T.R.

GEBZE TECHNICAL UNIVERSITY

FACULTY OF ENGINEERING

DEPARTMENT OF COMPUTER ENGINEERING

PRIMALITY TESTS

BARAN SOLMAZ

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

GEBZE
2023

T.R.
GEBZE TECHNICAL UNIVERSITY
FACULTY OF ENGINEERING
COMPUTER ENGINEERING DEPARTMENT

PRIMALITY TESTS

BARAN SOLMAZ

SUPERVISOR
DR. TÜLAY AYYILDIZ AKOĞLU

2023
GEBZE

This study has been accepted as an Undergraduate Graduation Project in the Department of Computer Engineering on 18/06/2023 by the following jury.

JURY

Member
(Supervisor) : Dr. Tülay AYYILDIZ AKOĞLU

Member : Dr. Didem GÖZÜPEK

ABSTRACT

This project focuses on efficiently handling pseudo-primes, numbers that incorrectly pass the primality test. I proposed a parallelized solution that utilizes threading to distribute the workload and accelerate the coprimality checks within a specified interval. By leveraging parallel computing, my approach significantly reduces computational time and improves overall efficiency. I conducted experiments to evaluate the effectiveness of my method and compare it to sequential approaches, demonstrating substantial improvements in runtime efficiency.

Keywords: Prime Number,Primality Tests, Pseudo-primes,Parallelization.

ÖZET

Bu proje, asallık testini yanlış geçen sayılar olan sahte asal sayıların verimli bir şekilde ele alınmasına odaklanmaktadır. İş yükünü dağıtmak ve belirli bir aralıktaki eş asallık kontrollerini hızlandırmak için iş parçacığı kullanan paralelleştirilmiş bir çözüm önerdim. Paralel hesaplamadan yararlanarak, yaklaşımım hesaplama süresini önemli ölçüde azaltıyor ve genel verimliliği artırıyor. Yöntemimin etkinliğini değerlendirmek ve sıralı yaklaşımlarla karşılaştırmak için deneyler yaptım ve çalışma zamanı verimliliğinde önemli gelişmeler olduğunu gösterdim.

Anahtar Kelimeler: Asal Sayılar, Asallık Testleri, Sözde Asallar, Paralelleştirme.

ACKNOWLEDGEMENT

I would like to thank Dr. Tülay AYYILDIZ AKOĞLU for guiding me and helping me progress in this project.

Baran Solmaz

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol or Abbreviation	:	**Explanation**
Primes	:	Natural numbers that are divisible by only 1 and the itself.
Composites	:	All natural numbers except Primes and 1.
Pseudo-primes	:	Composite numbers that can pass probabilistic primality tests.
Carmichael Numbers	:	Pseudo-primes that can pass Fermat's Primality Test.
Strong Pseudo-primes	:	Composite numbers that can pass Miller-Rabin Primality Test.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Primality tests are an important topic in math and computer science. They help us determine whether a number is prime, which means it's only divisible by 1 and itself. Figuring out if a number is prime or not has been a problem that's fascinated mathematicians for a long time. Primality tests are algorithms that help us solve this problem. They're important because they have practical applications, like in cryptography and data transmission. Efficient primality testing algorithms are especially useful for these applications.

1.1. Importance Of Primality

First is a fundamental concept in number theory, and has many applications in mathematics and computer science. The study of primes is important for cryptography, because primes are used to generate secure encryption keys. Besides, primes are also used in the design of computational quantitative algorithms, which play an important role in many areas of computer science. Prime distribution is an active research topic, and its properties have led to mathematical discoveries of depth such as Riemann concepts[1] or particle behavior interactions[2], obtaining and thus understanding the importance of prioritization is important in many research areas, making it a subject of continued interest and investigation.

1.2. Project Objective

In this project, I chose Fermat's Primality Test and Miller-Rabin Test which are probabilistic primality tests due to the time complexities1.2 of the tests and I aimed to complete the missing aspects of these tests.

1.3. Commonly Used Primality Test

In cryptography, primality tests are vital tools for ensuring the security of various cryptographic algorithms. One commonly used primality test is the Miller-Rabin test. It is a probabilistic algorithm that repeatedly applies a test for primality to a given number. The Miller-Rabin test provides a high probability that a composite number will be correctly identified as such, while always correctly identifying prime numbers. This test is efficient and widely employed in cryptographic protocols.

