THE LUCAS-LEHMER, MILLER-RABIN, AND AKS PRIMALITY TEST

INHO RYU

Contents

Date: July 17, 2023.

Abstract. In this paper, we explored three different primality tests: Lucas-Lehmer, Miller-Rabin, and AKS. First we briefly talked of their histories and applications. Then we talked of modular arithmetic and its properties, and we also talked of the repeated squaring algorithm. Then we then explored the tests themselves, which involved looking at each test's algorithm, proof, time complexities, pseudocode, the different variants of the primality tests, and examples.

1. Introduction

Prime numbers and, as such, primality tests, play a pivotal role in various areas of mathematics, computer science, and cryptography. The three primality tests covered are the Lucas-Lehmer, Miller-Rabin, and AKS primality test. These three primality tests are very different in nature, as the Lucas-Lehmer only tests Mersenne numbers, the Miller-Rabin can only tell if a number is probably prime, and AKS can deterministically test all numbers.

The Lucas-Lehmer test was originally developed by Édouard Lucas in 1876 and was then improved by Derrick Henry Lehmer in the 1930s. The Lucas-Lehmer test can test only Mersenne numbers, which are numbers of the form $M_n = 2^n - 1$ where $n \in \mathbb{Z}^+$ and $n \geq 2$. The test is comprised of a very simple algorithm which can be executed rapidly. The test goes as follows.

Theorem 1.1. *Define a sequence $\{s_i\}$ for all $i \geq 0$ by*

$$s_i = \begin{cases} 4 & \text{if } i = 0; \\ s_{i-1}^2 - 2 & \text{otherwise.} \end{cases}$$

M_p is prime if and only if

$$s_{p-2} \equiv 0 \pmod{M_p}$$

If M_p is prime, then we call it a Mersenne prime. Mersenne primes are quite significant in mathematics, as there are currently only 51 known Mersenne primes, and yet the 6 largest known prime numbers are all Mersenne primes. The search for the next largest prime number, and so subsequently Mersenne primes, have sparked the creation of the Great Internet Mersenne Prime Search (GIMPS), a collaborative project of people who use software on their PCs in order to find Mersenne primes. Anyone from around the world with a powerful enough PC is able to contribute their computing power in search of Mersenne primes.

The Lucas-Lehmer test has been instrumental in GIMPS because of its speed and exclusivity to Mersenne numbers, although GIMPS did switch to a Fermat probable prime test in 2018 due to computer hardware errors that may occur with the Lucas-Lehmer test. Now they only use the Lucas-Lehmer test for probable primes generated by the Fermat probable prime test as a double-check method.

The Miller-Rabin test was discovered by Gary L. Miller in 1976, first as a deterministic test, which was called the Miller test. The Miller test was significant in mathematics as it was the first to be a fully deterministic test which runs in polynomial time over all inputs, but the one major caveat was that it relied on the unproven generalized Riemann hypothesis.

Michael O. Rabin then modified the Miller test into an unconditional probabilistic algorithm in 1980. In other words, Rabin's modification of the Miller test does not confirm whether a number is prime, but rather, only confirms whether a number is *not* prime. The Miller-Rabin test consists of four steps.

Theorem 1.2. *Given an integer $n \geq 5$, this algorithm outputs either true or false. If it outputs true, then n is probably prime, and if it outputs false, then n is definitely composite.*

1. *Compute the unique integers m and k such that m is odd and $n - 1 = 2^k \cdot m$.*
2. *Choose a random integer a with $1 < a < n$.*
3. *Set $b = a^m \pmod{n}$. If $b \equiv \pm 1 \pmod{n}$ output true and terminate.*
4. *If $b^{2^r} \equiv -1 \pmod{n}$ for any r with $1 \leq r \leq k - 1$, output true and terminate. Otherwise output false.*

The simple nature and speed of the test makes it highly applicable, although as mentioned before, a Fermat probable prime test is used for GIMPS. A common followup test to the Miller-Rabin test is the Lucas-Lehmer test.

The AKS test is a deterministic primality-proving algorithm created and published by Manindra Agrawal, Neeraj Kayal, and Nitin Saxena on August 6, 2002.

The discovery of AKS is important because of 4 main reasons: its ability to verify the primality of any general number (unlike the Lucas-Lehmer test), the maximum running time of the algorithm can be bounded by a polynomial over the number of digits in the target number, the algorithm is guaranteed to distinguish deterministically whether the target number is prime or composite (unlike the Miller-Rabin test), and the correctness of AKS is not conditional on any subsidiary unproven hypothesis (unlike the Miller test).

While the theoretical importance of AKS is immense, the practical use of AKS is very small as other algorithms or primality tests are far superior in terms of performance and verification of primality, and the time benefits of AKS are only relevant for extremely large numbers, to the point where it's not practical in use.

The basic idea of the AKS test is as follows.

Theorem 1.3. *Suppose n is a natural number, and a an integer coprime to n. The number n is prime if and only if the relation*

$$(x + a)^n = x^n + a \quad \textit{in } (\mathbb{Z}/n\mathbb{Z})[x]$$

holds

In other words, if in $(x + a)^n - (x^n + a)$, all of the coefficients are multiples of n, then n is prime.

Section 2 will cover the main definitions and some necessary background information needed for the paper.

Section 3, 4, and 5 will cover the Lucas-Lehmer, Miller-Rabin, and AKS test, respectively.

Acknowledgements

I would like to thank Ryan Catullo and Simon Rubinstein-Salzedo for helpful conversations and insights on writing this paper.

2. Background

There are some terms that are important to know when covering the tests.

A perfect number is a positive integer that is equal to the sum of its positive divisors, excluding the number itself. There are no known odd perfect numbers.

Computational complexity of an algorithm is the amount of resources required to run it. Computational complexity of a problem is the complexity of the best algorithms that allow solving the problem.

In the Lucas-Lehmer test, the Lucas-Lehmer Residue is the remainder after the modulo function for each iteration of the algorithm is called the Lucas-Lehmer residue.

In the Miller-Rabin test, if n is composite and is said to be a strong probable prime to base a, it is called a strong pseudoprime. In the Miller-Rabin test, if we use base a and n is a pseudoprime, a is called a strong liar. By contrast, if n is declared definitely composite to base a, then a is called a witness for the "compositeness" of n.

2.1. **Modular Arithmetic.**

Modular arithmetic is used heavily in all the primality tests covered. Modular arithmetic is a system of arithmetic for integers, where numbers "wrap around" after reaching a certain value, called the modulus.

Time as seen on a 12-hour clock is one example, where the day is split into 12-hour periods. If it were 9:00 now, then after 5 hours it would be 2:00, not 14:00, as the clock "wraps around" every 12 hours. Because the hour number starts over when it reaches 12, this is arithmetic modulo 12. It would be said that 14 is congruent to 2 modulo 12 for the above example.

