EXPLANATION AND PERFORMANCE EVALUATION OF PRIME TESTS USED TO FIND VERY LARGE PRIME NUMBERS

Egemen ÇAMÖZÜ[a], Kerem Düz[b],Melih BOSTANCIERİ[c],
Furkan TOSUN[d],Yunus Emre BALCI[e], Enes Furkan ULUDAĞ[f]

[a]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye
[b]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye
[c]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye
[d]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye
[e]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye
[f]Computer Science and Engineering,Akdeniz University, Antalya,Türkiye

*Corresponding author: Egemen ÇAMÖZÜ, E.Ç., ÇAMÖZÜ; 20220808076@ogr.akdeniz.edu.tr

12.06.2024

Abstract

This study investigates the performance and efficiency of various primality testing algorithms for very large prime numbers in the literature, focusing on their effective operational ranges. To achieve this, the selected primality tests are explained in detail, and their algorithms are implemented in Java. A set of 15 prime numbers of varying sizes is tested using the Miller-Rabin, AKS, Fermat, and Solovay-Strassen methods in the same computational environment. The effectiveness of these algorithms is evaluated by comparing their performance limits and theoretical time complexities (Big-O notation). Experimental results reveal that although the AKS test is the most effective for numbers with up to 6 digits, it requires approximately 11 hours to test a 10-digit number, 2^{31} -1 .In contrast, the Miller-Rabin algorithm demonstrates the highest efficiency, testing a 664-digit number 2^{2203} -1 in just 0.0603874 seconds. These findings provide valuable insights for selecting the most suitable primality testing algorithm based on specific computational requirements.

Keywords: primality testing ; deterministic primality testing ; probabilistic primality testing ; efficient algorithms for large numbers ; Miller-Rabin

1 Introduction

Prime numbers were first mentioned in a Rhind papyrus about 3550 years ago. Euclid, in his famous 13-volume work The Elements, showed that there are infinitely many prime numbers, and since then, prime numbers have always been a curiosity.For this reason, mathematicians have developed various methods to find prime numbers. [1]

The first and most basic of these was created in 200 BC by the Greek mathematician Eratosthenes, who created an algorithm to calculate prime numbers using a method known as Eratosthenes' sieve. This method allowed us to find prime numbers when operating on small numbers, but it became extremely inefficient and expensive to implement when the size of the number increased, because it worked in exponential time. For instance, McGregor Dorsey explained in that for an input number of 20 digits, at a rate of discovering one prime number per second, it would take more than 14 years to determine the primality of the number. [2]
This algorithm works as follows:

1. Place the numbers from 2 to n^2 in an n×n grid.
2. Colour in the multiples of the primes up to n, excluding the primes themselves:
3. The remaining numbers that are not coloured at all will all be prime numbers:

Figure 1: Finding primes from 1 to 100 using the Sieve of Eratosthenes.

By the 17th century, famous mathematicians such as Fermat, Euler and Gauss were working to better understand prime numbers. However, the point where prime numbers gain more importance corresponds to the emergence of computer science and cryptology. Today, security is a priority in internet applications such as banking applications, e-commerce applications. In order to ensure security, the data must be encrypted. Encryption consists of two stages cryptography (encryption) and cryptanalysis (decryption). [3]

Asymmetric encryption algorithms such as RSA (Rivest-Shamir-Adleman) exploit the unique mathematical properties of prime numbers for security. In particular, the RSA algorithm relies on the mathematical difficulty of separating very large prime factors. Its basic principle is that it is easy to multiply two large prime numbers, but computationally extremely difficult to invert the product (factorisation).This impossibility prevents passwords from being cracked and is the basis of many Internet protocols today. Today, with RSA, a 1024-bit key (a number of about 300 digits) is considered a sufficient encryption technique for simple applications. However, continuously generating very large primes is a challenging process requiring high computational power. [4]

This approach relies on finding very large prime numbers and scientists have developed various tests and algorithms to find these very large primes. These tests are divided into deterministic primality tests(DPT) and probabilistic primality tests(PPT). With such primality tests, it is possible to determine with certainty whether a number is prime or not. Such methods are usually based on factorisation. Deterministic primality tests are not useful when testing large numbers as they require a lot of time. At the same time, these methods are very complicated, and the probability of making an error in their application is greater than the probability of making an error in a probabilistic primality test [5] (Silverman, 1997). Some of the deterministic tests we have examined are Elliptic curve primality proving (ECPP), AKS, Lucas-Lehmer-Raisel (LLR)

Another type of primality testing, the Probabilistic Primality Testing (PPT), proves that the passing number is prime with a high probability. In PPT, an n-bit random number is first generated to produce a prime number. Then it is subjected to a primality test. The prime number that passes the test is selected and used wherever desired. To minimise the margin of error in PPT, the number of tests to be passed can be increased. In this way, it is understood whether the number is prime or not with a very small margin of error. A number can be determined to be prime with a margin of error less than 2-100 (RSA, 1998).[6] These experiments are more preferred than deterministic prime tests because they are faster than deterministic prime tests. Some of the probabilistic tests we have examined are Fermat, Slovay-Strassen, Miller-Rabin and Quadratic Frobenius primality tests(QFT).

Deterministic primality tests give precise results but are very expensive and slow in very large numbers and are therefore not often preferred. In addition, some probabilistic tests, such as miller-rabin, have been shown to behave like deterministic tests in certain ranges. Therefore, their accuracy can be significantly increased and the probability of error can be reduced to negligible levels. Therefore, probabilistic primality tests with high accuracy rates are more preferred.

2 Main Body

ECPP

Elliptic Curve Primality Proving (ECPP) originated from groundbreaking research into interactive proof systems conducted by Shafi Goldwasser and Joe Killian in 1986. Their work explored methods to create verifiable mathematical proofs, specifically in the context of primality testing. By leveraging the rich structure of elliptic curves, they proposed a novel approach to ascertain whether a number is prime. Their idea built upon existing

advancements in number theory and elliptic curve cryptography, presenting an innovative direction for primality testing.

The introduction of ECPP had a profound impact on the scientific community. Mathematicians and computer scientists were intrigued by its potential to combine theoretical elegance with practical efficiency. The method promised significant improvements over traditional primality tests, offering a reliable and fast way to generate certificates of primality for very large numbers. This development also stimulated interest in the broader applications of elliptic curves in computational mathematics and cryptography.

Recognizing the potential of Goldwasser and Killian's idea, several researchers delved into developing and refining ECPP. Among them, Arthur Oliver Lonsdale Atkin stood out as a key figure who transformed the conceptual framework into a fully-fledged algorithm. François Morain also played a significant role in optimizing and implementing the method.