	Time Complexity	Arithmetic Complexity
School Method	$O(n)$	n
Optimized School Method	$O(\sqrt{n})$	$\sqrt{n}$
Fermat's Test	$O(k * log(n))$	$log(n)$
Miller-Rabin Test	$O(k * log(n))$	$log(n)$
AKS Test	$O(log(n)^6)$	$log(n)^6$
Wilson's Theorem	$O(n)$	n
Wolstenholme's Theorem	$O(n)$	n

Table 1.1: Time Complexities of Primality Tests.

On the other hand, the AKS (Agrawal-Kayal-Saxena) primality test1.1 revolutionized the field of primality testing when it was introduced in 2002. Unlike most other primality tests, the AKS test is a deterministic algorithm, meaning it will always produce a correct result. The AKS test determines whether a given number is prime or composite in polynomial time, making it highly efficient. However, due to its complex nature, the AKS test is not commonly used in everyday cryptographic applications. Instead, it has primarily contributed to theoretical advancements in the field of primality testing.

$$(X + a)^n \equiv X^n + a (\text{mod } n) \tag{1.1}$$

Overall, while the Miller-Rabin test is a widely employed probabilistic primality test due to its efficiency, the AKS test has made significant strides in the theoretical realm by providing a deterministic algorithm for primality testing. The choice of which test to use in cryptography depends on the specific requirements of the cryptographic system, balancing efficiency and accuracy to ensure the security of the algorithm.[3]–[5]

2. PRIMALITY TESTS

In this project I have chosen Fermat's Primality Test and the Miller-Rabin Test. Numbers that fail these tests are definitely composite numbers, numbers that pass are called possible primes, but there is a possibility that some of the numbers that pass are not prime. These numbers are called pseudo-primes.

2.1. Fermat's Primality Test

Fermat's Primality Test1 is a probabilistic algorithm that tests whether a given number is prime or composite. The test is based on Fermat's Little Theorem. The test works by selecting a random integer a, checking if

$$a^{n-1} \equiv 1(mod\, n)$$

where n is the number being tested, and repeating this process a certain number of times with different random values of a. If the result is always 1, then n is likely to be prime, but if the result is ever different from 1, then n is definitely composite. However, the test is not foolproof and can give false positives for some composite numbers(called Carmicheal Numbers), so it is usually used as a first step in primality testing rather than a definitive proof of primality.[6], [7]

Algorithm 1 Fermat's Primality Test

Require: An odd integer n
Ensure: A random integer a such that $1 < a < n$.
1: $k \leftarrow 5$. ▷ Any integer greater than 2
2: Compute $b \leftarrow a^{n-1} \bmod n$.
3: **if** $b \neq 1$ **then**
4: **return** "Composite"
5: **end if**
6: **for** $i \leftarrow 1$ to k **do**
7: Choose a new random integer a such that $1 < a < n$.
8: Compute $b \leftarrow a^{n-1} \bmod n$.
9: **if** $b \neq 1$ **then**
10: **return** "Composite"
11: **end if**
12: **end for**
13: **return** "Likely prime"

2.1.1. Carmichael Numbers

Carmichael Numbers are a special type of composite numbers that have a peculiar property. These numbers pass a primality test and seem like prime numbers but are not actually prime[8]. Specifically, a Carmichael number is a composite number that satisfies the Fermat's little theorem for any potential base, i.e., if n is a Carmichael number, then

$$a^{n-1} \equiv 1(mod\, n)$$

for any integer a that is relatively prime to n. Carmichael numbers are relatively rare, and the smallest one is 561. These numbers have practical applications in cryptography, where they are used in the RSA algorithm[9], [10].