2.1.1. *Congruence.*

Given an integer $n > 1$, called a modulus, two integers a and b are said to be congruent modulo n if n is a divisor of their difference. This is also shown as if there is an integer k such that $a - b = kn$.

Congruence modulo n is a congruence relation, meaning that it is an equivalence relation that one could do the operations of addition, subtraction, and multiplication. Congruence modulo n is denoted as:

$$a \equiv b \pmod{n}.$$

The parentheses means that $\pmod{n}$ applies to the entire equation, and not just to the right hand side (in this case b).

What the statement $a \equiv b \pmod{n}$ asserts is that a and b have the same remainder when divided by n. That is,

$$a = pn + r,$$

$$b = qn + r,$$

where $0 \leq r < n$ is the common remainder. Subtracting these two expressions, we can find the relation relation:

$$a - b = kn, \tag{2.1}$$

by setting $k = p - q$.

It is important the difference between this notation and the one without parenthesis, as without parenthesis it would refer to the modulo operation, which would be denoted as:

$$a \equiv b \mod n.$$

This from denotes the unique integer a such that $0 \leq a \leq n$ and $a \equiv b \pmod{n}$, or in other words, b is the remainder of a when divided by n. The congruence relation can be rewritten as:

$$a = kn + b,$$

explicitly showing its relationship with Euclidean division. Notice how this equation is equivalent to 2.1.

2.1.2. *Fermat's Little Theorem.* One of the advanced properties of modular arithmetic is Fermat's little theorem. Fermat's little theorem is the basis of the Fermat primality test, the main primality test used for GIMPS. Fermat's Little Theorem goes as follows.

Lemma 2.1. *If p is a prime number, then for any integer a, the number of $a^p - a$ is an integer multiple p.*

In the notation of modular arithmetic, this is expressed as

$$a^p \equiv a(\pmod{p}).$$

Also note that if a is not divisible by p (a is a coprime to p), then Fermat's little theorem is equivalent to saying that $a^{p-1} - 1$ is an integer multiple of p, or in the notation of modular arithmetic is equivalent to

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

Proof. Assume that a is positive and not divisible by p. If we write down the sequence of numbers

$$a, 2a, 3a, \ldots, (p-1)a \tag{2.2}$$

and reduce each one by modulo p, the resulting sequence turns out to be

$$1, 2, 3, \ldots, p-1. \tag{2.3}$$

If we multiply together the numbers in each sequence, then the results must be identical modulo p:

$$a \times 2a \times 3a \times \cdots \times (p-1)a \equiv 1 \times 2 \times 3 \times \cdots \times (p-1) \pmod{p}.$$

If we collect the a terms, we get

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

Cancelling out $(p-1)!$ from both sides, we get

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

■

There are two steps in this proof that needs justification: why the elements in 2.2 can be reduced into the rearrangement of 2.3, and why it is valid to cancel out terms in modular arithmetic.

To address the first point, let's do an example.

Example. If $a = 3$ and $p = 7$, then the sequence is

$$3, 6, 9, 12, 15, 18.$$

Reducing modulo 7 gives

$$3, 6, 2, 5, 1, 4,$$

which can be rearranged as

$$1, 2, 3, 4, 5, 6.$$

Multiplying them together gives

$$3 \times 6 \times 9 \times 12 \times 15 \times 16 \equiv 3 \times 6 \times 2 \times 5 \times 1 \times 4 \equiv 1 \times 2 \times 3 \times 4 \times 5 \times 6 \pmod 7$$

and that is

$$3^6(1 \times 2 \times 3 \times 4 \times 5 \times 6) \equiv (1 \times 2 \times 3 \times 4 \times 5 \times 6) \pmod 7.$$

Cancelling out $1 \times 2 \times 3 \times 4 \times 5 \times 6$ gives

$$3^6 \equiv 1 \pmod 7,$$

which is Fermat's litte theorem for the case $a = 3$ and $p = 7$.

Now to address the cancellation, in what I will call the cancellation law. What we're trying to prove here is that if u, w, and y are integers, and u is not divisible by prime number p, and if

$$ux \equiv uy \pmod p, \tag{2.4}$$

then we may cancel u to get

$$x \equiv y \pmod p. \tag{2.5}$$

Our use of the cancellation law in the proof was valid as the numbers $1, 2, \ldots, p-1$ are not divisible by p, as they are smaller than p.

To prove the cancellation law, we can use Euclid's lemma, which generally states that if a prime p divides a product ab (where a and b are integers), then p must divide a or b. 2.4 simply means that p divides $ux - uy = u(x - y)$. Since p is a prime which does not divide u, Euclid's lemma tells us that it must divide $x - y$ instead, or in other words, 2.5 holds.

It is important to see that the conditions under when you can use cancellation law are quite strict, and that is why Fermat's little theorem needs p to be prime. This can be seen as for example $2 \times 2 \equiv 2 \times 5 \pmod 6$, but $2 \not\equiv 5 \pmod 6$. However, there is a generalized version of the cancellation law. It goes as follows: if u, x, y, and z are integers, if u and z are relatively prime, and if

$$ux \equiv uy \pmod z,$$

then we may cancel u to obtain

$$x \equiv y \pmod z.$$

This follows from a generalization of Euclid's lemma.

Finally, we must explain why

$$a, 2a, 3a, \ldots, (p-1)a,$$

when reduced modulo p, becomes a rearrangment of the sequence

$$1, 2, 3, \ldots, p-1.$$

To start, none of the terms $a, 2a, 3a, \ldots, (p-1)a$ can be congruent to 0 modulo p, since if k is one of the numbers $1, 2, 3, \ldots, p-1$, then k is relatively prime with p, and so is a, so Euclid's lemma tells use that ka shares no factor with p. Therefore, we at least know that the numbers $a, 2a, 3a, \dots, (p-1)a$, when reduced modulo p, must be found among the numbers $1, 2, 3, \dots, p-1$.

We can also see that the numbers $a, 2a, 3a, \dots, (p-1)a$ must all be distinct after reducing them modulo p since if

$$ka \equiv ma \pmod{p},$$

where k and m are one of $1, 2, 3, \dots, p-1$, then the cancellation law tells us that $k \equiv m \pmod{p}$.

Since both k and m are in between 1 and $p-1$, they must be equal. Therefore, the terms $a, 2a, 3a, \dots, (p-1)a$ when reduced modulo p must be distinct.

Fermat's little theorem is important for the Miller-Rabin test as the Miller-Rabin test uses the following extension of Fermat's little theorem:

Lemma 2.2. *If p is an odd prime and $p-1 = 2^s d$ with $s > 0$ and d odd > 0, then for every a coprime p, either $a^d \equiv 1 \pmod{p}$ or there exists r such that $0 \leq r < s$ and $a^{2^r d} \equiv -1 \pmod{p}$*

This will be expanded upon in 4

2.2. **Time Complexity.**

Time complexity is the computational complexity that describes the amount of computer time it takes to run an algorithm. Time complexity is typically estimated by counting the amount of elementary operations performed by the algorithm, assuming that each elementary operation takes a fixed amount of time to perform. The amount of time taken and the number of elementary operations performed by the algorithm are taken to be related by a constant factor, or big O notation.