Atkin's contribution was instrumental in making ECPP practical. He devised a systematic approach to select suitable elliptic curves for primality testing, ensuring their properties aligned with the theoretical requirements. By introducing a recursive structure, he allowed the algorithm to rely on smaller primes' certificates to prove the primality of larger numbers. Furthermore, Atkin refined the process of generating and validating certificates, making the results not only precise but also easily verifiable. His work translated the abstract ideas of Goldwasser and Killian into a robust algorithm that remains one of the fastest and most reliable methods for primality proving today.

The ECPP Algorithm

The Elliptic Curve Primality Proving (ECPP) algorithm operates by leveraging the properties of elliptic curves to rigorously prove the primality of a given number n. Unlike probabilistic primality tests, ECPP provides a deterministic proof, accompanied by a primality certificate. The certificate can be independently verified, ensuring the robustness and reliability of the algorithm. Now our analysis will center on the theoretical underpinnings of study The Elliptic Curve Primality Proving , specifically the theorems and definitions employed by individuals Goldwasser and Killian. [7]

Definition 11.12. Let $P = (P_x : P_y : P_z)$ be a projective point on an elliptic curve $E/\mathbb{Q}$, with $P_x, P_y, P_z \in \mathbb{Z}$, and let N be a nonzero integer. If $P_z \equiv 0 \bmod N$ then P is *zero mod* N; otherwise, P is *nonzero mod* N. If $\gcd(P_z, N) = 1$ then P is *strongly nonzero mod* N.

Note that if P is strongly nonzero mod N, then P is nonzero mod p for every prime $p|N$. When N is prime, the notions of nonzero and strongly nonzero coincide. We now state the theorem, using $\Delta(E) := -16(4A^3 + 27B^2)$ to denote the discriminant of an elliptic curve $E\colon y^2 = x^3 + Ax + B$ in short Weierstrass form.

Theorem 11.13 (Goldwasser-Kilian)**.** *Let $E/\mathbb{Q}$ be an elliptic curve, and let $M, N > 1$ be integers with $M > (N^{1/4} + 1)^2$ and $N \perp \Delta(E)$, and let $P \in E(\mathbb{Q})$. If MP is zero mod N and $(M/\ell)P$ is strongly nonzero mod N for every prime $\ell|M$ then N is prime.*

Proof. Suppose for the sake of contradiction that the hypothesis holds and N is composite. Then N has a prime divisor $p \leq \sqrt{N}$, and E has good reduction at p since $N \perp \Delta(E)$. Let M_p be the order of the reduction of P on E modulo p. The point MP is zero mod N and therefore zero mod p, so $M_p|M$; and we must have $M_p = M$, since $(M/\ell)P$ is strongly nonzero mod N and therefore nonzero mod p, for every prime $\ell|M$. Thus P has order M on the reduction of E modulo p, and by the Hasse bound, $M \leq (\sqrt{p} + 1)^2$. But we also have $M > (N^{1/4} + 1)^2 \geq (p^{1/2} + 1)^2$, which is our desired contradiction. □

In order to apply the theorem, we need to know the prime factors q of M. In particular, we need to be sure that these q are actually prime! To simplify matters, we restrict ourselves to the case that $M = q$ is prime, and introduce the notion of a *primality certificate*.

Definition 11.14. A *primality certificate* for p is a tuple of integers

$$(p, A, B, x_1, y_1, q),$$

where $P = (x_1 : y_1 : 1)$ is a point on the elliptic curve $E\colon y^2 = x^3 + Ax + B$ over $\mathbb{Q}$, the integer $p > 1$ is prime to $\Delta(E)$, and qP is zero mod p with $q > (p^{1/4} + 1)^2$.

Figure 2: Definition of Primality Certificates for Elliptic Curves

Note that $P = (x_1 : y_1 : 1)$ is strongly nonzero mod p, since its z-coordinate is 1. Theorem 11.13 implies that if there exists a primality certificate $(p, \ldots, q)$ for $N = p$ in which $M = q$ is prime, then p is prime. Thus a primality certificate $(p, \ldots, q)$ reduces the question of p's primality to the question of q's primality. Using a chain of such certificates, we can reduce to a case in which q is so small that we are happy to test its primality via trial division. This leads to the following recursive algorithm.

Algorithm 11.15 (Goldwasser-Kilian ECPP)**.** Given an odd integer p (a candidate prime), and a bound b, with $p > b > 5$, either construct a primality certificate (p, A, B, x_1, y_1, q) with $q \leq (\sqrt{p}+1)^2/2$ or prove that p is composite.

1. Pick random integers $A, x_0, y_0 \in [0, p-1]$, and set $B = y_0^2 - x_0^3 - Ax_0$.
   Repeat until $\gcd(4A^3 + 27B^2, p) = 1$, then define E: $y^2 = x^3 + Ax + B$.
2. Use Schoof's algorithm to compute the number of points m on the reduction of E modulo p, assuming that p is prime. If anything goes wrong (which it might if p is actually composite), or if $m \notin \mathcal{H}(p)$, then return **composite**.
3. Write $m = cq$, where c is b-smooth and q is b-coarse (all prime factors greater than b). If $c = 1$ or $q \leq (p^{1/4}+1)^2$, then go to step 1.
4. Perform a Miller-Rabin test on q. If it returns **false** then go to step 1.
5. Compute $P = (P_x : P_y : P_z) = c \cdot (x_0 : y_0 : 1)$ on E, working modulo p.
   If $\gcd(P_z, p) \neq 1$, go to step 1, else let $x_1 \equiv P_x/P_z \bmod p$ and $y_1 \equiv P_y/P_z \bmod p$.
6. Compute $Q = (Q_x : Q_y : Q_z) = q \cdot (x_1 : y_1 : 1)$ on E, working modulo p.
   If $Q_z \not\equiv 0 \bmod p$ then return **composite**.
7. If $q > b$, then recursively verify that q is prime using inputs q and b; otherwise, verify that q is prime by trial division. If q is found to be composite, go to step 1.
8. Output the certificate $(p, A, \tilde{B}, x_1, y_1, q)$, where $\tilde{B} \equiv B \bmod p$ is chosen so that we have $y_1^2 = x_1^3 + Ax_1 + \tilde{B}$ (over $\mathbb{Z}$ not just modulo p).