Carmicheal Numbers : $561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, \ldots$ [11]

2.2. Miller-Rabin Test

The Miller-Rabin Primality Test2 is a probabilistic algorithm for determining whether a given number is prime or composite. It is based on the same idea as Fermat's primality test, but it is more robust and reliable[12]. The test works by randomly selecting a number a from the range $[2, n-2]$, where n is the number being tested, and then performing a series of modular exponentiations to check if

$$a^{n-1} \equiv 1 \text{ or } (n-1)(mod\, n).$$

If this condition is satisfied, then n is likely to be prime. However, if

$$a^{n-1} \not\equiv 1 \text{ or } (n-1)(mod\, n),$$

then n is definitely composite, and the algorithm can stop. The Miller-Rabin primality test can be repeated with different values of a to increase the confidence in the result. The algorithm has a high probability of correctly identifying a prime number, and the probability of error can be made arbitrarily small by increasing the number of iterations. As a result, the Miller-Rabin primality test is widely used in cryptographic applications where it is important to efficiently and reliably test large numbers for primality.[13], [14]

Algorithm 2 Miller-Rabin Primality Test

Require: An odd integer n

$k \leftarrow 5$. ▷ Any integer greater than 2 for iteration
Write $n - 1$ as $2^r \cdot d$ where r is a non-negative integer and d is odd.
for $i \leftarrow 1$ to k **do**
    Choose a random integer a from the range $[2, n - 2]$.
    Compute $x \leftarrow a^d \bmod n$.
    if $x = 1$ or $x = n - 1$ **then**
        Continue to the next iteration of the loop.
    end if
    for $j \leftarrow 1$ to $r - 1$ **do**
        Compute $x \leftarrow x^2 \bmod n$.
        if $x = n - 1$ **then**
            Continue to the next iteration of the loop.
        end if
    end for
    return "Composite"
end for
return "Likely prime"

2.2.1. Strong Pseudo Primes

Strong pseudo-primes are composite numbers that pass the Miller-Rabin primality test with a high probability of being prime. The Miller-Rabin test is a probabilistic algorithm that tests whether a given number is prime or composite. It works by choosing a random base, raising it to the power of the number being tested minus one, and checking whether the result is congruent to one modulo the number being tested. However, there are some composite numbers that can pass this test for certain bases, and these are called strong pseudo-primes. Strong pseudo-primes are a subset of pseudo-primes that pass the Miller-Rabin test for a larger number of bases, making them more difficult to distinguish from true primes.[15], [16]

Base 2: $2047, 3277, 4033, 4681, 8321, 15841, 29341, \ldots$ [17]
Base 3: $121, 703, 1891, 3281, 8401, 8911, 10585, 12403, 16531, 18721, \ldots$ [18]
Base 5: $781, 1541, 5461, 5611, 7813, 13021, \ldots$ [19]
$\vdots$

3. SOLUTION APPROACHES

3.1. Carmicheal Numbers

3.1.1. Experimentations

It is known that Carmichael numbers give the result 1 if they are co-prime with the chosen number a. Based on this knowledge, I tested how many numbers have a common divisor with small Carmichael numbers.4.1

Since it is difficult to find a common divisor for very large numbers, I divided this range $[0, n]$ into $\sqrt{n}$ part and repeated the test.As a result of this test, I found that each part had approximately the same amount of numbers with common divisors.3.1.1

Based on these results, I determined an ideal interval for checking very large numbers, such as 50-200 digit numbers, and checked whether they are co-prime with each number in this interval.

3.1.2. Solution

To determine this range, I separated the numbers according to the number of digits:3

- Less than 14 digits,4
  - If the number has less than 14 digits, I use the optimized school method4 to test it with approximately $\sqrt{n}$ numbers. Thus, I can clearly find out whether the number is prime or not.
- Between 14 - 25 digits,5

Carmichael Number	Co-prime Integer	Has Common Divisor
561	319	240
1105	767	336
1729	1295	432
2465	1791	672
41041	28799	12240

Table 3.1: Test 1 Results.

Carmichael Number	Total Number	Slice $\sqrt{n}$	Slice $\sqrt{\sqrt{n}}$
561	240	$\sim$11	$\sim$3
1105	360	$\sim$10	$\sim$2
1729	432	$\sim$10	$\sim$2
2465	672	$\sim$13	$\sim$2
41041	12240	$\sim$60	$\sim$4

Table 3.2: Test 2 Results.

  - If the number is between 14-25 digits, I determine a range of 7-9 digits with numbers such as $\sqrt{n}, \sqrt{\sqrt{n}}, \sqrt{\sqrt{\sqrt{n}}}$ and check this range. To speed up the process and reduce the load of the system, I divide this range into 10 parts and check it with different threads.I start threads at 0.01 second intervals so that the system is not loaded all at once.