Since the running time of the algorithm may vary among different inputs of the same size, it is common to consider the worst-case time complexity, or the maximum amount of time required for inputs of a given size. A less common method is the average-case complexity, which is the average of the time taken on inputs of a given size, which is possible as there are a finite number of possible inputs of a given size. In both cases, the time complexity is generally expressed as a function of the size of the input. Since this function is generally difficult to compute exactly, and the running time for small inputs is usually not important, one commonly focuses on the behavior of the complexity when the input size increases, or in other words, the asymptotic behavior of the complexity. Therefore, the time complexity is commonly expressed using big O notation, such as $O(n), O(n \log n), O(n^a), O(2^n)$, etc. where n is the size in units of bits needed to represent the input.

Algorithmic complexities are classified according to the type of function appearing in the big O notation. For example, an algorithm with time complexity $O(n)$ is a linear time algorithm and an algorithm with time complexity $O(n^\alpha)$ for some constant $\alpha > 1$ is a polynomial time algorithm, to name a few.

2.2.1. *Repeated Squaring.*

An algorithm for a computational problem is a computational method, a procedure, that takes in the inputs and spits out an output that obeys the mathematical relation defined in the computational problem.

There are two algorithms for the modular exponentiation problem, one we can call the naive algorithm and one we call the repeated squaring algorithm. The naive algorithm simply starts with b and multiplies by b over and over, calculating the result $\pmod{m}$ each time. After $k-1$ multiplications, the result is $b^k \pmod{m}$.

The repeated squaring algorithm consists of two parts. In the first part, the algorithm starts with b, then multiplies it by itself ("squaring" it) $\mod m$, then squares the result $\mod m$, and then squares that $\mod m$, and so on until the power is greater than or equal to the original power. In the second part, the algorithm combines together some of these results, multiplying them together $\mod m$. In order to make the algorithm more precise, we need the notion of binary expansion.

We ordinarily write numbers in base 10 (decimal) notation. The rightmost digit is in the ones place, the second-to-rightmost digit is in the tens place, the third-to-rightmost digit is in the hundreds place, and so on. Thus 765 represents 7 times 10^2 plus 6 times 10^1 plus 5 times 10^0.

Another system for writing numbers is base 2 (binary), where the rightmost digit is in the ones place, the second-to-rightmost digit is in the twos place, the third-to-rightmost digit is in the fours place, the fourth-to-rightmost digit is in the eights place, and so on. For example, 100111 represents 1 times 2^5 plus 0 times 2^4 plus 0 times 2^3 plus 1 times 2^2 plus 1 times 2^1 plus 1 times 2^0, which adds up to be 39 in decimal notation. The digits in binary notation are called bits (short for binary digits). By convention, the leftmost bit must be a 1 (just as we don't normally write the number 47 as 047).

Back to the repeated-squaring algorithm. To calculate $b^k \pmod m$, we write k in binary. The position of the leftmost bit tells us how many squarings need to take place in the first part of the algorithm. For example, if the leftmost bit is in the 2^{11}s place, then we need to calculate $b^{2^0}, b^{2^1}, b^{2^2}, b^{2^3}, \dots, b^{2^{11}}$. Since the first number is just b, we need 11 squarings.

The 1's in the binary representation of k tell us which powers of b that we have calculated we need to combine to get the final result.

Say the exponent k takes L bits to represent. Then the number of multiplications for the first part of the algorithm is $L-1$. In the second part of the algorithm, we need to multiply together some of the L results we obtained in the first part. Since we need to multiply together at most L values, we need at most $L-1$ multiplications in the second part of the algorithm.

The number of bits needed to represent a positive integer k is $1 + \lfloor \log_2 k \rfloor$ (the base-2 logarithm of k). For example, $\log_2 2184 = 11.092757\cdots$, so the rounded-down value of the logarithm is 11. Thus this formula predicts that the number of bits needed to represent 2184 is 12. For most purposes, it is good enough to neglect the "1+" and estimate the number of bits by $\log_2 k$.

Similarly, the number of decimal digits needed to represent a number, say m, is 1 plus $\lfloor \log_1 0k \rfloor$, the rounded down value of the base-10 logarithm. For example, $\log_1 0765 = 2.8836614\cdots$, so the rounded-down value is 2. Thus, this formula predicts that the number of decimal digits to represent 765 is 3.

It is easy to convert between the base-10 log of a number and its base-2 log using the following formula.

$$\log_2 x \approx 3.3 \log_{10} x$$

3.3 is the base-2 logarithm of 10.

Example. Say we wish to know the value $3^{200} \mod 50$.

(1) $3^1 = 3 \mod 50 \to 3 \mod 50$
(2) $3^2 = 9 \mod 50 \to 9 \mod 50$
(3) $3^4 = 81 \mod 50 \to 31 \mod 50$
(4) $3^8 = 961 \mod 50 \to 11 \mod 50$

(5) $3^{16} = 121 \mod 50 \to 21 \mod 50$
(6) $3^{32} = 441 \mod 50 \to 41 \mod 50$
(7) $3^{64} = 1681 \mod 50 \to 31 \mod 50$
(8) $3^{128} = 961 \mod 50 \to 11 \mod 50$
(9) 3^{256}, but exponent is larger than initial, so halt.

We can rewrite 200 in binary as 11001000. In other words (or not words), 200 = 128+64+8. By looking at where the 1's are in the binary, we can see that $3^{200} \mod 50 = 3^{128+64+8} \mod 50 = 3^{128}3^{64}3^{8} \mod 50$, and if we replace the factors with what they are equal to mod 50, we get $(11)(31)(11) \mod 50 = 3751 \mod 50 = 1 \mod 50$.

2.2.2. *Wilson's Theorem.* This section will talk about Wilson's test. While not a main primality test to be talked about, it is included as to show how fast the Lucas-Lehmer, Miller-Rabin, and AKS test compared to Wilson's theorem.

Wilson's theorem goes as follows.

Lemma 2.3. *A natural number $n > 1$ is a prime number if and only if the product of all the positive integers less that n is one less than a multiple of n. In the notation of modular arithmetic the factorial $(n-1)!$ satisfies*

$$(n-1)! \equiv -1(\pmod n)$$

exactly when n is a prime number. In other words, any number n is a prime number if and only if $(n-1)!+1$ is divisible by n.

