# Survey of Primality Testing Methods

Nicholas Pun

## Contents



## 1 Introduction

In this expository project, we describe several primality testing algorithms and delve a bit into the major theorems that motivated their creation. The majority of the focus will be on probable prime tests. These tests will be very efficient, but are only capable of confirming composite-ness or telling us that a number is "probably a prime". This will be Section 2. In Section 3, we will spend some time describing the AKS primality test. This algorithm is deterministic, polynomial in time and only returns correct answers. Finally, in Section 4, we'll end with Maple implementations of two of the probable prime tests to exhibit their shortcomings, but also how we can combat their limitations.

## 2 Probable Prime Testing

The main idea behind probable prime tests is to take a theorem that holds true for all primes:

*If $p$ prime, then $S$ is true*

and verify $S$ for our given input $n$. If $S$ is false, we can safely conclude that our given number is composite. On the other hand, if $S$ is true, then $n$ is probably a prime, but could also be a composite number that happens to satisfy $S$. For example, consider the statement:

*If $p$ prime (and $p$ is not 2), then $p$ odd*

Even numbers (except 2) all clearly do not satisfy the conclusion, so we know they are composite. However, 15 is odd an number, but clearly not prime!

In this section, we will highlight more versatile conditions that lead to relatively fewer false positives and the efficient primality tests we can build from them.

## 2.1 Fermat Test

From here on out, we assume that any input we are given will be an odd number, since testing for even-ness is easy.

Our first test is based on Fermat's little theorem:

**Theorem 1** (Fermat's Little Theorem [1])**.** *If $p$ is a prime number and $a$ is any number not divisible by $p$, then*

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

Now, suppose we are given some number $n$. Attempting to check that $a^{n-1} \equiv 1 \pmod{n}$ for all $a \in \mathbb{Z}_n$ is no better than performing trial division against the integers from 2 to $\sqrt{n}$. As such, our strategy here will be to choose a uniformly random integer $a \in [2, n-2]$ and compute $b := a^{n-1} \pmod{n}$. If $b \neq 1$, then we safely conclude that $n$ is composite, and otherwise, we conclude that $n$ is "probably prime". This gives us the Fermat test:

**Algorithm 1:** Fermat Test

**Input:** Odd number $n \in \mathbb{N}_{\geq 5}$
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 Select $a \in [2, n-2]$ uniformly at random
2 $b \leftarrow a^{n-1} \pmod{n}$
3 **if** $b \neq 1$ **then return** "Composite"
**else return** "Probably Prime"

**Remark.** We choose $a \in [2, n-2]$ since $a = 1$ and $a = n-1$ satisfy Equation (1) for any odd $n$

The algorithm correctly returns "probably prime" whenever $n$ is prime (By Theorem 1). When $n$ is composite and we happen to choose $a$ with $\gcd(a, n) \neq 1$, then $\gcd(b, n) \neq 1$ as well, so the algorithm also correctly returns "composite".

So, we'll proceed with the assumptions that $\gcd(a, n) = 1$ and $n$ composite. Define $Z_n^\times = \{a \bmod n \mid \gcd(a, n) = 1\}$ (the multiplicative group of units of $\mathbb{Z}_n$) and $L_n = \{u \in \mathbb{Z}_n^\times \mid u^{n-1} \equiv 1 \pmod{n}\}$.

Suppose there exists $a \in \mathbb{Z}_n^\times \setminus L_n$. So, $\gcd(a, n) = 1$ but $a^{n-1} \not\equiv 1 \pmod{n}$, then our algorithm correctly returns "composite". We call such $a$ a **Fermat witness**. On the other hand, if $a \in L_n$, then $\gcd(a, n) = 1$ and $a^{n-1} \equiv 1 \pmod{n}$. So, this $a$ has tricked our algorithm into returning "probably prime" and we call it a **Fermat liar**

**Example 1.** Suppose $n = 15$. If $a$ is randomly chosen to be 4, then $4^{14} \equiv 1 \pmod{15}$. So, 4 is a Fermat Liar. On the other hand, if $a$ is randomly chosen to be 7, then $7^{14} \equiv 4 \pmod{15}$, so 7 is a Fermat witness

**Lemma 1.** *When they exist, there are as many Fermat witnesses as there are Fermat liars*

Since, when there exists a Fermat witness then, $L_n \leq Z_n^\times$ (as groups), so $|L_n| \leq \frac{1}{2}|Z_n^\times|$, by Lagrange's theorem.