Figure 3: Definition of Algorithm

Note that step 4 is not strictly necessary, a composite q would eventually be detected in the recursive call, but it greatly reduces the probability that we will waste time in the recursive call, which speeds up the algorithm. When the input to Algorithm 11.15 is prime, it will output a sequence of certificates, one for each recursive call, that reduce the question of p's primality to that of a prime q ¡ b that has been proved prime via trial division. Taken together, the sequence of primality certificates constitute a primality proof for p. The complexity of this algorithm, and the complexity of verifying the primality proof it generates, are considered in the problem set, under the heuristic assumption that the integer m behaves like a random integer of similar size in terms of its factorization into b-smooth and b-coarse parts. Without any heuristic assumptions, Goldwasser and Kilian proved that for almost all inputs p of a given size (all but a subexponentially small fraction), the expected running time of this algorithm is polynomial in log p. Heuristically, this is believed to be true for all inputs, but we cannot prove this.

AKS

The AKS Primality Test represents a milestone in the field of number theory and computational mathematics. Developed in 2002 by three researchers from the Indian Institute of Technology Kanpur (IIT Kanpur)—Manindra Agrawal, Neeraj Kayal, and Nitin Saxena—the algorithm is named after the initials of its inventors' last names. Manindra Agrawal had been studying the relationship between polynomials and primality. Collaborating with his students, Neeraj Kayal and Nitin Saxena, he developed a novel approach to leverage polynomial equations for primality testing. Building on foundational mathematical principles such as Fermat's Little Theorem and modular arithmetic, the team formulated the AKS algorithm. The AKS algorithm works universally for any integer n, without requiring additional assumptions or randomness. It systematically verifies primality by leveraging properties of modular arithmetic and polynomial congruences. The pivotal achievement of their work was proving that primality testing could be performed in polynomial time with a deterministic approach. This resolved a significant open question in computational number theory. [8]

2.1 The AKS Algorithm

1. On input n, if n is a perfect power, output COMPOSITE.
2. Set $R = \log^5(n)$ and $A = \log^6(n)$.
3. If n has any factors $\in (1, R)$, output COMPOSITE.
4. For each $r \in [R]$
   For each $a \in [A]$
   Check that $(X + a)^n \equiv (X^n + a) \pmod{n, X^r - 1}$ by repeated squaring
5. If all identities hold, output PRIME, else COMPOSITE.

Figure 4: The AKS Algorithm

Time Complexity The time complexity of the AKS algorithm is: $\mathbf{O((\log(n))^6)}$

It can be seen from Figure 4 that, log n represents the number of digits in n. While this is a polynomial-time complexity, which is groundbreaking from a theoretical perspective, the AKS algorithm is not as efficient in practice compared to other primality tests like Miller-Rabin (a probabilistic test) or the ECPP algorithm.
Key Factors in Complexity

1. Perfect Power Check: This step is efficient and runs in $O((\log(n))^3)$ time.
2. Factor Search in Range (1, R): This requires testing divisors up to R, where $R = \log(n)^5$ which adds computational overhead.
3. Polynomial Congruence Testing: This step involves verifying the congruence $(X + a)^n = X^n + a \pmod{n, X^r - 1})$, which is computationally expensive due to polynomial arithmetic operations.

Despite its polynomial-time guarantee, AKS is rarely used in practice because the constants and overhead associated with polynomial and modular arithmetic make it slower than probabilistic methods for large nnn. However, its theoretical importance lies in proving that primality testing can be performed in deterministic polynomial time.

APR,APR-CL

The APR test, introduced in 1983, is one of the deterministic algorithms for primality testing. It is named after its discoverers, Leonard Adleman, Carl Pomerance, and Robert Rumely. The test employs arithmetic operations within cyclotomic fields. In their original paper [9], the authors demonstrated that the algorithm could determine the primality of numbers with up to 213 decimal digits in approximately 10 minutes.

Subsequently, the algorithm was enhanced by Henri Cohen and Hendrik Willem Lenstra, resulting in the APR-CL algorithm, which can handle 100-digit numbers in a matter of seconds [10]. This improved primality testing method is currently employed in the PARI/GP computer algebra system.

Algorithm

Firstly, small primes (p=2,3,5,...) are used to quickly assess the likelihood that n is prime number.This is done by verifying:
N mod p =0, p $\in$2,3,5...k

Here k is typically chosen as all small primes up to $\sqrt{n}$.

Mathematically speaking, the APR test is a refined application of reciprocity laws. This development inspired a series of new approaches to primality testing, which subsequently led to algorithms achieving various degrees of simplification. All these algorithms adhere to a similar framework for selecting primes to test. Consequently, the asymptotic sub-exponential time bound remains unchallenged. However, these developments indicate that there is still significant room for improvement, even within the APR scheme for selecting testing primes. In the original APR test, the central stage is dedicated to verifying the following property [11]:

$$\text{ind}_q(r) \equiv k \cdot \Theta \bmod p, \text{ for some } k \in N,$$

Figure 5:

where r is a tested prime factor of the number n, p is an initial prime number, q is a Euclidean prime with p‖q-1,and $ind_r(r)$, is the index of r in $(Z/qZ)^*$ with respect to a chosen generator of the group.The symbol θ represents a computed number depending on p and q.

In the final stage, for every Euclidean prime q, the test must solve systems of congruences with the initial primes dividing q-1 as moduli. It further resolves systems of congruences with the Euclidean primes as moduli to determine the list of all possible divisors of n not exceeding $n^{1/2}$. In [12], it is demonstrated that the following stronger property can indeed be tested during the central stage:

(1.2) There is some $m > n^{1/2}$, and $o(n \bmod m)$ in the group $(Z/mZ)^*$ is bounded by $log^{clogloglogn}$ for some constant c, so that for every $r \mid n$, $r \equiv n^a \bmod m$ for some $a \in N$.

Figure 6:

The fact that the above statement can be tested during the central stage simplifies the final stage to basic trial division. This represents a significant factor contributing to the practical efficiency of the Cohen-Lenstra test. Moreover, it enables a more flexible selection of test primes. One of the distinctive features of this test is its replacement of higher reciprocity laws with elementary properties of Gauss sums. As a result, this test serves as a simplified primality test that does not rely on reciprocity laws.

Time Complexity APR-CL is one of the most widely used deterministic primality tests. APR-CL runs unconditionally in [13]:
$\mathbf{O}(\log n^{clogloglogn})$ where c is a small constant.

The time complexity of this algorithm is particularly noteworthy because it is technically exponential. However, the (log log log n) exponent term grows so slowly that, in practice, the algorithm is much faster than many other primality tests. Despite this, the usage of this algorithm is limited to specific software implementations.

Fermat's Primality Test

Primality testing is a critical operation in computational number theory and cryptography. Fermat's primality test is one of the simplest and most well-known methods for determining whether a number is prime. Although it is an efficient and straightforward method, the test is not completely reliable due to the existence of certain composite numbers known as Carmichael numbers.