Proof. If n is composite, it is divisible by some prime number q, where $2 \leq q \leq n-2$. Because q divides n, let $n - qk$ for some integer k. Suppose for the sake of contradiction that $(n-1)!$ were congruent to $-1 \pmod n$ where n is composite. Then $(n-1)!$ would be congruent to $-1 \pmod q$ as $(n-1)! \equiv -1 \pmod n$ implies that $(n-1)! = nm - 1 = (qk)m - 1 = q(km) - 1$ for some integer m which shows $(n-1)!$ being congruent to $-1 \pmod q$. But $(n-1)! \equiv 0 \pmod q$ byt the fact that q is a term in $(n-1)!$ making $(n-1)!$ a multiple of q. A contradiction is now reached. ■

Wilson's theorem used as a primality test is useless because computing $(n-1)!$ modulo n for large n is computationally complex, and many much faster primality tests are known, even including trial division.

While if used to determine the priamltiy of the successors of large factorials, it is a fast and effective method, but that is not what we are looking for here.

3. Lucas-Lehmer Test

The Lucas-Lehmer test is a primality test exclusively for Mersenne numbers. Mersenne numbers are numbers of the form $M_n = 2^n - 1$ where $n \in \mathbb{Z}^+$ and $n \geq 2$. The Lucas-Lehmer test is the main test used in GIMPS because of its speed compared to other similar primality tests and also for its exclusiveness for Mersenne numbers.

3.1. **Mersenne Primes.** There are a few additional things to note about Mersenne primes. Mersenne primes were named after Marin Mersenne, who studied them in the early 17th century. They were since studied thoroughly due to their close connection with perfect numbers as seen in the Euclid-Euler theorem (see theorem 3.2).

There are also some additional theorems about Mersenne numbers/primes that are notable.

Lemma 3.1. *If n is prime, then $2^n - 1$ is also prime.*

Proof. Suppose that p is composite, therefore p can be written as $p = ab$ with a and $b > 1$. Then $2^p - 1 = 2^a b - 1 = (2^a)^b - 1 = (2^a - 1)((2^a)^{b-1} + (2^a)^{b-1} + (2^a)^{b-2} + \ldots + 2^a + 1$ so $2^p - 1$ is composite. By contraposition, if $2^p - 1$ is prime, then p is prime. ■

Therefore, an equivalent definition of Mersenne primes would be $M_p = 2^p - 1$ for some prime p.

Many questions about Mersenne primes remain unsolved. One such question is whether the set of Mersenne primes is finite or infinite. The Lenstra-Pomerance-Wagstaff conjecture asserts that there are infinite many Mersenne primes and even predicts their order of growth, although the predictions do seem inaccurate. Another unsolved question that is pondered about is whether there are infinitely many Mersenne numbers with prime exponents that are composite.

Mersenne primes, as mentioned before, also have a close connection with perfect numbers. In the 4th century BC, Euclid proved the following:

Theorem 3.2. *If $2^p - 1$ is prime, then $2^{p-1}(2^p - 1)$ is a perfect number.*

In the 18th century, Leonhard Euler proved that all even perfect numbers have this form. This is known as the Euclid-Euler theorem. Thus it is shown that all even perfect numbers are the product of a power of two and a Mersenne prime.

3.2. **The Test.** For this paper I will set 4 as s_0 (the starting value), although the possibility of different starting values will be covered in 3.4.

The Lucas-Lehmer test works as follows. Let $M_p = 2^p - 1$ be the Mersenne number to test with p an odd prime. The primality of p can be efficiently checked with a simple algorithm like trial division since p is exponentially smaller than M_p. Define a sequence $\{s_i\}$ for all $i \geq 0$ by

$$s_i = \begin{cases} 4 & \text{if } i = 0; \\ s_{i-2}^2 - 2 & \text{otherwise.} \end{cases}$$

M_p is prime if and only if

$$s_{p-2} \equiv 0 \mod M_p$$

3.3. **Pseudocode.** The algorithm can be written in pseudocode as follows.

```
Lucas-Lehmer(p)
    var s = 4
    var M = 2^p-1
    repeat p - 2 times:
        s = ((s x s) - 2) mod M
    if s == 0 return PRIME else return COMPOSITE
```

3.4. **Alternative Starting Values.** Starting values s_0 other than 4 are possible, and the Lucas-Lehmer residue calculated with these alternative starting values will still be zero if M_p is a Mersenne prime. However, the terms of the sequence will be different and if M_p is not prime then the Lucas-Lehmer residue will be different from when calculated with $s_0 = 4$.

Some examples of universal starting values, as in they are valid for all (or nearly all) p, are 4, 10, and $(2 \pmod{M_p})(3 \pmod{M_p})^{-1}$, which is usually denoted by $2/3$ for short. $2/3$ equals $(2^p + 1)/3$, which is the Wagstaff number with exponent p.

3.5. **Sign of the Penultimate Term.** The sign of penultimate term is important to know in case there may be a pattern of signs in the penultimate terms, and although currently there does not seem to be a set sequence.

If $s_{p-2} = 0 \pmod{M_p}$ then the penultimate is $s_{p-3} = \pm 2^{(p+1)/2} \pmod{M_p}$. The sign of this penultimate term is called the Lehmer symbol $\epsilon(s_0, p)$.

In 2000 S.Y. Gebre-Egziabher proved that for the starting value $2/3$ and for $p \neq 5$ the sign is:

$$\epsilon(\frac{2}{3}, p) = (-1)^{p-1/2}$$

He also proved that the Lehmer symbols for starting values 4 and 10 when p is not 2 or 5 are related by:

$$\epsilon(10, p) = \epsilon(4, p) \times (-1)^{(p+1)(p+3)/8}$$

So $\epsilon(4, p) \times \epsilon(10, p) = 1$ iff $p = 5$ or $7 \pmod 8$ and $p \neq 2, 5$.

3.6. **Time Complexity.** In the algorithm for the Lucas-Lehmer test, there are two operations during each iteration: the multiplication of $s \times s$ and the $\pmod M$ operation.

The $\pmod M$ operation can be made efficient on basic computers by observing the following.

$$k \equiv (k \pmod{2^n}) + \lfloor k/2^n \rfloor (\pmod{2^n - 1}).$$

3.7. **Examples.**

Example. Suppose we want to know whether Mersenne number $M_7 = 2^7 - 1 = 127$ is prime or composite. The test goes as follows.

(1) $s_0 = 4 \pmod{127}$.
(2) $s_1 = (4^2 - 2) \pmod{127} = 14 \pmod{127}$
(3) $s_2 = (14^2 - 2) \pmod{127} = 67 \pmod{127}$
(4) $s_3 = (67^2 - 2) \pmod{127} = 42 \pmod{127}$
(5) $s_4 = (42^2 - 2) \pmod{127} = 111 \pmod{127}$
(6) $s_5 = (111^2 - 2) \pmod{127} = 0 \pmod{127}$

Therefore, 127 is prime.