When $L_n = \mathbb{Z}_n^\times$, i.e. there are no Fermat witnesses, we call $n$ a **Carmichael number** [2]. These numbers will always trick our algorithm unless we happen to pick a factor of $n$. In Section 2.2, we will discover a method to return accurate results despite their existence.

Excluding Carmichael numbers for now, by Lemma 1, our algorithm will correctly return "composite" with probability at least $\frac{1}{2}$. In general, if we run the algorithm $k$ times and it returns "composite" in any of the $k$ iterations, we can conclude that $n$ is indeed composite. Then, we also achieve a success probability of at least $1 - \left(\frac{1}{2}\right)^k$.

**Theorem 2** ([3, p. 520]). *If $n$ is prime, then Fermat's test returns "possibly prime" with probability 1. If $n$ is composite and not a Carmichael number, then Fermat's test returns "composite" with probability at least $1 - \left(\frac{1}{2}\right)^k$, where $k$ is the number of repeated iterations of the algorithm*

Finally, we'll analyze the time complexity of Fermat's test. We'll assume that the random decision in Line 1 takes some negligible polynomial time. The modular exponentiation in Line 2 can be computed using repeated squaring [3, p. 75-76] in time at most $\mathcal{O}(\log n \cdot M(\log n))$, where $M$ is the multiplication time for integers. (see [3, Defn. 8.26]).

**Theorem 3** ([3, p. 520]). *The algorithm has time complexity $\mathcal{O}(k \cdot \log n \cdot M(\log n))$ (where $k$ is as above)*

### 2.2 Miller-Rabin Test

We saw that Fermat's test does not perform well on inputs that are Carmichael numbers. Although the Carmichael numbers may seem like an edge case, there are (unfortunately) infinitely many Carmichael numbers. Further, they might not be as rare as we think.

**Theorem 4** ([4]). *There are infinitely many Carmichael numbers. Further, for all sufficiently large $x$, there are more than $x^{2/7}$ Carmichael numbers up to $x$.*

Despite this, the next theorem will help us improve the accuracy of Fermat's test by imposing a stronger test condition.

**Theorem 5** ([5, Theorem 3.5.1, p. 135]). *Suppose that $n$ is an odd prime and $n - 1 = 2^s t$, where $t$ is odd. If $a$ is not divisible by $n$ then,*

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

The following algorithm was discovered by Rabin [6] in 1980 as a probabilistic extension of Miller's [7] deterministic test in 1975.

**Algorithm 2:** Miller-Rabin Test

**Input:** Odd number $n \in \mathbb{N}_{\geq 5}$
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 Write $n = 1 + 2^s t$ with $t$ odd (we can repeatedly divide $n-1$ by 2 until we reach an odd $t$)
2 Select $a \in [2, n-2]$ uniformly at random
3 $b \leftarrow a^t \pmod{n}$
4 **if** $b = 1$ *or* $b = n-1$ (*i.e* $b \equiv -1 \pmod{n}$) **then return** "Probably Prime"
5 **for** $i$ *from* 1 *to* $s-1$ **do**
6 $\quad b \leftarrow b^2 \pmod{n}$
7 $\quad$ **if** $b = n-1$ **then return** "Probably Prime"
**return** "Composite"

Whenever $n$ is prime, the algorithm will return "probably prime" on Line 4. Whenever $n$ is composite, much like with the Fermat test, we want to determine a bound on how often the algorithm be tricked into returning "probably prime". Let $n$ be a composite number. We call $a$ a **witness** (or **strong witness**) to the compositeness of $n$ if $a$ violates the conditions in Equation (2) (and the algorithm is lead to correctly return "composite"). Then,

**Theorem 6** ([6, Theorem 1])**.** *For every composite $n > 4$, there are at least $\frac{3}{4}(n-1)$ witnesses to the compositeness of $n$*

Theorem 6 implies that, even including the Carmichael numbers, our algorithm will return "composite" with success probability at least $\frac{3}{4}$. Further, we can improve this probability by running $k$ iterations of the Miller-Rabin test. If the algorithm returns "composite" in any of the $k$ iterations, then our input must be composite. To that end, we can attain a success probability of at least $1 - \left(\frac{1}{4}\right)^k$.