Carmichael numbers are numbers that pass the test but are not prime, which shows the flaw in Fermat's primality test. While Fermat's primality test is quite efficient, especially when working with large numbers, it does not always yield accurate results. The theoretical foundation of this test is based on Fermat's Little Theorem, which plays an important role in determining whether a number is prime.

A Carmichael number is a composite number that satisfies Fermat's Little Theorem for all integers a that are coprime with the number. In other words, for a Carmichael number n, the equation a^{n-1}=1 (mod n) holds true for all a where gcd(a,n)=1, even though n is not prime. These numbers are sometimes called "Fermat pseudoprimes" because they can pass Fermat's primality test while being composite.

Fermat's Little Theorem: States that p is a prime number and a is an integer that is not divisible by p, then $a^{p-1} = 1 \pmod p$.In the other words, for any integer a that is not divisible by p, raising a to the power of p-1 will yield a result that is congruent to 1 when divided by p.
$a^p = a \pmod p$, $a^{p-1} = 1 \pmod p$
If we were to provide an example of the use of Carmichael numbers in this theorem:
Exemple 1: Take the number 561 as an example . This number is the first known Carmichael number. According to Fermat's Primality Test, if n is a prime number, then the equation $a^{n-1} = 1 \pmod n$ should hold for any a, where a and n are coprime. 561 = 3 x 11 x 17 (it is composite) Howewer for the Fermat Fermat Primality Test, 561 satisfies the equation $a^{560} = 1 \pmod{561}$ for any integer a that is coprime with 561. For example if we choose a = 2: $2^{560} = 1 \pmod{561}$ This shows that 561 behaves like a prime number under Fermat's test, even though it is not prime.

Fermat Primality Test Algorithm

1. Check small numbers:
   - If n ¡= 1 , return Composite.
   - If n=2 , return Prime.
   - If n is even, return Composite.
2. Repeat k times:
   - Choose random a such that 2 ¡= a ¡= n-2.
   - Compute a^{n-1} (mod n).
   - If a^{n-1} (mod n) != 1 , return Composite.
3. Return "Prime" if all iterations pass.

Algorithm Time Complexity and Accuracy The time complexity of Fermat's Primality Test is **O(k.log n)**, where k is the number of iterations and log n comes from the modular exponentiation step. As for accuracy, Fermat's test can produce false positives, particularly for Carmichael numbers, which satisfy Fermat's Little Theorem for all bases but are not prime. Increasing k improves accuracy but cannot guarantee perfect results.

Relation to Cryptography Fermat's Little Theorem is also very important in modern cryptography. Many cryptographic algorithms, like RSA encryption, use ideas from number theory, such as prime number testing. For example, the security of RSA encryption depends on the difficulty of factoring large composite numbers, and this becomes harder as large primes are used. Although the Fermat Primality Test is a quick way to check if a number is prime, it is not always accurate. This shows that better methods are needed to keep cryptographic systems secure.
In conclusion, even though Fermat's Little Theorem and the Fermat Primality Test are historically important and useful, their limits, especially because of Carmichael numbers, show that we need more advanced methods for both prime testing and cryptographic security. The continued use of these theorems and tests in cryptography shows how important number theory is for modern security.

Solovay-Strassen The Solovay-Strassen test is a probabilistic algorithm used to quickly determine if a number is prime. This test uses mathematical concepts like Euler's criterion and the Jacobi symbol to check for primes. However, because the test is probabilistic, it doesn't always give a definite answer. In other words, it can sometimes give incorrect results. Still, when used correctly, it provides a very effective and fast solution, especially for working with large numbers. This makes the Solovay-Strassen test a valuable tool when dealing with large prime numbers
Principles of the Solovay-Strassen Primality Test:
The Solovay-Strassen test is built upon mathematical concepts such as the Jacobi symbol and modular exponentiation. The fundamental principle of this test is based on the relationship between prime numbers and the Jacobi symbol. The test provides an approach to determine whether a number is prime, though it is not 100% accurate.

Legendre Symbol Definiton: The Legendre symbol is a mathematical tool used to determine whether a given integer b is a quadratic residue modulo a prime number p. For a prime number p and a positive integer b, the Legendre symbol is defined as follows:

$$\left[\frac{b}{p}\right] = \begin{cases} 0 & \text{if } b \text{ and } p \text{ are not relatively prime,} \\ 1 & \text{if } b \text{ is a quadratic residue mod } p, \\ -1 & \text{if } b \text{ is a quadratic non-residue mod } p \end{cases}$$

Figure 7:

The Jacobi symbol determines whether a number is a quadratic residue modul a given number. This symbol is mathematically defined as: **(m/n)**

This symbol indicates whether the number m is a quadratic residue modul n. If n is a quadratic residue modul n, the symbol is evaluated as 1; otherwise, it is evaluated as -1.

Quadratic Residue: A **Quadratic Residue** in modular arithmetic refers to whether a number is a square modul a given modulus. More technically, a number m is a quadratic residue modul n if the following holds:
A number m is a quadratic residue modul n if there exists an integer x such that:

$x^2 \equiv m \pmod{n}$

In other words, the square of the integer x will be congruent to m, and this congruence holds modul n.
Exemple 2: Let's take 7 for the example, $1^2 = 1 \equiv 1 \pmod 7$ $2^2 = 4 \equiv 1 \pmod 7$ $3^2 = 9 \equiv 2 \pmod 7$

As can be seen in this Exemple2 1, 4, and 2 are quadratic residues modul 7 because these numbers can be obtained as squares of some integers.

Euler's Criterion Euler's criterion determines whether a number is a quadratic residue modulo another number. For a number p that is prime, the following must hold [20]:

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

a is the coprime to n.
a/p is the jacobi symbol

Modular Exponentiation It refers to the computation of a number (base) raised to a large exponent under a modulus (divisor). This method is used to obtain correct results quickly and efficiently, especially when working with large exponents.

$$x^n = \begin{cases} 1 & n = 0 \\ x^{n/2} \cdot x^{n/2} & n \text{ is even} \\ x^{n-1} \cdot x & n \text{ is odd} \end{cases}$$

Figure 8: How Modular Exponentation works

However, performing calculations directly with large exponents is very time-consuming and computationally inefficient. For instance, an operation like $a^{500} \pmod n$ is impractical if you multiply a by itself 500 times. Instead, fast modular exponentiation is used. [21]

Solovay-Strassen Algorithm :

choose a random integer a such that $1 \leq a \leq n-1$
$x \leftarrow \left(\frac{a}{n}\right)$
if $x = 0$
  then return ("n is composite")
$y \leftarrow a^{(n-1)/2} \pmod n$
if $x \equiv y \pmod n$
  then return ("n is prime")
  else return ("n is composite")