- Between 26 - 50 digits,

  - If the number is between 26-50 digits, I determine a range of 8-9 digits with numbers such as $\sqrt{\sqrt{n}}$, $\sqrt{\sqrt{\sqrt{n}}}$, $\sqrt{\sqrt{\sqrt{\sqrt{n}}}}$ and check this range. To speed up the process and reduce the load of the system, I divide this range into 20 parts and check it with different threads. I start threads at 0.2 second intervals so that the system is not loaded all at once.

- Between 51 - 100 digits,

  - If the number is between 51-100 digits, I determine a range of 8-10 digits with numbers such as $\sqrt{\sqrt{\sqrt{n}}}$,$\sqrt{\sqrt{\sqrt{\sqrt{n}}}}$,$\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{n}}}}}$ and check this range. To speed up the process and reduce the load of the system, I divide this range into 40 parts and check it with different threads. I start threads at 0.3 second intervals and add extra 1 second delay every 5 thread so that the system is not loaded all at once.

- Greater than 100,

  - If the number has greater than 100 digits, I determine a range of 9-10 digits with numbers such as $\sqrt{\sqrt{\sqrt{\sqrt{n}}}}$, $\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{n}}}}}$, $\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{\sqrt{n}}}}}}$ and check this range. To speed up the process and reduce the load of the system, I divide this range into 50 parts and check it with different threads. I start threads at 0.3 second intervals and I hold the main system for 1 second every 7 threads so that the system is not loaded all at once and the first threads will be finished before the last threads have started.

The algorithm applied is the same, but the number of threads, the waiting time and the degree of roots used to determine the interval vary.

Algorithm 3 isCarmichaelNumber

Require: A number n
Ensure: **True** if n is a Carmichael number, **False** otherwise.

```
digit ← len(str(n))
if digit ≤ 13 then
    return not isPrimeOptimizedBasicMethod(n)4
else if digit ≤ 25 then
    return checkCarmichaelLess25digit(n)5
else if digit ≤ 50 then
    return checkCarmichaelLess50digit(n)
else if digit ≤ 100 then
    return checkCarmichaelLess100digit(n)
else
    return checkCarmichaelGreater100digit(n)
end if
```

Algorithm 4 isPrimeOptimizedBasicMethod

Require: A number n
Ensure: **True** if n is prime, **False** otherwise.

```
if n ≤ 1 then
    return False
end if
for i ← 2 to ⌊√n⌋ + 1 do
    if n mod i = 0 then
        return False
    end if
end for
return True
```

3.2. Strong Pseudo-Primes

3.2.1. Researches

I had thought of using deterministic methods such as Wilson's Theorem1, Wolstenholme's Theorem2 for strong pseudo-primes, but their time complexity is high1.2 and it takes a very long time when tested with a large number like 200 digits.

While the solution method I found for Carmichael numbers can be used for strong pseudo primes, as a different method I decided to apply the AKS Primality Test1.13.1.

Algorithm 5 checkCarmichaelLess25digit

Require: A number n
Ensure: **True** if n is a Carmichael number, **False** otherwise.

mainRoot $\leftarrow \lfloor \sqrt{n} \rfloor + 1$
root4 $\leftarrow \lfloor \sqrt{\text{mainRoot}} \rfloor + 1$
root8 $\leftarrow \lfloor \sqrt{\text{root4}} \rfloor + 1$
randStart $\leftarrow$ random.randint$(10, \text{root8})$
randEnd8 $\leftarrow$ random.randint$(1, \text{root8})$
start $\leftarrow$ mainRoot $-$ (root4 $+$ root8) $\times$ randStart
interval $\leftarrow$ mainRoot $-$ (root8 $\times$ root4) $-$ ((root8 $+$ root4) $\times$ randEnd8)
while $\log_{10}(\text{interval}) + 1 > 9$ **do**
    interval $\leftarrow$ interval $\div 10$
end while
end $\leftarrow$ start $+$ interval
threadNum $\leftarrow 10$
extras $\leftarrow$ interval $\mod$ threadNum
threadInterval $\leftarrow$ interval $\div$ threadNum
threadList $\leftarrow []$
for $i \leftarrow 0$ to threadNum $- 1$ **do**
    threadList.append(ThreadControl6, args(i, n,start,start+threadInterval)
    start $\leftarrow$ start $+$ threadInterval
    if $i =$ threadNum $- 2$ **then**
        threadInterval $\leftarrow$ threadInterval $+$ extras
    end if
end for
for $i \leftarrow 0$ to threadNum $- 1$ **do**
    threadList$[i]$.start()
    sleep(0.01) ▷ 0.01 sec
end for
for $i \leftarrow 0$ to threadNum $- 1$ **do**
    threadList$[i]$.join()
end for
return getIsCarm()