**Theorem 7.** *If $n$ is prime, then the Miller-Rabin test returns "possibly prime" with probability 1. If $n$ is composite, then the Miller-Rabin test returns "composite" with probability at least $1 - \left(\frac{1}{4}\right)^k$, where $k$ is the number of repeated iterations of the algorithm.*

Finally, the time complexity of the Miller-Rabin test is similar to Fermat's test. The modular exponentiation on Lines 3 and 6 have time complexity at most $\mathcal{O}(\log n \cdot M(\log n))$

**Theorem 8.** *The Miller-Rabin test uses $\mathcal{O}(k \cdot \log n \cdot M(\log n))$ word operations*

### 2.3 Solovay-Strassen Test

Let $p$ be an odd prime, and $a$ an integer, the **Legendre symbol** is defined as:

$$\left(\frac{a}{p}\right) := \begin{cases} 1 & \text{if } \gcd(a,p) = 1 \text{ and there exists } x \text{ such that } a \equiv x^2 \pmod{p} \\ -1 & \text{if } \gcd(a,p) = 1 \text{ and } a \text{ is not a square mod } p \\ 0 & \text{if } \gcd(a,p) \neq 1 \end{cases} \tag{3}$$

For arbitrary odd $n$, the **Jacobi symbol** generalizes the Legendre symbol. Suppose $n = p_1^{\alpha_1} p_2^{\alpha_2} \dots p_k^{\alpha_k}$ is the prime factorization of $n$, then the Jacobi symbol is defined as:

$$\left(\frac{a}{n}\right) := \left(\frac{a}{p_1}\right)^{\alpha_1} \left(\frac{a}{p_2}\right)^{\alpha_2} \dots \left(\frac{a}{p_k}\right)^{\alpha_k} \tag{4}$$

where on the right, we are evaluating Legendre symbols.

**Theorem 9** ([8, p. 29-31])**.** *For arbitrary $a, n \in \mathbb{Z}$, the Jacobi symbol can be computed in $\mathcal{O}\left((\log a)(\log n)\right)$ time.*

Our next test is based on the following primality condition.

**Theorem 10** (Euler's Criterion [9, Theorem 83, p. 69])**.** *Let $p$ be an odd prime and $a$ an integer coprime to $p$, then*

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p} \tag{5}$$

**Algorithm 3:** Solovay-Strassen Test [10]

**Input:** Odd number $n \in \mathbb{N}_{\geq 5}$
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 Select $a \in [2, n-2]$ uniformly at random
2 **if** $\gcd(a, n) \neq 1$ **then return** "Composite"
3 $b \leftarrow \left(\frac{a}{n}\right)$
4 **if** $a^{\frac{n-1}{2}} \equiv b \pmod{n}$ **then return** "Possibly Prime"
**else return** "Composite"

Much like before, whenever $n$ is prime, the algorithm will correctly return "Possibly Prime". For composite $n$, first we define $G_n = \{u \in \mathbb{Z}_n^\times \mid u^{\frac{n-1}{2}} \equiv \left(\frac{u}{n}\right) \pmod{n}\}$. Then, $G_n \leq \mathbb{Z}_n^\times$ and so if $G_n \neq \mathbb{Z}_n^\times$, then $|G_n| \leq \frac{1}{2}|\mathbb{Z}_n^\times|$. That is, the algorithm will correctly identify that $n$ is composite with probability at least $\frac{1}{2}$.

**Theorem 11** ([10])**.** *If $n$ is prime, then the Solovay-Strassen test returns "possibly prime" with probability 1. If $n$ is composite, then the Solovay-Strassen test returns "composite" with probability at least $1 - \left(\frac{1}{2}\right)^k$, where $k$ is the number of repeated iterations of the algorithm.*

Finally, a single iteration of the Solovay-Strassen test takes $\mathcal{O}(\log n \cdot M(\log n))$ time since Theorem 9 bounds the time complexity of Line 3 and the analysis for Line 4 is exactly the same as before.

