STM JOURNALS

International Journal of
Algorithms Design and
Analysis Review

ISSN: 2584-1866
Volume 2, Issue 2, 2024
July—December
DOI (Journal): 10.37591/IJADAR

https://journals.stmjournals.com/ijadar

Research IJADAR

# Primality Testing: A Comprehensive Analysis of Methods and Time Complexity

Sheetal S. Patil[1,*], Avinash M. Pawar[2]

***Abstract***

*This paper examines various primality testing algorithms and analyzes their time complexity. The algorithms we examine include the trial division, which is straightforward but becomes inefficient with large numbers; Fermat's little theorem which is a probabilistic method included in Monte Carlo type of randomized algorithm; the Solovay–Strassen, based on properties from number theory, particularly those related to Euler's criterion and Jacobi symbols; and the Miller–Rabin Probabilistic Test, which balances efficiency and accuracy, providing probable results for very large numbers. The advantages and disadvantages of each algorithm are evaluated, considering both their theoretical efficiency and practical use in real-world scenarios. Additionally, we analyze the applicability of these algorithms in cryptography and other real-world contexts where identifying prime numbers is critical.*

**Keywords:** Primality testing, trial division, probabilistic method, randomized algorithm

## INTRODUCTION

Prime numbers serve as the fundamental building blocks of the number system and play a crucial role in diverse fields, such as cryptography, coding theory, and computational mathematics. The primality test is essential to determine whether a given number is prime. This test has applications that extend far beyond theoretical mathematics, touching upon areas such as secure communications and digital currencies [1].

This research on primality testing aims to demystify these enigmatic numbers through computational efficiency and innovative design [2].

This research paper is not just about implementing and comparing algorithms. It is about understanding the underlying principles, appreciating the beauty of mathematical logic, and recognizing the practical implications of these theories. Prime numbers may appear to be a specialized topic, but their applications are wide-ranging and diverse. The importance of prime numbers cannot be overstated by securing online transactions to efficiently compress data [3].

***Author for Correspondence**
Sheetal S. Patil
E-mail: sspatil@bvucoep.edu.in

[1]Assistant Professor, Department of Computer Engineering, Bharati Vidyapeeth Deemed University College of Engineering, Pune, Maharashtra, India
[2]Associate Professor, Department of Mechanical Engineering, Bharati Vidyapeeth's College of Engineering for Women, Pune, Maharashtra, India

Received Date: October 19, 2024
Accepted Date: October 22, 2024
Published Date: November 06, 2024

**Citation:** Sheetal S. Patil, Avinash M. Pawar. Primality Testing- A Comprehensive Analysis of Methods and Time Complexity. International Journal of Algorithms Design and Analysis Review. 2024; 2(2): 25–31p.

## PROPOSED METHODOLOGY

### Trial Division

The trial division method is one of the simplest and most straightforward methods for testing whether a number is prime. It is an entry-level algorithm widely taught in introductory programming and mathematics courses because it is easy to understand and implement [4].

 

**Algorithm**

```
Algorithm Trial Division Primality Test(n):
  if n <= 1:
       return False
    if n == 2 or n == 3:
     return True
    if n % 2 == 0:
     return False
       for i = 2 to sqrt(n):
       if n % i == 0:
     return False
     return True
```

**Fermat's Little Theorem**

Fermat's theorem is a fundamental concept in number theory, providing an intriguing approach to primality testing, although it is not entirely reliable [5].

Fermat's Little Theorem states that if $p$ is a prime number and $a$ is an integer not divisible by $p$, then $a^{(p-1)} = 1\ (mod\ p)$.

**Algorithm**

```
Algorithm Fermat Test(n, k):
   if n <= 1:
      return False
   if n == 2 or n == 3:
      return True
   if n % 2 == 0:
      return False
   for i = 1 to k do:
      a = random(2, n - 2)
      if mod_exp(a, n-1, n)!= 1:
         return False
   return True
function mod_exp(base, exp, mod):
   result = 1
   while exp > 0:
      if exp % 2 == 1:
         result = (result * base) % mod
      base = (base * base) % mod
      exp = exp / 2
   return result
```

**Miller–Rabin Test**

The Miller–Rabin test relies on the principles of modular arithmetic. It is probabilistic, meaning that it can indicate that a number is composite with certainty, but it can only suggest probable primality [6].

It involves selecting a random integer a between 2 and p-2, computing the value of b = a^(p-1) mod p, and checking whether b is not equal to 1. If b is not equal to 1, then the number p is a composite. If b is equal to 1, the strong Fermat test is performed, which involves factorizing p-1 as 2^k * m, computing x = a^m mod p, and checking whether x is equal to 1 or p-1. If x is equal to 1 or p-1, the number p is likely to be primed. Otherwise, for i = 1 to k-1, x is the squared modulo p, and if the result is equal to p-1, the number p is likely to be prime [7]. Otherwise, p is a composite value. To increase the probability of correctness, the test was repeated multiple times using different random bases. The greater the number of iterations, the lower the probability of a false positive (a composite number incorrectly identified as prime).