Figure 9: Algorithm of Solovay-Stressen Test

An n¿2 to be tested for primality, k is the number of iterations.
Repeat k times:
Randomly select an integer a such that 1 ¡ a ¡ n-1.
Compute the Jacobi symbol J = (a / n).
Compute y = a $^(n-1)/2$ (mod n)
If J *neq* y mod n then return composite.
If the test passes all k iterations, n is declared probably prime. [22]

Algorithm Complexity and Accuracy The Solovay-Strassen test involves operations such as modular exponentiation and calculating the Jacobi symbol. These operations have a time complexity of $\mathbf{O(k.log^3\ n)}$. Modular exponentiation has a time complexity of $\mathbf{O(log^2\ n)}$. Since the loop repeats these operations for k iterations, the total complexity in the worst case can be expressed as $\mathbf{O(k.log^3\ n)}$. However, in practice, if the Jacobi symbol computation is considered to have a complexity of $\mathbf{O(log^2\ n))}$, the test typically operates with a complexity of $\mathbf{O(k.log^2\ n)}$. [24]

Miller-Rabin Test

The Miller-Rabin test is a powerful probabilistic method for determining whether a number is prime. Based on Fermat's little theorem and Euler's congruence, this test can detect a composite number with great accuracy. Here is a comprehensive review of the theory, application and impact of this test.

The Miller-Rabin test combines theoretical insights into an efficient algorithmic framework. It leverages modular exponentiation and repeated squaring to test primality properties without directly factoring the number. The algorithm employs the decomposition of n-1 into powers of two, allowing iterative checks on modular congruences. [28]

How Algorithm Works The Miller-Rabin test works by checking whether a number n behaves as a prime under modular arithmetic. For an odd integer n¿1 the process is as follow:

The Miller-Rabin test works by checking whether a number n behaves as a prime under modular arithmetic. For an odd integer n¿1 the process is as follow:

1. Decompose: n-1: Write n-1=2^e . k, where k is odd. This factorization separates the even and odd parts of n-1.
2. Select a Random Base a: Choose an integer a such that 2≤a≤n-2
3. Perform Modular Tests
   - Compute a^k (mod n). If $a^k \equiv 1$ (mod n) , n passes for this base.
   - Compute successive squarings $a^{2i} + a^k$ (mod n) for i ∈ 0, e-1. If any $a^{2i} + a^k \equiv$ -1 mod n, n passes for this base.
4. Conclude: If none of these conditions hold, n is composite. Otherwise, n is "probably prime". [28]

Definitions and Theorems Used in the Algorithm

To understand how the Miller-Rabin test operates, it's essential to explore the core theoretical foundations, particularly the concepts of witnesses and non-witnesses. These ideas form the basis for distinguishing between prime and composite numbers.

Key Definitions

1. **Witness:** A base that demonstrates the compositeness of a number n by failing one or more conditions of the Miller-Rabin test. If is a witness, n is guarenteed to be composite.
2. **Non-Witness:** A base α for which the test passes all conditions, leading n to be classified as "probably prime." Non-Witnesses include true primes and strong pseudoprimes to the base α.For composite n, non-witnesses represent false positives.[28]
3. **Strong Pseudoprime:** A composite number n that passes the Miller-Rabin test for a specific base α. Strong pseudoprimes behave prime-like under certain modular arithmetic properties.[29]

Theorems and Insights

1. **Fermat's Little Theorem:** If n is a prime number and α is co-prime to n, then $a^{(n-1)} \equiv 1$ mod n. For composite n, this congruence often fails. The Miller-Rabin test extends this principle by iteratively testing n for additional modular properties.
2. **Key Conditions in the Miller-Rabin Test:** Let n-1 = 2^e . k, where k is odd. For a number n to pass the test for a base α:
   - Either $a^k \equiv 1$ mod n, or
   - There exists some i∈[0, e-1] such that $a^{2i} + a^k$ (mod n).

   If neither condition is satisfied, α is a witness, proving n is composite. Otherwise α is a non-witness.
3. **Proportion of Witnesses (Miller-Monier Theorem):** For any composite number n, at least 75 % of the possible bases α in [2,n-2] are witnesses. This ensures that the probability of incorrectly identifying n as prime decreases exponentially with the number of trials:
   Error Probability= $(1/4)^t$, where t is the number of trials. [28]

Relation Between Witnesses and Non-Witnesses For any odd n¿1:

- A base α is a witness if α fails all test conditions, proving n is composite.
- A base α is a non-witness if n passes for α. Non-witnesses can include true primes and strong pseudoprimes to α. [28]

Algorithm

MillerRabin(n)

If $n > 2$ and n is even, return **composite.**
/* Factor $n-1$ as $2^s t$ where t is odd. */
$s \leftarrow 0$
$t \leftarrow n-1$
while t is even
  $s \leftarrow s+1$
  $t \leftarrow t/2$
end /* Done. $n-1 = 2^s t$. */
Choose $x \in \{1, 2, \ldots, n-1\}$ uniformly at random.
Compute each of the numbers $x^t, x^{2t}, x^{4t}, \ldots, x^{2^s t} = x^{n-1} \bmod n$.
If $x^{n-1} \not\equiv 1 \pmod{n}$, return **composite.**
for $i = 1, 2, \ldots, s$
  If $x^{2^i t} \equiv 1 \pmod{n}$ and $x^{2^{i-1} t} \not\equiv \pm 1 \pmod{n}$, return **composite.**
end /* Done checking for fake square roots. */
Return **probably prime.**

Figure 10: Algorithm of Miller-Rabin

Time Complexity

By using the binary expansion of an exponent t and repeated squaring you can compute x^t modulo n with O(log n) modulo n multiplication operations. [30]
And each modulo n multiplication and division will take $\mathbf{O(\log^2 n)}$ integer operations. So this makes $\mathbf{O(\log^3 n)}$ integer operations.
Once you have x^t modulo n, then x^{2t}, x^{4t}, $^{2^8 t}$ modulo n can be obtained by $s \leq \log_a^n$ iterations of repeated squaring modulo n.
All the other operations are of lower complexity. If you repeat k times to reduce probability of error, you get $\mathbf{O(k \log^8 n)}$.

The Miller-Rabin test is an elegant synthesis of theoretical number theory and practical algorithm design. Its ability to efficiently handle large numbers with probabilistic guarantees makes it invaluable for modern applications, particularly in cryptography. Despite its probabilistic nature, the test's reliability and rapid convergence make it one of the most trusted tools in primality testing. The Miller-Rabin algorithm is especially prominent in encryption software that requires prime number generation, such as code implementing the RSA algorithm.