**Theorem 12** ([10])**.** *The Solovay-Strassen test has time complexity $\mathcal{O}(k \cdot \log n \cdot M(\log n))$*

## 2.4 Lucas & Frobenius Tests

Let $f(x) = x^2 - ax + b$ where $a, b$ are non-zero integers with $D = a^2 - 4b$ (called the discriminant) non-square. For $j \geq 0$, let

$$U_j = U_j(a, b) = \frac{x^j - (a-x)^j}{x - (a-x)} \pmod{f(x)} \tag{6}$$

$$V_j = V_j(a, b) = x^j + (a-x)^j \pmod{f(x)} \tag{7}$$

The sequences $(U_j)_{j\geq 0}, (V_j)_{j \geq 0}$ are called **Lucas sequences** with parameters $(a, b)$ [5, 11]. In fact, the sequences are integral and both satisfy the recurrence relation

$$U_j = aU_{j-1} - bU_{j-2},\ V_j = aV_{j-1} - bV_{j-2}$$

with initial values

$$U_0 = 0, U_1 = 1,\ V_0 = 2, V_1 = a$$

**Example 2.** One may recognize the recurrences look similar to that of the Fibonacci sequence. Indeed, we can recover the Fibonacci sequence through $(U_j(1, -1))_{j \geq 0}$.

The main testing condition for the following two tests will be the following:

**Theorem 13** ([5, Theorem 3.6.3, p. 143])**.** *Let $(U_j)$ be defined as above with parameters $(a, b)$ and $D$ non-square. If $p$ is prime and $\gcd(p, 2abD) = 1$, then*

$$U_{p-\left(\frac{D}{p}\right)} \equiv 0 \pmod{p} \tag{8}$$

*(where $\left(\frac{D}{p}\right)$ is the Legendre symbol defined in Section 2.3)*

To compute $U_j$, we'll use the following identities:

**Lemma 2** ([5]). *Let $(U_j), (V_j)$ be defined as above with parameters $(a, b)$ and $D$ non-square, then*

$$U_j = D^{-1}\left(2V_{j+1} - aV_j\right) \tag{9}$$

*Further, let $A = a^2b^{-1} - 2$, then*

$$U_{2j}(a, b) = ab^{j-1}U_j(A, 1) \tag{10}$$

Equation (9) tells us that can retrieve $(U_j)$ by computing $(V_j)$ and Equation (10) tells us that we can compute the sequence in a doubling fashion. Combining the two equations we get:

$$\begin{aligned} U_{2j}(a, b) &= ab^{j-1}(A^2 - 4)^{-1}(2V_{j+1}(A, 1) - AV_j(A, 1)) \\ &= ab^{j-1}(a^4b^{-2} - 4a^2b^{-1})^{-1}(2V_{j+1}(A, 1) - AV_j(A, 1)) \\ &= ab^{j-1}(a^2b^{-2}D)^{-1}(2V_{j+1}(A, 1) - AV_j(A, 1)) \\ &= (aD)^{-1}b^{j+1}(2V_{j+1}(A, 1) - AV_j(A, 1)) \end{aligned} \tag{11}$$

**Theorem 14** ([12]). *There exists an algorithm to compute the sequence $(V_n)$ with time complexity $\mathcal{O}(\log n \cdot M(\log n))$ using the doubling technique described above.*

Let's rephrase Theorem 13 using Equation (11):

**Theorem 15.** *Let $(U_j), (V_j)$ be defined as above with parameters $(a, b)$ and $D$ non-square. Let $A = a^2b^{-1} - 2$. If $p$ is prime and $\gcd(p, 2abD) = 1$, then*

$$2V_{\frac{1}{2}\left(p-\left(\frac{D}{p}\right)\right)+1}(A, 1) \equiv AV_{\frac{1}{2}\left(p-\left(\frac{D}{p}\right)\right)}(A, 1) \pmod{p} \tag{12}$$

This gives us the following algorithm:

**Algorithm 4:** Lucas Probable Prime Test

**Input:** Odd $n \in \mathbb{N}_{\geq 5}$, $a, b, D \in \mathbb{Z}$, where $D = a^2 - 4b$ non-square
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 $A \leftarrow a^2b^{-1} - 2$
2 **if** $2V_{\frac{1}{2}\left(n-\left(\frac{D}{n}\right)\right)+1}(A, 1) \equiv AV_{\frac{1}{2}\left(n-\left(\frac{D}{n}\right)\right)}(A, 1) \pmod{n}$ **then return** "Probably Prime"
**else return** "Composite"

The Frobenius probable prime test is an extension of the Lucas Probable Prime test but performs one more computation.