 

***Algorithm***

```
Algorithm MillerRabin(n, k):
    if n <= 1:
        return False
    if n == 2 or n == 3:
        return True
    if n % 2 == 0:
        return False

    s = 0
    d = n - 1
    while d % 2 == 0:
        d = d / 2
        s = s + 1

    for i = 0 to k - 1:
        a = random(2, n - 2)
        x = mod_exp(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for r = 0 to s - 1:
            x = mod_exp(x, 2, n)
            if x == n - 1:
                break
        if x!= n - 1:
            return False

    return True

function mod_exp(base, exp, mod):
    result = 1
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp = exp / 2
    return result
```

**Solovay–Strassen Test**

The Solovay–Strassen primality test is a probabilistic method that was developed by Solovay and Strassen. Solovay and Volker Strassen were involved in checking a condition derived from Euler's criterion using the Jacobi symbol. This test can reliably determine whether a number is composite or likely prime. The test never states a prime number as a composite number, similar to the Fermat and Miller–Rabin primality tests [8]. It is used in cryptography because of its very low error probability.

**Algorithm**

```
Algorithm SolovayStrassen(n, k):
    if n <= 2 or n % 2 == 0:
        return False
    for i = 1 to k do:
        a = random(2, n - 1)
        jacobi = computeJacobiSymbol(a, n)
        if jacobi = 0 or mod_exp(a, (n-1)/2, n)!= jacobi % n:
            return False
```

 

```
    return True

Algorithm computeJacobiSymbol(a, n):
    # Compute Jacobi symbol (a/n)
    # This is a placeholder function; actual implementation requires the handling of edge cases [9].

Algorithm mod_exp(base, exp, mod):
    result = 1
    while exp > 0:
        if exp % 2 == 1:
            result = (result * base) % mod
        base = (base * base) % mod
        exp = exp / 2
    return result
```

## RESULTS AND COMPLEXITY ANALYSIS

Figures 1–5 represent the results and the complexity analysis of the proposed model.