Example. Now let's suppose we want to know whether Mersenne number $M_4 = 2^4 - 1 = 15$ is prime or composite. The test goes as follows.

(1) $s_0 = 4 \pmod{15}$.
(2) $s_1 = (4^2 - 2) \pmod{15} = 14 \pmod{15}$
(3) $s_2 = (14^2 - 2) \pmod{15} = 14 \pmod{15}$
(4) $s_2 = (14^2 - 2) \pmod{15} = 14 \pmod{15}$
(5) ...

As shown, the algorithm is put into a loop and therefore $0/Mod15$ will never occur, meaning 15 is composite.

3.8. **Proof.** We will prove the Lucas-Lehmer test by assuming a Mersenne prime is composite, and prove that there's a contradiction.

Let $\omega = 2+\sqrt{3}$ and $\overline{\omega} = 2-\sqrt{3}$.

Note that $\omega\overline{\omega} = (2+\sqrt{3})(2-\sqrt{3}) = 2^2 - \sqrt{3}^2 = 4-3 = 1$

Lemma 3.3. $S_m = \omega^{2m-1} + \overline{\omega}^{2m-1}$

This is found by induction. So $S_{m-1}^2 = (\omega^{2^{m-2}} + \overline{\omega}^{2^{m-2}})^2 = \omega^{2^{m-1}} + \overline{\omega}^{2^{m-1}} + 2(\omega\overline{\omega})^{2^{m-2}} = \omega^{2m-1} + \overline{\omega}^{2m-1} + 2 = S_m + 2$. So $S_m = S_{m-1}^2 - 2$. It also follow that if M_p divides S_{p-1}, then $\omega^{2^{p-2}} + \overline{\omega}^{2^{p-2}} \equiv 0 \pmod{p}$. Explicit;y, we write $\omega^{2^{p-2}} + \overline{\omega}^{2^{p-2}} = RM_p$ for $R \in \mathbb{Z}$. Multiplying this identity by $\omega^{2^{p-2}}$, we get $\omega^{2^{p-2}} = RM_p\omega^{2^{p-2}} - 1$. Now squaring, $\omega^{2^p} = (RM_p\omega^{2^{p-2}} - 1)^2$. Here is where we assume that M_p is composite, as well as choose a prime divisor q with $q^2 \leq M_p$. Note that we don't want $q = 2$.

Lemma 3.4. *Let X be a set with a binary operation which is associative and has an identity. Then the set X^* of invertible elements in X forms a group.*

To prove this, we want to notice $1 \in X^*$, so X^* is a non-empty set. Now, we want to show that X^* is closed under the binary operation. Consider the invertible elements x_1 and x_2 with inverses x_1^{-1} and x_2^{-1}. Thus we see that x_1x_2 has the inverse $x_1^{-1}x_2^{-1}$.

Lemma 3.5. *If G is a finite group then the order of an element is at most the order of the group. If $x \in G$ and $x^r = 1$ then the order of x divides r.*

The order of a group is the number of elements in that group and the order of a ω is the smallest positive integer r such that $\omega^r = 1$. So, if $\omega^r = 1$, then the period of $\omega | r$. To prove this, consider a group that has 3 elements a, b, and 1. If $a \neq aa \neq aaa \neq aaaa$ we have already 4 different elements in the group which is a contradiction since the order of the group is 3. this the order of $a \leq 3$.

Proof. Now we can start the proof. Let Z_q denote the set of integers modulo q, and X denote the set $a + b\sqrt{3} | a, b \in Z_q$. We define two binary operations on X, addition and multiplication, in the traditional manner. For multiplication, we wan to choose representatives in $Z[\sqrt{3}]$ of our elements of X and compute the product. So $(a_1 + b_1\sqrt{3})(a_2 + b_2\sqrt{3}) = (a_1a_2 + 3b_1b_2) + (a_1b_2 + a_2b_1)\sqrt{3}$. Now we want to reduce to coefficients modulo q. We see that we have a commutative group in the case of addition and an associative and commutative group int he case of multiplication with the identity 1.

Let X^* denote the group of invertible elements of X with respect to multiplication. From 3.4 we can see that this is a group. In addition, from 3.5 we can see that the order of any element X^* is at most $q^2 - 1$, since X^* contains at least one non-invertible element, namely 0.

Now let's consider $\omega = 2 + \sqrt{3}$ as an element of X. Since q divides M_p, we see that $RM_p\omega^{2^{p-2}}$, when viewed as an element of X, is 0. From before $\omega^{2^{p-1}} = RM_p\omega^{2^{p-2}} - 1 = -1$ and $\omega^{2^p} = (RM_p\omega^{2p-2} - 1)^2 = (-1)^2 = 1$. So, $\omega \in X^*$ and has order 2^p. By 3.5, the order of $\omega | 2^p$, but cannot be less that 2^p. So, using 3.5 we get that $2^p \leq q^2 - 1$. However, $q^2 - 1 \leq M_p - 1 = 2^p - 2$, thus we have reached a contradiction. ■

4. Miller-Rabin Test

The Miller-Rabin test is a probabilistic primality test, as in when given an integer $n \geq 5$, the algorithm outputs whether n is *probably* prime (true) or whether n is *definitely* not prime (false). Historically, the Miller-Rabin test was significant in the search for a polynomial-time deterministic primality test. It is widely used as it is one of the simplest and fastest tests known. The Miller-Rabin test saves countless amounts of computing time and power as it narrows down the list of possible primes, and works extremely hand-in-hand with other fast primality tests such as the Lucas-Lehmer (for Mersenne numbers only) and AKS.

4.1. **Euler's Criterion and Root Bound.**

While not important to know for performing the Miller-Rabin test, these theorems are important for the proof of the Miller-Rabin test.

4.1.1. *Euler's Criterion.* Let p be an odd prime and a and integer not divisible by p. Using primitive roots, Euler managed to show that $(\frac{a}{p})$ is congruent to $a^{p-1/2}$ modulo p.

Proposition 4.1. *We have* $(\frac{a}{p}) = 1$ *if and only if*

$$a^{p-1/2} \equiv 1 \pmod{p} \tag{4.1}$$

4.1.2. *Root Bound.* This proposition shows that a polynomial of degree d over a field, such as $\mathbb{Z}/p\mathbb{Z}$, can have at most d roots.

Proposition 4.2. *Let* $f \in k[x]$ *be a nonzero polynomial over a field* k*. Then there are at most* $\deg(f)$ *elements* $a \in k$ *such that* $f(a) = 0$*.*

4.2. **The Test.**

Given an integer $n \geq 5$, this algorithm outputs either true or false. If it outputs true, then n is probably prime, and if it outputs false, then n is definitely composite.