**Theorem 16** ([5, Theorem 3.6.8, p .148]). *Let $n$ be a composite number, $(U_j), (V_j)$ be Lucas sequences with parameters $(a, b)$, $D$ non-square, and $A = a^2b^{-1} - 2$. We call $n$ a **Frobenius pseudoprime** with respect to $(a, b)$ if and only if*

$$2V_{\frac{1}{2}\left(n-\left(\frac{D}{n}\right)\right)+1}(A, 1) \equiv AV_{\frac{1}{2}\left(n-\left(\frac{D}{n}\right)\right)}(A, 1) \pmod{n} \tag{13}$$

*and also*

$$b^{\frac{(n-1)}{2}}V_{\frac{1}{2}\left(n-\left(\frac{D}{n}\right)\right)}(A, 1) \equiv 2 \pmod{n} \tag{14}$$

This gives us a natural extension to the Lucas probable prime test:

**Algorithm 5:** Frobenius Probable Prime Test

**Input:** Odd $n \in \mathbb{N}_{\geq 5}$, $a, b, D \in \mathbb{Z}$, where $D = a^2 - 4b$ non-square
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 $A \leftarrow a^2 b^{-1} - 2$
2 **if** $2V_{\frac{1}{2}(n-(\frac{D}{n}))+1}(A, 1) \not\equiv AV_{\frac{1}{2}(n-(\frac{D}{n}))}(A, 1) \pmod{n}$ **then return** "Composite"
3 $B \leftarrow b^{\frac{(n-1)}{2}} \pmod{n}$
4 **if** $BV_{\frac{1}{2}(n-(\frac{D}{n}))}(A, 1) \equiv 2 \pmod{n}$ **then return** "Probably Prime"
**else return** "Composite"

**Theorem 17** ([5])**.** *The Lucas probable prime test and Frobenius probable prime tests both have time complexity* $\mathcal{O}(\log n \cdot M(\log n))$.

We note however that the Lucas probable prime test will take around twice the time compared a Fermat (or Miller-Rabin) test, since it takes $\mathcal{O}(\log n \cdot M(\log n))$ to compute each $V_{\frac{1}{2}(n-(\frac{D}{n}))+1}(A, 1)$ and $V_{\frac{1}{2}(n-(\frac{D}{n}))}(A, 1)$. In that vein, the Frobenius prime test will take three times the time, since it must also compute $b^{\frac{(n-1)}{2}}$. (say, using repeated squaring)

Let $(a, b)$ be the parameters to a Lucas sequence and $n$ a composite number, we call $n$ a **Lucas pseudoprime** with respect to parameters $(a, b)$ if it satisfies Equation (12). Baillie and Wagstaff showed that Lucas pseudoprimes are relatively rare:

**Theorem 18** ([13, Theorem 6])**.** *For fixed* $a, b$*, there is a constant* $C(x)$ *so that the number of Lucas pseudoprimes less than* $x$ *is bounded by* $x \exp(-C(x)\sqrt{\log x \log\log x})$.

In both tests above, we are given the parameters $(a, b)$. Baillie and Wagstaff [13] suggest two methods for finding $(a, b)$:

(A) Let $D$ be the first element of the sequence $5, -7, 9, -11, 13, \ldots$ (i.e. Odd numbers with alternating signs, except $-3$) for which $\left(\frac{D}{n}\right) = -1$. Let $a = 1$ and $b = \frac{1-D}{4}$

(B) Let $D$ be the first element of the sequence $5, 9, 13, 17, 21, \ldots$ for which $\left(\frac{D}{n}\right) = -1$. Let $a$ be the least odd number exceeding $\sqrt{D}$ and $b = \frac{a^2-D}{4}$

They show that on average, less than 2 values must be tried for $D$.

Finally, they recommend a mixed probable prime test algorithm that makes no mistakes for any $n < 25 \cdot 10^9$:

**Algorithm 6:** Baillie-Wagstaff Probable Prime Test

**Input:** Odd $n \in \mathbb{N}_{\geq 5}$
**Output:** Whether $n$ is "Composite" or "Probably Prime"

1 Run the Miller-Rabin test on $n$ with $a = 2$. If the test returns "Composite" then return "Composite"
2 Choose parameters $(a, b)$ using method A or B above.
3 Run $n$ on the strong version of the Lucas probable prime test and return the result. The idea is similar that of how we transformed the Fermat test into the Miller-Rabin test using Theorem 5

The individual steps of this algorithm are just subroutine calls to our previous algorithms. Hence, the time complexity of this algorithm is still $\mathcal{O}(\log n \cdot M(\log n))$.