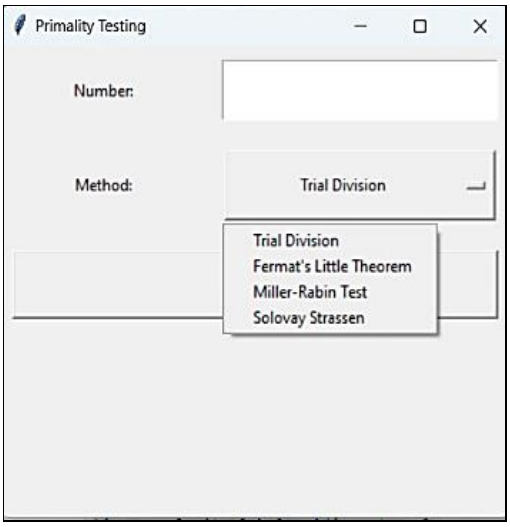


**Figure 1.** GUI for primality testing and time complexity analysis.

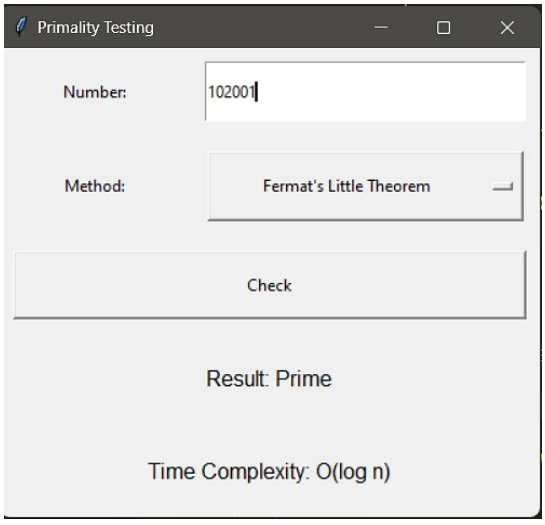


**Figure 2.** Fermat's Little Theorem.

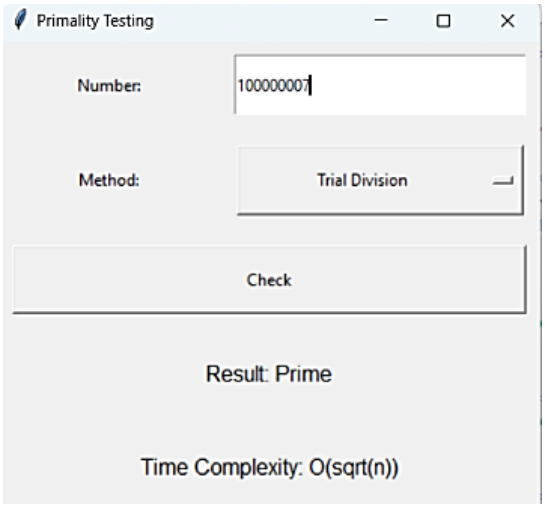


**Figure 3.** Trial division.

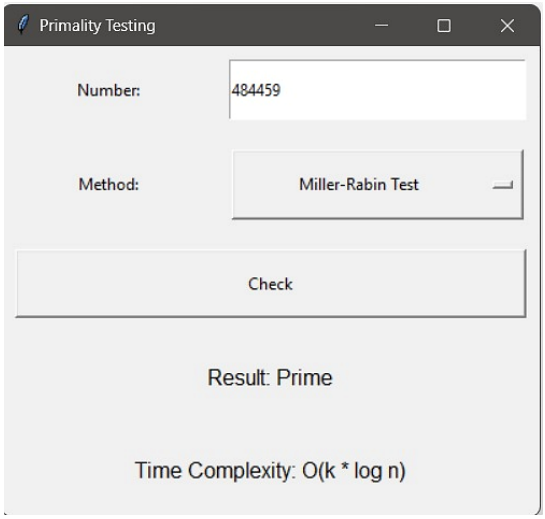


**Figure 4.** Miller–Rabin Test.

 

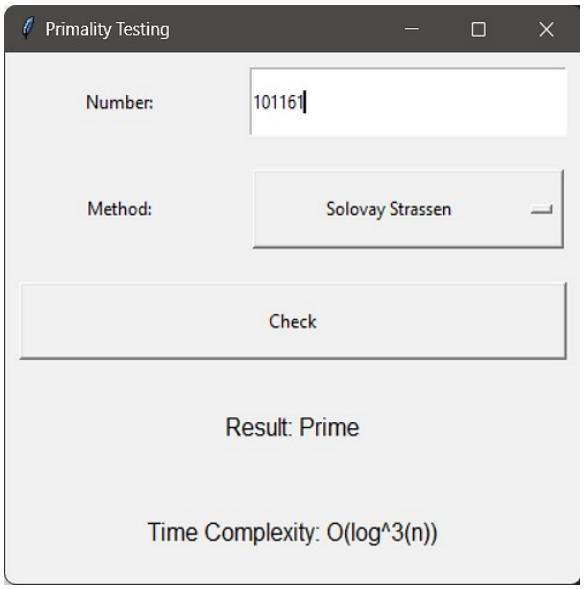


**Figure 5.** Solovay–Strassen method.

**Trial Division**

The trial division method has a time complexity of O(n). This is because to test whether a number n is prime, divisors need to be checked up to n [10].

**Fermat's Little Theorem**

*Modular exponentiation:* This can be performed in O(log n) time using the method of exponentiation by squaring. This is efficient, even for large exponents.

*Fermat's test loop:* The test is repeated k times for accuracy, and each iteration involves a single modular exponentiation. Consequently, the total time complexity was $O(k \log n)$

**Miller–Rabin Test**

*Preprocessing step* (finding s and d): This step involves repeatedly dividing n−1 by 2, which requires O(logn) time [11].

*Main loop:* Each iteration involves modular exponentiation, which can be performed in O (log n) time.

*Overall complexity:* Because the test is run k times for reliability, the total time complexity is $O(k \log n)$

**Solovay–Strassen Method**

*Modular exponentiation:* Done in O(logn) time using exponentiation by squaring.

*Jacobi symbol calculation:* Also takes $O(k \log n)$ time.

*Overall complexity:* Because the test is repeated k times for accuracy, the total time complexity is $O(k \log n)$.

**CONCLUSION**

Various primality testing algorithms offer a trade-off between simplicity, efficiency, and accuracy, making them suitable for different applications. The trial division method, while simple, has a high time complexity of O(n), making it impractical for large numbers because of the need to check divisors up

 

to n. Fermat's Little Theorem, with modular exponentiation performed in O(log n) time, improves efficiency, but its reliability depends on the number of iterations k, resulting in a total time complexity of O(k log n).

The Miller–Rabin test further improves upon Fermat's test by introducing a preprocessing step to break down the problem, which requires O (log n) time, and its main modular exponentiation loop operates in O(log n) time. The overall time complexity is O (k log n) by repeating the test kkk times to increase accuracy. Similarly, the Solovay–Strassen method employs modular exponentiation in O (log n) time, coupled with the Jacobi symbol calculation, which also operates in O (k log n), resulting in the same overall complexity.