Baillie-PSW

The Baillie-PSW primality test is a composite test that integrates aspects of the Fermat and Lucas tests. Designed to improve upon the weaknesses of each test individually, Baillie-PSW is a widely trusted method for determining the primality of numbers, particularly in cryptographic systems like RSA. Its reliability lies in its ability to combine these tests with independence, ensuring an extremely low likelihood of false positives. To date, no composite number has been discovered that can pass the Baillie-PSW test, even for extremely large numbers, making it a cornerstone of computational primality testing.[32]

Theoric The Baillie-PSW primality test combines two key methods:

1. **Fermat Primality Test:**This test uses Fermat's Little Theorem, which states that if n is prime, $2^{n-1} \equiv 1 \pmod{n}$. While effective at detecting many composites, some pseudoprimes, like Carmichael numbers, can still pass this test.

2. **Lucas Primality Test:** The Lucas test checks modular properties of numbers based on Lucas sequences. By choosing specific parameters P and Q, as suggested by Selfridge, the test complements the Fermat test and catches pseudoprimes missed by it. This combination makes the Baillie-PSW test highly reliable.[32]

   Selfridge specified the following parameters for the generation of the Lucas sequences: P = 1 and Q = (1 - D)/4, where D is the first integer in the sequence 5, -7, 9, -11, 13, -15, ... for which GCD(D,N)=1 and

the Jacobi symbol (D—N) = -1. Note, however, that if N is a perfect square, no such D will exist, and the search for D would continue all the way to $\pm \sqrt{N}$, at which point GCD(D,N)= $\sqrt{N}$ would expose N as composite. Consequently, the algorithm also presumes the presence of a preliminary check for perfect squares (as well as even integers and integers ¡ 3). [33]

Algorithm

1. Optionally perform trial division test to some convenient limit. In other words, try dividing n by all primes up to the limit. If some such prime divides n, then n is composite, otherwise continue.
2. Perform strong Fermat test to base 2. If n fails, then it is composite, otherwise continue.
3. Chose parameters P, Q for strong Lucas test by one of these two methods:
   - method A:
     Let D be the first element of the sequence $5, -7, 9, -11, 13, ...$ such that $\left(\frac{D}{n}\right) = -1$. Set $P = 1$ and $Q = \frac{1-D}{4}$.
   - method B:
     Let D be the first element of the sequence $5, 9, 13, 17, 21, ...$ such that $\left(\frac{D}{n}\right) = -1$. Let P be the least odd number exceeding $\sqrt{D}$ and let $Q = \frac{P^2-D}{4}$.
4. Perform strong Lucas test with parameters P, Q. If n fails, then it is composite, otherwise it is almost certainly a prime.

Figure 11: Algorithm of Blaise-PSW test

In step 3 Method A is used most often, because it is simpler and seems to result in less Lucas pseudoprimes than method B. They found no Baillie-PSW pseudoprimes up to 0^8 showed that there are no Baillie-PSW pseudoprimes up to 2^{64}, which means that for n 2^{64} the test is deterministic. [32]

Time Complexity BPSW requires $\mathbf{O((\log n)^3)}$ bit operations, as do Fermat's test and the Miller-Rabin algorithm. However, a BPSW test typically requires roughly three to seven times as many bit operations as a single Miller-Rabin test. The strong version of BPSW differs only in replacing the standard Lucas-Selfridge test with the strong Lucas-Selfridge test. The strong Lucas-Selfridge test produces only roughly 30 % as many pseudoprimes as the standard version; for example, among the odd composites N ¡ 10^6, there are 219 standard Lucas-Selfridge pseudoprimes, 58 strong Lucas-Selfridge pseudoprimes, and 46 base-2 strong pseudoprimes (these totals presume no screening with odd trial divisors). Since the strong Lucas-Selfridge test incurs roughly 10 % more running time, the strong tests appear to be more effective.[33]

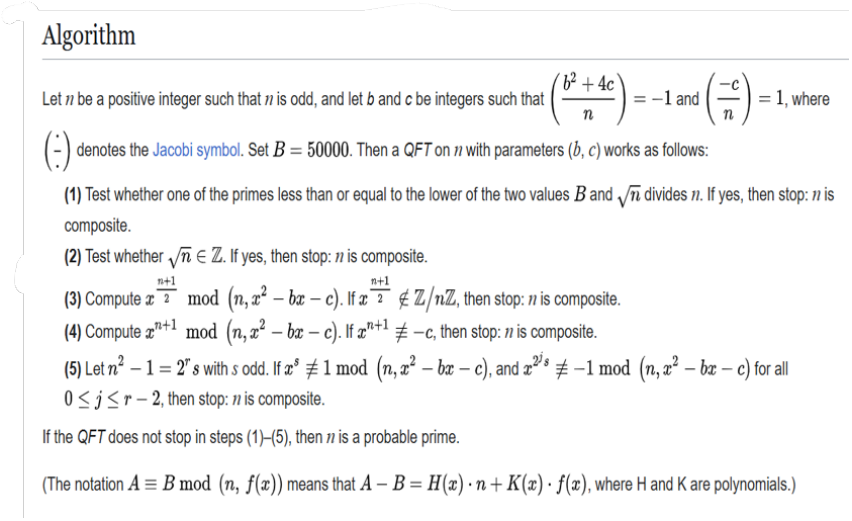
The Baillie-PSW primality test is a robust and reliable method for determining whether a number is prime, combining the strengths of the Fermat and Lucas tests. Its unique approach minimizes the risk of false positives, and no composite number has yet been found to pass it. With deterministic accuracy up to 2^{64}, it is a frequently used tool in cryptography and computational number theory because it offers both efficiency and precision.

Quadratic Frobenius Test

The Quadratic Frobenius Test (QFT), introduced by Grantham, extended this framework, incorporating advanced algebraic properties to achieve better error rates. The need for lightweight and enhanced tests later led to the development of Simplified QFT (SQFT) and Extended QFT (EQFT). [25] The QFT is built on the principles of modular arithmetic and quadratic extensions. It works by:

1. Constructing a quadratic extension ring R(n,c) = Z[x]/(n,x^2-c).
2. Verifying algebraic properties that must hold if n is prime.
3. Using probabilistic checks to reduce the likelihood of incorrectly identifying a composite as a prime. [25]

Algorithm



Algorithm

Let n be a positive integer such that n is odd, and let b and c be integers such that $\left(\frac{b^2+4c}{n}\right) = -1$ and $\left(\frac{-c}{n}\right) = 1$, where $\left(\frac{\cdot}{\cdot}\right)$ denotes the Jacobi symbol. Set $B = 50000$. Then a *QFT* on n with parameters (b, c) works as follows:

(1) Test whether one of the primes less than or equal to the lower of the two values B and $\sqrt{n}$ divides n. If yes, then stop: n is composite.