# 3 A Deterministic Algorithm

We'll now describe a deterministic algorithm created by Agrawal, Kayal, and Saxena in 2002 [14]. The motivating theorem for this algorithm is the following:

**Theorem 19** ([14, Lemma 2.1])**.** *Let $a \in \mathbb{Z}, n \in \mathbb{N}, n \geq 2$ with $\gcd(a, n) = 1$. Then, $n$ is prime if and only if*

$$(x + a)^n \equiv x^n + a \pmod{n} \tag{15}$$

*(where $x$ is an indeterminate)*

We'll briefly justify this. First, we can write: $(x + a)^n = \sum_{k=0}^{n} \binom{n}{k} a^{n-k} x^k$. Then, when $n$ is prime, then for all $0 < k < n$, $n$ divides the numerator of $\binom{n}{k} \pmod{n}$:

$$\binom{n}{k} = \frac{n \cdot (n-1) \cdot \ldots (n-k+1)}{k \cdot (k-1) \ldots 1} \equiv 0 \pmod{n}$$

And, when $n$ is composite, let $p$ be the smallest prime factor and let $k = \frac{n}{p}$. Then,

$$\binom{n}{p} = \frac{n \cdot (n-1) \cdot \ldots (n-k+1)}{p \cdot (p-1) \ldots 1} = \frac{k \cdot (n-1) \cdot \ldots (n-k+1)}{(p-1)!} \not\equiv 0 \pmod{n}$$

This characterizes the prime numbers and one can verify the primality of any number $n$ simply by choosing $a$ and running through Equation (15). However, this will take an exponential amount of time since this would require expanding $(x + a)^n$. Instead, we will evaluate Equation (15) over the ring $\mathbb{Z}_n[x]/(x^r - 1)$ and we write this as:

$$(x + a)^n \equiv x^n + a \pmod{x^r - 1, n} \tag{16}$$

Then, the time complexity is bounded by $r$.

**Lemma 3** ([14])**.** *We can choose $r$ that is polynomial in $n$. In particular, there exist a $r \leq \max\{3, \lceil \log^5 n \rceil\}$ such that $o_r(n) > \log^2 n$*

$o_r(n)$ is the order of $n$ modulo $r$, i.e. it is smallest $k$ such that $n^k \equiv 1 \pmod{r}$. We will see that this condition will be required for the algorithm. The lemma tells us that we can perform our proposed algorithm using a polynomial amount of time by choosing $r$ appropriately.

What remains is to show that the congruence in Equation (16) implies the congruence in Equation (15). We are left in a similar situation as with the probable prime tests. While the implication certainly holds for primes, it will also hold for $n$ that is composite. However, the authors show that if we can show that Equation (16) holds for a class of numbers $a$, then the composite number must be a prime power.

**Theorem 20** ([14])**.** *Let $n$ be an odd integer and $r$ be the smallest integer such that $o_r(n) > \log^2 n$. Suppose that*

*(a) $n$ is coprime will all primes $\leq r$*

*(b)* $(x+a)^n \equiv x^n + a \pmod{x^r - 1, n}$ *for all* $1 \leq a < \sqrt{\phi(r)} \log n$. *(where* $\phi(r)$ *is Euler's totient function)*

*Then,* $n$ *is a prime power.*

This gives us the following algorithm:

**Algorithm 7:** AKS Primality Test [14]

**Input:** Integer $n > 1$
**Output:** Whether $n$ is "Composite" or "Prime"

1 **if** $n = a^b$ *for* $a \in \mathbb{N}$ *and* $b > 1$ **then return** "Composite"
2 Let $r$ be the smallest number such that $o_r(n) > \log^2 n$
3 **if** $\gcd(a, n) \neq 1$ *for any* $a \leq r$ **then return** "Composite"
4 **if** $n \leq r$ **then return** "Prime"
5 **for** $a = 1$ *to* $\left\lfloor \sqrt{\phi(r)} \log n \right\rfloor$ **do**
6 **if** $(x+a)^n \not\equiv x^n + a \pmod{x^r - 1, n}$ **then return** "Composite"
7 **return** "Prime"

**Theorem 21** ([14])**.** *The AKS Primality Test returns "Prime" if and only if* $n$ *is prime.*

Suppose $n$ is prime power, then we return "Composite" on Line 1. Line 2 selects an $r$ as in Theorem 20 and if we violate either of the assumptions in Line 3 or 6, then we return "Composite". Finally, if we make it to Line 7, then by Theorem 20, $n$ is a prime power, but since $b > 1$ in Line 1, it must be the case that $n$ is a prime.