Each of these algorithms provides a balance between computational cost and accuracy, with the Miller–Rabin Test and Solovay–Strassen method being the most widely used because of their efficiency and probabilistic guarantees. In practice, the choice of algorithm depends on the size of n, the required level of certainty, and the computational resources available. For large-scale cryptographic applications, probabilistic methods such as Miller–Rabin tend to be favored because of their significantly lower computational complexity when compared to deterministic methods such as trial division.

**Applications and Future Scope**

Primality testing, the process of determining whether a number is prime, plays a vital role in several fields and has significant potential for future advancements. In cryptography, it is important to generate large prime numbers, which are integral to encryption algorithms such as Rivest-Shamir-Adleman (RSA), which secure online transactions and communications [12, 13]. Efficient primality tests are also critical in computer science, aiding the design of algorithms, improving computational efficiency, and addressing integer factorization challenges. In the fields of blockchain and cryptocurrencies, primality testing plays a crucial role in maintaining the security and integrity of decentralized systems. In pure mathematics, it is employed to analyze the distribution of prime numbers and verify various mathematical conjectures. Additionally, prime numbers are essential in scientific computing, contributing to the precision of numerical simulations and large-scale computations [14].

The future of primality testing appears promising and expansive. Researchers are continually striving to develop faster and more efficient algorithms with ongoing efforts to improve the Azure Kubernetes Service (AKS) primality test for wider practical use. Quantum computing has the potential to transform primality testing by introducing exponentially faster methods for integer factorization. As encryption demands increase, there is an increasing need for more robust primality tests capable of handling larger prime numbers and supporting advanced encryption methods such as post-quantum cryptography [15]. The integration of artificial intelligence and machine learning can further enhance the field, allowing AI to predict prime numbers and create new testing algorithms. Primality testing is expected to find applications across diverse sectors, including finance, genetics, logistics, and AI, thereby influencing the digital and scientific domains of the future.

**REFERENCES**

1. Agrawal M, Kayal N, Saxena N. PRIMES is in P. Ann Math. 2004;781–93.
2. Kiss G. A strategy for elliptic curve primality proving. Acta Univ Sapientiae Inform. 2015;7:125–42. DOI: 10.1515/ausi-2015-0015.
3. Lenstra HW, Pomerance CB. Primality testing with Gaussian periods. J Eur Math Soc. 2019;21:1229–69. DOI: 10.4171/jems/861.
4. Shor PW. Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings 35th Annual Symposium on Foundations of Computer Science, Santa Fe, NM, USA, 1994. p. 124–34. DOI: 10.1109/SFCS.1994.365700.
5. Agrawal M. Primality tests based on Fermat's little theorem. In: International Conference on Distributed Computing and Networking; 2006 Dec 27. Springer Berlin Heidelberg; p. 288–93.

 

6. Solovay R, Strassen V. A fast Monte-Carlo test for primality. SIAM J Comput. 1977;6:84–5. DOI: 10.1137/0206006.
7. Pocklington HC. The determination of the prime or composite nature of large numbers by Fermat's theorem. Proc Camb Philos Soc. 1915;18:29–30.
8. Rabin MO. Probabilistic algorithm for testing primality. J Number Theory. 1980;12:128–38. DOI: 10.1016/0022-314X(80)90084-0.
9. Lenstra HW Jr. Elliptic curve factorisation and primality testing. Paper presented at: Computational Number Theory Conference; 1985 Aug; Arcata, California. Available from: https://www.math.leidenuniv.nl/~lenstrahw/PUBLICATIONS/1986d/art.pdf.
10. Islam MM, Hossain MS, Hasan MK, Shahjalal M, Jang YM. FPGA implementation of high-speed area-efficient processor for elliptic curve point multiplication over a prime field. IEEE Access. 2019;7:178811–26. DOI: 10.1109/ACCESS.2019.2958491.
11. Antonov N, Ishmukhametov S. An intelligent choice of witnesses in the Miller–Rabin primality test. Reinforcement learning approach. Lobachevskii J Math. 2022;43:3420–9. DOI: 10.1134/S1995080222150045.
12. Schoof R. Four primality testing algorithms. arXiv Preprint ArXiv:0801.3840 [Preprint]. 2008 Jan 24.
13. Alford WR, Granville A, Pomerance C. There are infinitely many Carmichael numbers. Ann Math. 1994;139:703–22. DOI: 10.2307/2118576.
14. Bos JW, Costello C, Longa P, Naehrig M. Selecting elliptic curves for cryptography: An efficiency and security analysis. J Cryptogr Eng. 2016;6:259–86. DOI: 10.1007/s13389-015-0097-y.
15. Rivest RL, Shamir A, Adleman L. A method for obtaining digital signatures and public-key cryptosystems. Commun ACM. 1978;21:120–6. DOI: 10.1145/359340.359342.

 