring and Primality Testing B. C. (Before Computers)*, Mathematics Magazine 75, 18-29, 2002.

[9] R. Lidl, H. Niederreiter, *Introduction to Finite Fields and their Applications*, Cambridge University Press, 1986.

[10] M. Nair, *On Chebyshev-type inequalities for primes*, American Mathematical Monthly 89, 126-129, 1982.