**Theorem 22** ([14])**.** *The time complexity of the AKS Primality Test is* $\tilde{\mathcal{O}}(\log^{10.5} n)$

($\tilde{\mathcal{O}}(f(n))$ is shorthand for $\mathcal{O}(f(n) \log^k(f(n)))$ for some $k$)

For Line 1, $b < \log n + 1$, so for each $b$ we can use binary search to determine the existence of an integer $a$ so that $n = a^b$. This will give the naïve bound of $\tilde{\mathcal{O}}(\log^4 n)$. Gathen and Gerhard improves this primarily using Newton Iteration to tighten the bound to the desired $\tilde{\mathcal{O}}(\log^3 n)$ time [3, Exercise 9.44].

For Line 2, we can naïvely compute $n^k \not\equiv 1$ for $k \leq \log^2 n$ for successive values of $r$. Lemma 3 tells us that $r$ is bounded above by $\log^5 n$, so Line 2 has time complexity $\tilde{\mathcal{O}}(\log^7 n)$.

Line 3 consists of performing at most $\log^5 n$ gcd computations. This has time complexity $\tilde{\mathcal{O}}(\log^7 n)$.

Finally, Lines 6 has time complexity $\tilde{\mathcal{O}}(r \log^2 n)$ and the computation is performed $\sqrt{\phi(r)} \log n$ times. In total, this is $\tilde{\mathcal{O}}(r\sqrt{\phi(r)} \log^3 n) = \tilde{\mathcal{O}}(r^{\frac{3}{2}} \log^3 n) = \tilde{\mathcal{O}}(\log^{10.5} n)$, which dominates the overall time complexity.

# 4 Maple Implementations

In this section, we implement the Fermat test and Miller-Rabin test using Maple. In particular, we'll perform some experiments with the implementations to illustrate the limitations of the tests, how we can overcome them, and how well they stack up against the built-in Maple `isprime` function.

First we present the code:

**Listing 1: Fermat Test**

```
# Returns true is n is "probably prime" and false otherwise
FermatTest := proc(n)
  local a, b;
  a := rand(2..(n - 2))();
  b := a &^ (n-1) mod n;
  return evalb(b = 1);
end:
```

**Listing 2: Miller Rabin Test**

```
# Returns true if n is "probably prime" and false otherwise
MillerRabinTest := proc(n)
  local nn, s ,t, a, b, i;

  # Write n = 1 + 2^s * t where t odd
  nn := n - 1;
  s := 0;
  while (nn :: even) do s := s + 1; nn := nn / 2; end:
  t := nn;

  # Select random a \in [2, n-2]
  a := rand(2..(n-2))();

  # Check the first condition in Equation (2)
  b := a &^ t mod n;
  if (b = 1) or (b = n-1) then return true end if;

  # Check the second condition in Equation (2)
  for i from 1 to s-1 do
    b := b &^ 2 mod n;
    if (b = n - 1) then return true end if;
  end:

  return false;
end:
```

Recall that the motivation for improving the Fermat test was due to the existence of Carmichael numbers. Our next two experiments will be focused on the results of running the Fermat and Miller-Rabin tests on only the Carmichael numbers. We use the following theorem to generate Carmichael numbers.

**Theorem 23** ([15])**.** *For $k \geq 1$, the number $(6k+1)(12k+1)(18k+1)$ is a Carmichael number if all 3 factors are prime.*

For our first experiment, we want to see how many times the Fermat test fails to recognize a Carmichael numbers as composite as compared to the Miller-Rabin test. The following code generates 10000 Carmichael numbers and runs them against the Fermat and Miller-Rabin tests.