1. Compute the unique integers m and k such that m is odd and $n - 1 = 2^k \cdot m$.
2. Choose a random integer a with $1 < a < n$.
3. Set $b = a^m \pmod{n}$. If $b \equiv \pm 1 \pmod{n}$ output true and terminate.
4. If $b^{2^r} \equiv -1 \pmod{n}$ for any r with $1 \leq r \leq k - 1$, output true and terminate. Otherwise output false.

This version of the test involves choosing the base a at random. A deterministic version of this would involve trying all possible bases, but that would result in an inefficient deterministic algorithm, where another test such as the Miller test would be more efficient.

4.3. **Choices of Bases.**

No composite number is a strong pseudoprime to all bases at the same time (put thing for this), but there is no known simple way of finding a witness. One way is to try all possible bases, which would be deterministic, but this is inefficient and the Miller test would be a better variant for this task.

Another solution is to pick a base at random as is established in the Miller-Rabin test. This version yields a fast probabilistic test, as when n is composite, most bases are witnesses, meaning the test will detect n as composite with a high probability. We can reduce the chance of a false positive by testing more bases. There seems to be diminishing returns in trying many bases, because if n is a pseudoprime to some base, then it seems more likely to be a pseudoprime to another base.

4.4. **Proof.**

Proof. We will prove that the algorithm is correct, but we will not cover the accuracy of the test (see Accuracy). We must prove that if the algorithm pronounces an integer n composite, then n really is composite. So suppose n is prime, yet the algorithm outputs that n composite. Then $a^m \not\equiv \pm 1 \pmod{n}$, and for all r with $1 \leq r \leq k - 1$ we have $a^{2^{k-1}} \not\equiv -1 \pmod{n}$. Since n is prime and $2^{k-1}m = (n-1)/2$, 4.1.1 implies that $a^{2^{k-1}m} \equiv \pm 1 \pmod{n}$, so by our hypothesis $a^{2^{k-1}m} \equiv 1 \pmod{n}$. But then $(a^{2^{k-1}m})^2 \equiv 1 \pmod{n}$, so by 4.1.2, we have $a^{2^{k-2}m} \equiv \pm 1 \pmod{n}$. Again, by our hypothesis, this implies $a^{2^{k-2}m} \equiv 1 \pmod{n}$. Repeating this argument inductively, we see that $a^m \equiv \pm 1 \pmod{n}$, which contradicts our hypothesis on a. ■

4.5. **Examples.**

Example. Suppose we play dumb and wish to know whether 11 is prime or composite. If we go through the steps, it goes as shown.

1. Compute the unique integers m and k such that m is odd and $n - 1 = 2^k \cdot m$.
   (a) We set $n = 11$, $k = 1$, and $m = 5$. The values for k and m are the only ones possible. $11 - 1 = 2^1 \cdot 5$
2. Choose a random integer a with $1 < a < 11$.
   (a) Set $a = 6$, as it falls under $1 < a < 11$.
3. Set $b = a^m \pmod{n}$. If $b \equiv \pm 1 \pmod{n}$ output true and terminate.
   (a) $b = 6^5 \pmod{11}$. $b \equiv -1 \pmod{11}$. Therefore, 11 is probably prime and terminate the process.

Example. Now let's again play dumb and wish to know whether 15 is prime or composite.

1. Compute the unique integers m and k such that m is odd and $n - 1 = 2^k \cdot m$.
   (a) We set $n = 15$, $k = 1$, and $m = 7$. The values for k and m are the only ones possible. $15 - 1 = 2^1 \cdot 7$
2. Choose a random integer a with $1 < a < n$.
   (a) Set $a = 4$, as it falls under $1 < a < 15$.
3. Set $b = a^m \pmod{n}$. If $b \equiv \pm 1 \pmod{n}$ output true and terminate.
   (a) $b = 4^7 \pmod{15}$. $b \equiv 4 \pmod{15}$. We move onto the next step.
4. If $b^{2^r} \equiv -1 \pmod{n}$ for any r with $1 \leq r \leq k - 1$, output true and terminate. Otherwise output false.
   (a) We pick r with $1 \leq r \leq 1 - 1$. But then r must be with $1 \leq r \leq 0$, which means r does not exist, and therefore the program outputs false and 15 is definitely composite.

4.6. **Pseudocode.** The algorithm can be written in pseudocode as follows. The parameter k determines the accuracy of the test. The greater the number of iterations of the test, the more accurate the result (see 4.8).

```
let s>0 and d odd > 0 such that n - 1 = (2^s)d
repeat k times:
    a <- random(2, n - 2)
    x <- a^d mod n
    repeat s times:
        y <- x^2 mod n
        if y = 1 and x != 1 and x != n - 1 then
            return "composite"
        x <- y
    if y != 1 then
        return "composite"
return "probably prime"
```

4.7. **Complexity.** Using repeated squaring, the running time of this algorithm is $O(k \log 3n)$, where n is the number tested for primality, and k is the number of rounds performed; thus this is an efficient, polynomial-time algorithm. FFT-based multiplication (Harvey-Hoeven algorithm) can decrease the running time to $O(k \log^2 n \log \log n = \tilde{O}(k \log^2 n)$.

4.8. **Accuracy.** The error made by the Miller-Rabin test is measured by the probability that a composite number is declared to be probably prime. The more bases a that are tried, the better the accuracy of the test. Shown in (insert the thing), at most 1/4 of the bases a are strong liars for n. So if n is composite, then running the Miller-Rabin test k times would result in n being declared probably prime with a probability at most 4^{-k}.

Now while 4^{-k} is the worst case scenario, for larger values of n, the probability for a composite number to be declared probably prime is often significantly smaller than 4^{-k}. For most numbers n, the probability is bounded by 8^{-k}, as the probability gets extremely impossible as we consider larger values of n.

However, this improved error rate should not be relied on to verify primes, as there could be a carefully chosen pseudoprime in order to defeat the primality test.

5. AKS Test

The AKS test was quite revolutionary in mathematics as the first primality-proving algorithm to be able to verify the primality of any general number, have the maximum running time be bounded by a polynomial over the number of digits in the target number, deterministically distinguish whether the number is prime or composite, and is not conditional on any subsidiary unproven hypothesis.

5.1. **The Basic Idea Behind AKS.** The basic idea behind AKS is the following.

Lemma 5.1. *Suppose n is a natural number, and a an integer coprime to n. The number n is prime if and only if the relation*

$$(x+a)^n = x^n + a \quad \text{in } (\mathbb{Z}/n\mathbb{Z})[x] \tag{5.1}$$

holds

Proof. Suppose first that $n = p$ is a prime. Observe that $\binom{p}{i} = p!/(i!(p-i)!)$ is a multiple of p for all $1 \leq i \leq p-1$. Therefore, using the binomial theorem, in $(\mathbb{Z}/p\mathbb{Z})[x]$, we have

$$(x+a)^p = x^p + \sum_{i=1}^{p-1} \binom{p}{i} x^{p-i} a^i + a^p = x^p + a^p = x^p + a \tag{5.2}$$

where the last relation holds because $a^p \equiv a \pmod{p}$ for all $a \in \mathbb{Z}$ by Fermat. This proves one direction of the lemma.