(2) Test whether $\sqrt{n} \in \mathbb{Z}$. If yes, then stop: n is composite.

(3) Compute $x^{\frac{n+1}{2}} \mod (n, x^2 - bx - c)$. If $x^{\frac{n+1}{2}} \notin \mathbb{Z}/n\mathbb{Z}$, then stop: n is composite.

(4) Compute $x^{n+1} \mod (n, x^2 - bx - c)$. If $x^{n+1} \not\equiv -c$, then stop: n is composite.

(5) Let $n^2 - 1 = 2^r s$ with s odd. If $x^s \not\equiv 1 \mod (n, x^2 - bx - c)$, and $x^{2^j s} \not\equiv -1 \mod (n, x^2 - bx - c)$ for all $0 \le j \le r - 2$, then stop: n is composite.

If the *QFT* does not stop in steps (1)–(5), then n is a probable prime.

(The notation $A \equiv B \mod (n, f(x))$ means that $A - B = H(x) \cdot n + K(x) \cdot f(x)$, where H and K are polynomials.)

Figure 12: Algorithm of QFT

Simplified QFT (SQFT): To optimize QFT for constrained environments like smart cards, Martin Seysen introduced the SQFT. This version:

- Uses fewer algebraic checks, focusing on key Frobenius properties.
- Prepares the quadratic extension R(n,c) using information from an initial Miller-Rabin test.
- Requires computational effort equivalent to about two Miller-Rabin tests per round.

Key Features

- Efficiency: Designed for devices with limited resources.
- Error Probability: Worst-case probability of error is $O(2^{-12t})$, lower than QFT with fewer computational steps. [26]

Extended QFT (EQFT): Damgård and Frandsen extended QFT to enhance both average-case and worst-case error probabilities. The EQFT introduces:

- Root-of-Unity Tests: Checks for 3rd and 4th roots of unity within the quadratic extension.
- Precomputations: Ensures the quadratic ring R(n,c) satisfies stricter subgroup properties.[25]

Algorithm Variants

EQFTac: Optimized for average-case performance, with expected runtime of about two Miller-Rabin tests per round.

EQFTwc: Optimized for worst-case performance, achieving an error probability of $O(576^{-2t})$.

Theoretical Background

1. Galois Fields and Quadratic Extensions
   - The ring R(n,c) = Z[x]/(n,x^2-c) is a quadratic extension of .
   - If n is prime and x^2-c is irreducible, R(n,c) ≅ GF (n^2), where the multiplicative group GF(n^2)* is cylic of order n^2 -1.
2. Frobenius Automorphism
   - For prime n, the Frobenius automorphism z↦z^2 holds in GF(n^2). The SQFT verifies this property by testing z^n =(the conjugate of z).
3. Cyclotomic Polynomials
   - The tests ensure z satisfies ϕq (z)=0 for q-th roots of unity.

   The tests ensure z satisfies (z)=0 for q-th roots of unity.
4. Norm Function and Conjugation

Norm: N(z)=b^2-ca^2, where z = ax + b ∈ R (n,c).

Conjugate: $\overline{Z}$ =b-ax, a key check in the Frobenius property. If n is prime and x^2-c is irreducible, R(n,c) ≅ GF (n^2), where the multiplicative group GF(n^2)* is cylic of order n^2 -1.

5. Error Probability

QFT: $O(19.8^{-t})$.
   - For t rounds of SQFT, the worst-case error probability is 2^{-12t}.
   - For EQFTwc, the error probability reduces further to $256/570^2$t

The Quadratic Frobenius Test (QFT) and its variants represent significant advancements in primality testing, balancing efficiency, and reliability. While QFT serves as the foundation, SQFT optimizes for resource-constrained devices, and EQFT provides robust guarantees for cryptographic applications. Together, these tests offer tailored solutions for diverse computational environments.

3 Experiments and Discussion

In order to evaluate the computational efficiency of 4 primality tests, 1 deterministic (AKS) and 3 probabilistic (Miller-Rabin, Solovay-Stressen, Fermat), a set of prime numbers proven to be prime of various sizes was tested. For consistency, each measurement was made on the same computer in nanoseconds. It was measured and recorded in nanoseconds. The results of these tests and measurements are visualized in Figure 13, which shows the computational time according to the size of the tested prime numbers.

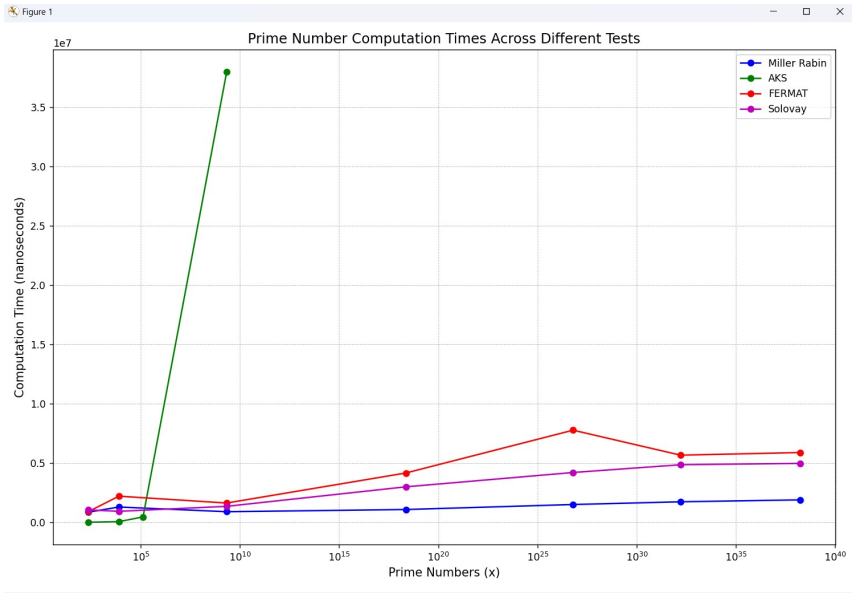


Figure 13: Chart of nanosecond-primeNumber(x)

As shown in Figure 13, it is seen how long the AKS test took to test even the 4th smallest element of the sample, 2147483647 (2^{31} -1) with 10 digits. This time is approximately 11 hours. This shows us once again how costly deterministic tests can be. This makes it computationally impractical for larger prime numbers. In addition, this behavior is consistent with its theoretical complexity, which is polynomial but has a large fixed overhead, as explained on page 5. The steep increase in the graph shows that the AKS algorithm, while accurate and precise, is impractical for most real-world applications that require speed.Although Fermat's probabilistic prime number test performs relatively well for smaller prime numbers, its computational time increases with larger inputs, making it clearly more costly than both Solovay-Strassen and Miller Rabin in terms of computational overhead.