**Listing 3: Experiment 1**

```
counter := 0: # counts the number of times FermatTest fails to recognize a composite number
counter2 := 0: # counts the number of times MillerRabinTest fails to recognize a composite number
k := 1;
for i from 1 to 10000 do
  # Generate the next Carmichael number according to the theorem
  while (not (isprime(6*k + 1) and isprime(12*k + 1) and isprime(18*k + 1))) do
    k := k + 1;
  end:
  n := (6*k+1)*(12*k+1)*(18*k+1);

  # update counters as necessary
  if FermatTest(n) then counter := counter + 1; end if;
  if MillerRabinTest(n) then counter2 := counter2 + 1; end if;

  # increment to generate next Carmichael number
  k := k + 1;
end:

# Print results
lprint(counter, counter2);
```

We find that the Fermat test fails all 10000 times, while the Miller-Rabin test fails only 994 times. This matches our expectations that the Fermat test performs badly on Carmichael numbers. Further, this illustrates that the Miller-Rabin test is indeed an improvement on the Fermat test.

For our second experiment, we recall that we can improve the probability of success by simply running each test on the same input multiple times. As such, we will run each test 10 times on the same input. If the test identifies the input as composite any of the 10 times, then we report that the number is composite.

**Listing 4: Experiment 2**

```
# ... initialize counter, counter2 and k as before ...
for i from 1 to 10000 do
  # ... Generate Carmichael numbers as before ...
  fermat := true;
  millerRabin := true;
  # Perform each test 10 times
  for j from 1 to 10 do
    fermat := fermat and FermatTest(n);
    millerRabin := millerRabin and MillerRabinTest(n);
  end;

  # update counters as necessary
  if fermat then counter := counter + 1; end if;
```

```
14   if millerRabin then counter2 := counter2 + 1; end if;
15
16   k := k + 1; # increment to generate next Carmichael number
17 end:
```

Now, we find that the Fermat test still fails 9999 times, while the Miller-Rabin test succeeds in recognizing all 10000 numbers as composite! This matches our expectations in Theorem 7.

Finally, for the first $10^6$ odd numbers, we find that the Miller-Rabin test (run 10 times) is as good as Maple's built-in `isprime` function.

**Listing 5: Experiment 3**

```
counter := 0;
for n from 5 to 10^6 by 2 do
  millerRabin := true;
  # run Miller-Rabin 10 times on the input n
  for i from 1 to 10 do
    millerRabin := millerRabin and MillerRabinTest(n);
  end:

  # update the counter if the Miller-Rabin test produces a different result
    ↪ from isprime()
  if millerRabin <> isprime(n) then counter := counter + 1; end if;
end:
```

## References


[1] Moses Liskov. *Fermat's Little Theorem*, pages 456–456. Springer US, Boston, MA, 2011.

[2] R. D. Carmichael. Note on a new number theory function. *Bulletin of the American Mathematical Society*, 16(5):232–239, Feb 1910.

[3] Joachim von zur Gathen and Gerhard Jürgen. *Modern Computer Algebra*. Cambridge University Press, 3 edition, 2013.

[4] W. R. Alford, Andrew Granville, and Carl Pomerance. There are infinitely many carmichael numbers. *Annals of Mathematics*, 139(3):703–722, 1994.

[5] Richard E. Crandall and Carl Pomerance. *Prime Numbers: A Computational Perspective*. Springer, 2010.

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

[7] Gary L. Miller. Riemann's hypothesis and tests for primality. In *Proceedings of the Seventh Annual ACM Symposium on Theory of Computing*, STOC '75, page 234–239, New York, NY, USA, 1975. Association for Computing Machinery.

[8] Henri Cohen. *A course in computational algebraic number theory*. Graduate Texts in Mathematics. Springer, 3rd, corr. print edition, 1993.

[9] E M Wright G H Hardy. *Introduction To The Theory Of Numbers, An.* Oxford University Press, 1985.

[10] R. Solovay and V. Strassen. A fast monte-carlo test for primality. *SIAM Journal on Computing*, 6(1):84–85, 1977.

[11] Paulo Ribenboim. *My Numbers, My Friends: Popular Lectures on Number Theory.* Springer, 1 edition, 2000.

[12] Daniel Bleichenbacher. *Efficiency and Security of Cryptosystems Based on Number Theory.* PhD thesis, Swiss Federal Institute of Technology in Zurich, 1996.

[13] Robert Baillie and Samuel S. Wagstaff. Lucas pseudoprimes. *Mathematics of Computation*, 35(152):1391–1417, 1980.

[14] Manindra Agrawal, Neeraj Kayal, and Nitin Saxena. Primes is in p. *Annals of Mathematics*, 160, 09 2002.

[15] Jack Chernick. On fermat's simple theorem. *Bull. Amer. Math. Soc.*, 45(4):269–274, 04 1939.