Conversely, if n is not prime, then there is some $1 \leq i \leq n-1$ with $\binom{n}{i}$ not being a multiple of n. Therefore in this case the binomial theorem shows that the coefficients of x^{n-1} (or x^i) on both sides of the identity of the lemma do not match mod n. ■

5.1 can be used as a test to check whether n is prime. But this is not a very useful test because in order to check whether $(x+a)^n = x^n + a$ in $\mathbb{Z}/n\mathbb{Z}[x]$ we must compare n coefficients, and this will take at least n operations to do. The key idea behind the AKS test is instead to check whether $(x+a)^n \equiv x^n + a \mod I$, where I is the ideal $(x^r-1)(\mathbb{Z}/n\mathbb{Z}[x]$, for a suitable value of r and some (not too many) values of a. Congruence modulo I means that we are checking polynomials treating coefficients $modn$, and treating any multiples of $(x^r - 1)$ as zero - this helps because using repeated squaring we'll have to check only on the order of $r \log n$ coefficients, and if r is like a power of $logn$ we would be able to do this in polynomial time.

5.2. **The Algorithm.** The AKS test can be boiled down into 4 steps.

(1) First we check that n is not a perfect power. One can rapidly do this, because if $n = m^k$ for some $k \geq 2$, then we must have $k \leq \log_2 n$. So there are not many choices for k, and for each choice k, we can compute quickly whether n is a k-th power or not. If n is a k-th power for some $k \geq 2$, we stop and output that n is composite.
(2) Second, let us check and make sure that n has no prime factor smaller than $100(\log n)^5$. Since there are only $100(\log n)^5$ divisions to check, this too is rapid. If we do find a small prime factor, we can stop and declare n to be composite.
(3) Find the smallest integer r such that the order of $n \pmod r$ is $\geq 9(\log n)^2$. It is crucial that there is a small value of r with this property, and this is guaranteed by the following lemma.

    Lemma 5.2. *There exists* $r \leq 100(\log n)^5$ *such that the order of* $n \pmod r$ *is at least* $9(\log n)^2$.

(4) This involves checking the following key identity

$$(x + a)^n \equiv x^n + a \mod (n, x^r - 1),$$

for various values of $a \in \mathbb{Z}$. This identity means that $(x + a)^n$ (which is in $\mathbb{Z}[x]$) differs from $x^n + a$ by an element in the ideal $(n, x^r - 1)$ of $\mathbb{Z}[x]$ — or in other words, the difference $(x + a)^n - x^n - a$ can be expressed as $nf(x) + (x^r - 1)g(x)$ where f and g are in $\mathbb{Z}[x]$. This is the most important point of the AKS algorithm:

Lemma 5.3. *Let* $n \geq 10^6$ *be given, with* n *not a perfect power. Let* r *be natural number such that all prime factors of* n *are larger than* r*, and such that the order of* $n \mod r$ *is at least* $9(\log n)^2$*. Then the key identity (5.1) holds for all* $1 \leq a \leq r$ *if and only if* n *is a prime number.*

Thus, in the final step, it is enough to check for all $1 \leq a \leq r \leq 100(\log n)^5$ and if n satisfies all these identities we can declare if to be prime. In Theorem 5.3, note that if n is prime then $(x + a)^n \equiv x^n + a \mod n$ for all natural numbers a, so that (5.1) holds for all a and all r in this case. The interesting bit is the converse, that if the key identity holds for sufficiently many cases, then n must be prime.

5.3. **Proofs.**

5.3.1. *Proof of Lemma 5.2.* Suppose instead that all $r \leq R = 2\lceil 49(\log n)^5 \rceil + 1$ are such that the order of $n \mod r$ is at most $K = \lfloor 9(\log n)^2 \rfloor$. This means that each $r \leq R$ divides some $n^k - 1$ with $k \leq K$. Therefore,

$$(\text{lem of all } 1 \leq r \leq R) \text{ divides } \prod_{k=1}^{K} (n^k - 1). \tag{5.3}$$

We shall obtain a contradiction by establishing an upper bound for the right side of 5.3, and a lower bound for the left side — the goal will be to have the lower bound larger than the upper bound, which would be impossible as a larger number cannot divide a smaller one. So the right side of 5.3 is

$$\leq \prod_{k=1}^{K} n^k = \exp\left(\frac{K(K+1)}{2}\log n\right) \leq \exp(45(\log n)^5).$$

As for the left side, using the following four equations,

$$\log d_N =$$

, with d_{2M+1} denoting the lcm of the numbers up to $2M+1$,

$$d_{2M+1}$$

Therefore the left side of 5.3 is

$$\geq 4^{49(\log n)^5}. \tag{5.4}$$

Since $4 \geq e$, this lower bound is in conflict with our upper bound, and completes our proof.

5.3.2. *Proof of Lemma 5.3.* The key to proving 5.3 is that 5.1 for different values of a can be used to generate many other similar relations. If there is a composite n satisfying 5.1 for many values of a, then eventually we will obtain so many relations that in a suitable field we'll be able to cook up a polynomial with more roots than its degree, thus getting a contradiction.

Lemma 5.4. *Suppose n, r, and a are such that*

$$(x+a)^n \equiv x^n + a \mod n, x^r - 1). \tag{5.5}$$

Let p be some prime factor of n. Then the relation

$$(x+a)^m \equiv x^m + a \mod p, x^r - 1 \tag{5.6}$$

holds for all m of the form $n^i p^j$ with i and j being non-negative integers.

By assumption the relation 5.3 holds for $m = n$. By the binomial theorem, as in 5.1, the relation 5.3 also holds for $m = p - (x+a)^o \equiv x^p + a^p \equiv x^p + a \mod p$. To prove our lemma, we establish that if 5.3 holds for $m = k$ and $m = \ell$ then it also holds for $m = k\ell$.

Indeed

$$(x+a)^{k\ell}((x+a)^k)^\ell \equiv (x^k + a)^\ell \mod p, x^r - 1, \tag{5.7}$$

upon using 5.3 for $m = k$. Now 5.3 with $m = \ell$ (and replacing x by y)

$$(y+a)^\ell \equiv y^\ell + a \mod p, y^r - 1, \tag{5.8}$$

and if we take $y = x^k$ it follows that

$$(x^k + a)^\ell \equiv x^{k\ell} + a \mod p, x^{kr} - 1. \tag{5.9}$$

Since $x^r - 1$ divides $x^{kr} - 1$, we conclude that $(x^k + a)^\ell \equiv x^{k\ell} + a \mod p, x^r - 1$, which completes our proof of Lemma 5.4.