The Solovay-Strassen test showed a balanced performance for all input sizes. Although not as fast as Miller Rabin, it is more reliable than Fermat. Its moderate computational efficiency and probabilistic nature may make it a suitable choice for certain applications.

The Miller Rabin test consistently performed the best among the tested algorithms, especially for larger prime numbers. As a probabilistic algorithm, it achieves significant speed advantages at the expense of a small error probability, which can be reduced by increasing the number of rounds. Therefore, as a result of this experiment, it is clearly seen that the most suitable test to use and perform is the Miller Rabin test.

Codes and sample for our experiments can be found https://github.com/egemencamozu/deneme.git

4 Conclusion

In this paper, we examined the primality tests used in finding very large prime numbers and proving their primality. For this examination, we first explained the definitions, historical processes, algorithms and time complexities of certain deterministic and probabilistic tests theoretically. Then, we analyzed the computation

times of Miller-Rabin, AKS, Fermat and Solovay-Strassen primality tests for selected prime numbers of different sizes. According to these results, we observed that Miller-Rabin test is the most efficient and scalable algorithm, making it ideal for cryptographic applications. On the other hand, we observed that AKS, although deterministic and theoretically groundbreaking and robust, is almost impossible to calculate for larger prime numbers due to its high time complexity. Although Fermat and Solovay-Strassen offer alternative probabilistic approaches, their reliability and performance are variable and the fact that Carmichael numbers can mislead the Fermat test also reduces the possibility of using the Fermat test.

In conclusion, the choice of a prime number test depends on the balance between speed, accuracy, and the specific requirements of the application. Future research should focus on optimizing deterministic algorithms such as AKS and exploring hybrid approaches to further increase the practicality and reliability of prime number testing in modern computing systems, and attempting to improve performance by leveraging hardware features such as quantum computers are also areas of focus.

5 References

[1.] $Üzücek, Melike. Alparslan, Eda. Nas, .\backslash Asal Sayı Nedir? Asal Sayılar Neden Bu Kadar Önemli?."$ $Edited by Eda Alparslan. Ev$

[2.] $Duta, Cristina-Loredana, Laura Gheorghe, and Nicolae Tapus."$ $Framework for Evaluation and Comparison of Primality T$

[3.] $Ganti, Ishaan, and Ethan Hutt."$ $Comparing and Reviewing Modern Primality Tests."$ $Journal of Student Research 11, no. 3(20$

[4.] $https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/07/sifreleme-yöntemleri$

[5.] $SILVERMAN, R.D., Fast Generation of Random, Strong RSA Primes, RSA Laboratories' CryptoBytes Magazine, 1997.$

[6.] $RSA, RSAFAQv4, Frequently Asked Questions About Today's Cryptography–What's Primality Testing?, 1998.$

[7.] $Manindra Agrawal, Neeraj Kayal, Nitin Saxena, Primes Is In P, Annals of Math. 160(2004), 781–793.$

[8.] $A. Granville, It is Easy To Determine Whether a Given Integer Is Prime, Bull. Amer. Math. Soc. (N.S.) 42$

[9.] $L.M. Adleman, C. Pomerance and R.S. Rumely, "On distinguishing prime numbers from composite numbers", Annals of Math$ 206, 1983.

[10.] $https://t5k.org/prove/prove4_1.html$

[11.] $Ming-deh, A Huang. On A Simple Primality Testing Algorithm. Department of Electrical Engineering and Computer Scienc$

[12.] $H. Cohen, H.W. Lenstra, Jr., "Primality Testing and Jacobi Sums", to appear in Math. Comput.$

[13.] $https://www.researchgate.net/publication/369685477_Comparing_and_Reviewing_Modern_Primality_Tests$

[14.] $Math Monks, Fermat's Little Theorem, https://mathmonks.com/remainder-theorem/fermats-little-theorem$

[15.] $https://cs.uno.edu/\ dbilar/10CSCI6130-$

[16.] $Zachary S. McGregor-Dorsey, Methods of Primality Testing, https://cs.uno.edu/\ dbilar/10CSCI6130-Cryptography/papers/DORSEY$

[17.] $Keith Conrad, Fermat's Test, https://kconrad.math.uconn.edu/blurbs/ugradnumthy/fermattest.pdf.$

[18.] $Fermat Method of Primality Test, https://www.geeksforgeeks.org/fermat-method-of-primality-test/, Geeksfo$

[19.] $Java Program to Implement Fermat Primality Test Algorithm, https://www.sanfoundry.com/java-program-implem$

[20.] $Euler's Criterion, https://www.chegg.com/homework-help/questions-and-answers/79-theorem-euler-s-c$

[21.] $Modular Arithmetic, https://www.geeksforgeeks.org/modular-arithmetic/, Geeks for Geeks.$

[22.] $Church - TuringThesis, https://qc - at - davis.github.io/QCC/Classical - Computation/Church - Turing - Thesis$

[23.] $SlidePlayerPresentation, https://slideplayer.com/slide/5864331/google_vignette, SlidePlayer.$

[24.] $DebdeepMukhopadhyay, TheRSACryptosystem : PrimalityTesting, https://cse.iitkgp.ac.in/\ debdeep/courses_iitkgp/C$

[25.] $Grantham, J.QuadraticFrobeniusPrimalityTests.$1998.

[26.] $Seysen, M.ASimplifiedQuadraticFrobeniusPrimalityTest.$2005.

[27.] $Damgård, I., Frandsen, G.AnExtendedQuadraticFrobeniusPrimalityTest.BRICSReportSeries,$ 2003.

[28.] $Conrad, Keith."TheSolovay–StrassenTest."UniversityofConnecticut, https://kconrad.math.uconn.edu/blurbs/gradnu$

[29.] $https://www.researchgate.net/publication/238834304_RabinMiller_Primality_Test_Composite_Numbers_Which_Pass_It$

[30.] $https://crypto.stackexchange.com/questions/95110/rabin - miller - primality - testcomplexity$

[31.] $https://www.researchgate.net/publication/369685477_Comparing_and_Reviewing_Modern_Primality_Tests$

[32.] $https://is.muni.cz/th/e1pe6/diplomka_text.pdf$

[33.] $https://faculty.lynchburg.edu/\ nicely/misc/bpsw.html$