Onto our main proof, suppose $n \geq 10^6$ is not a perfect power. Suppose that n is not divisible by any prime at most r, and that the order of $n \mod r$ is $\geq 9(\log n)^2$. Suppose that 5.1 holds for all $1 \geq a \geq r$. We must now show that n is a prime. Suppose it is not, and let p be a prime factor of n such that the order of $p \mod r$ is > 1 — such a prime p exists because if all prime factors of n were $\equiv 1 \mod r$, then $n \mod r$ itself would have order 1.

Suppose $p \mod r$ has order k — thus $r|(p^k - 1)$ (and $r \nmid (p^j - 1)$ for any $j < k$), and $k > 1$ by our choice for p. We will work in a finite field $\mathbb{F}_q$ with $q = p^k$ elements. Let β be a generator of $\mathbb{F}_q^\times$, and take $\alpha = \beta^{(p^k-1)/r}$. Thus α is an element $\mathbb{F}_q^\times$ whose order is exactly r, and in particular

$$a^r = 1. \tag{5.10}$$

Consider the relations (5.1) and (5.3) from Lemma 5.4. Put

5.4. **Running Time.** So we can use Lemma 5.1 as a test to check whether n is prime. But, this is not a very useful test because in order to check whether $(x+a)^n \equiv x^n + a \pmod{I}$, we must compare n coefficients, and this will take at least n operations to do. The key idea behind the AKS test is instead to check whether

$$(x+a)^n \equiv x^n + a \pmod{I}, \tag{5.11}$$

where I is the ideal $(x^r - 1)(\mathbb{Z}/n\mathbb{Z})[x]$ for a suitable value of r and some, but not too many, values of a. Congruence modulo I means that we are checking polynomials treating coefficients mod n, and treating any multiples of $(x^r - 1)$ as zero. This helps because using repeated squaring we'll have to check only on the order of $r \log n$ coefficients, and if r is like a power of $\log n$, we would be able to do this in polynomial time, which is the desired time.

This will involve analyzing how the AKS test fits the polynomial time, but nothing of optimizing it. Also, we won't keep track of constants and we will omit terms like $(\log n)^e$. Running times for the basic operations of arithmetic, such as adding and subtracting k bit numbers, takes on the order of k steps, and multiplying/dividing a k bit number and an ℓ bit number takes on the order of $k\ell$ steps. Now let's move on to show how the AKS test is a polynomial time algorithm.

(1) Given $k \geq 2$ and n, the time it takes to compute n to the k-th power with ℓ bits takes no more than $\ell^2 k^3$ steps, since this is just multiplying a number to itself many times without the use of repeated squaring. To check if n is a k-th power, we should start working out the binary expansion of $n^{1/k}$. The k-th root will have about $(\log_2 n)/k$ bits, and to figure out each bit we will have to take the k-th power of some number and check if it is larger than n or not. Each k-th power takes about $\log n^3$ steps, and doing this for each $2 \leq k \leq \log_2 n$, we can check if n is a perfect power in about $(\log n)^4$ steps.

(2) Here we need to divide n by numbers up to about $(\log n)^5$. Each division takes about $(\log n)(\log \log n)$ steps. The $\log \log n$ comes from the number of bits in a number of size $(\log n)^5$. So in total this step takes $(\log n)^6 \log \log n$ operations, which may be bounded by $(\log n)^7$ for simplicity.

(3) For each r going up to $100(\log n)^5$, we must compute the order of $n \pmod{r}$, which we want to be large. Given r, we simply compute $n^1, n^2, \cdot, n^k$ all $\pmod{r}$, with $K = 9(\log n)^2$. We can begin by reducing $n \pmod{r}$, which takes about $(\log n)(\log r)$ operations, and then the computations of the powers $n^j \pmod{r}$ will all take $K(\log r)^3$

operations, which is at most some constant times $(\log n)^3$. Doing this for each r in our range, we can complete Step 3 and find a suitable r in at most $(\log n)^8$ operations.

(4) Here we must verify the key identity (5.1) for r values of a. For each a, by repeated squaring we must perform on the order of $\log n$ multiplications of polynomials $(\mathrm{mod}\ n, x^r - 1)$. Each such multiplication involves computing r coefficients, and each coefficient involves about r multiplications of numbers of size at most n. Therefore, each polynomial multiplication takes about $r^2(\log n)^2$ steps at most. So for each a, our identity may be checked in about $r^2(\log n)^3$ steps. And finally, ranging over all $a \leq r$, we can complete Step 4 of the algorithm in $r^3(\log n)^3$, which is at most $(\log n^1 8$. Thus the AKS test is a polynomial time algorithm.

Now in addition to this, it is important to note that there have been, since the discovery of the AKS test, new variants which greatly imporved the speed of the algorithm. After (include several theorems or similar if necessary, prob not) the new upper bound complexity was $\tilde{O}(\log(n)^7.5)$. There was also a proposal of a variant which would run in $\tilde{O}(\log(n)^3)$ by Agrawal, Kayal, and Saxena, which was then suggested to be false by Pomerance and Lenstra.

5.5. **Examples.** Let's suppose we wish to know whether 3 is prime or not. Then the test goes as follows.

Example. We set $n = 3$ and $a = 1$, as 1 is a coprime to 3.

$(x-1)^3 - (x^3 - 1) = (x^3 - 3x^2 + 3x - 1) - (x^3 - 1) = -3x^2 + 3x$

All the coefficients are divisible by 3, so 3 is prime.

As a simple visualization, if we set $a = 1$ for all values n (as 1 is coprime to all numbers), then we can simply look at Pascal's Triangle and see if the values are all multiples of the row number, excluding the 1's.

$$\begin{array}{c}
1 \\
1 \quad 1 \\
1 \quad 2 \quad 1 \\
1 \quad 3 \quad 3 \quad 1 \\
1 \quad 4 \quad 6 \quad 4 \quad 1 \\
1 \quad 5 \quad 10 \quad 10 \quad 5 \quad 1 \\
1 \quad 6 \quad 15 \quad 20 \quad 15 \quad 6 \quad 1 \\
1 \quad 7 \quad 21 \quad 35 \quad 35 \quad 21 \quad 7 \quad 1
\end{array}$$

References

[Ivo16] James Ivory. *Demonstration of a theorem respecting prime numbers*, volume 1, page 6–8. Palapa Press, 2016.

[Leh27] Derrick Henry Lehmer. *Tests for primality by the converse of Fermat's theorem.* Project euclid, 1927.

[Sou23] Kannan Soundararajan. *Finite Fields, with applications to combinatorics.* American Mathematical Society, 2023.

[Ste09] William Stein. *Elementary Number Theory: Primes, Congruences, and Secrets.* Springer, 2009.

[Wol96] George Woltman, 1996.

[Leh27] [Sou23, Chapter 7] [Ste09, Chapter 2] [Ivo16] [Wol96]