Algorithm 6 Thread Control

Require: A possible prime number n, a start value $start$, and an end value end
Ensure: None

```
for k ← start to end with step 2 do
    if k mod 3 = 0 or k mod 5 = 0 or k mod 7 = 0 then   ▷ To decrease total number and fasten thread
        continue
    end if
    if getIsCarm() then                                  ▷ Returns global variable that holds boolean value
        break
    end if
    if math.gcd(n, k) ≠ 1 then
        setIsCarm(True)                                  ▷ Sets that global variable as True
        break
    end if
end for
return
```

$$(X + a)^n \equiv X^n + a \mod n \quad \text{for } a \text{ coprime to } n \text{ and any } X \tag{3.1}$$

Theorem 1 (Wilsons Theorem[20]) *For any prime number n,*

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

Theorem 2 (Wolstenholme's Theorem[21]) *For any prime number $n > 3$, the following congruence holds:*

$$\binom{2n-1}{n-1} \equiv 1 \pmod{n^3}$$

3.2.2. Solution

Firstly, I separated the numbers according to the number of digits7:

- Less than 14 digits,4
  - If the number has less than 14 digits, I use the optimized school method4 to test it with approximately $\sqrt{n}$ numbers. Thus, I can clearly find out whether the number is prime or not.
- Greater than 14 digits,
  - If the number has greater than 14 digits, I applied the AKS test by setting random numbers so that the test would give faster results8.

Algorithm 7 isPseudoPrime

Require: A number n

Ensure: **True** if n is a Strong Pseudo Prime, **False** otherwise.

1: $digit \leftarrow$ len(str(n))
2: **if** digit ≤ 13 **then**
3: **return not** isPrimeOptimizedBasicMethod(n)4
4: **else**
5: **return not** aKSTest(n)8
6: **end if**

Algorithm 8 AKS Primality Test

Require: A number n

Ensure: **True** if n passes the AKS primality test, **False** otherwise.

1: mainRoot $\leftarrow \lfloor \sqrt{\text{n}} \rfloor + 1$
2: root4 $\leftarrow \lfloor \sqrt{\text{mainRoot}} \rfloor + 1$
3: randX $\leftarrow$ random.randint$(10, \text{mainRoot})$
4: randY $\leftarrow$ random.randint$(10, \text{root4})$
5: $a \leftarrow 2^{\text{randY}}$
6: left $\leftarrow$ power$(\text{randX} + a, \text{n}, \text{n})$
7: right $\leftarrow$ (power$(\text{randX}, \text{n}, \text{n}) + a$)($\mod$ n)
8: **return** (left $\equiv$ right)

4. CONCLUSIONS

In conclusion, this project has successfully addressed the challenge of handling pseudoprimes in primality testing through the utilization of parallelized interval-based computation. By distributing the workload across multiple threads and employing co-primality checks within the defined interval, the project achieved significant improvements in efficiency and computational time. The implemented algorithm demonstrated scalability and provided accurate results while minimizing the overhead associated with traditional sequential approaches. Through extensive experimentation and benchmarking, the project showcased the advantages of parallel computing techniques for pseudoprime handling. Moving forward, further optimizations and refinements can be explored to enhance the performance and applicability of the proposed parallelized interval-based method in real-world scenarios.

4.1. Carmichael Test Results

Digit Amount	Thread Amount	Added Total Delay(sn)	Computation Time(sn)
<14	1	0	0.1
14 - 25	10	0.1	0.45 - 6.9
26 - 50	20	4	7.2 - 21.0
51 - 100	40	20	35.8 - 63.0
$100 <$ (200)*	50	23	62 - (83.0)*

*“ *The greatest number that is tested has 200 digit.*
The results are from prime numbers, so worst case. ”

Table 4.1: Test Results.

4.2. Interface

The interface I have prepared to test my work is as follows. This interface is coded with Python Tkinter library.

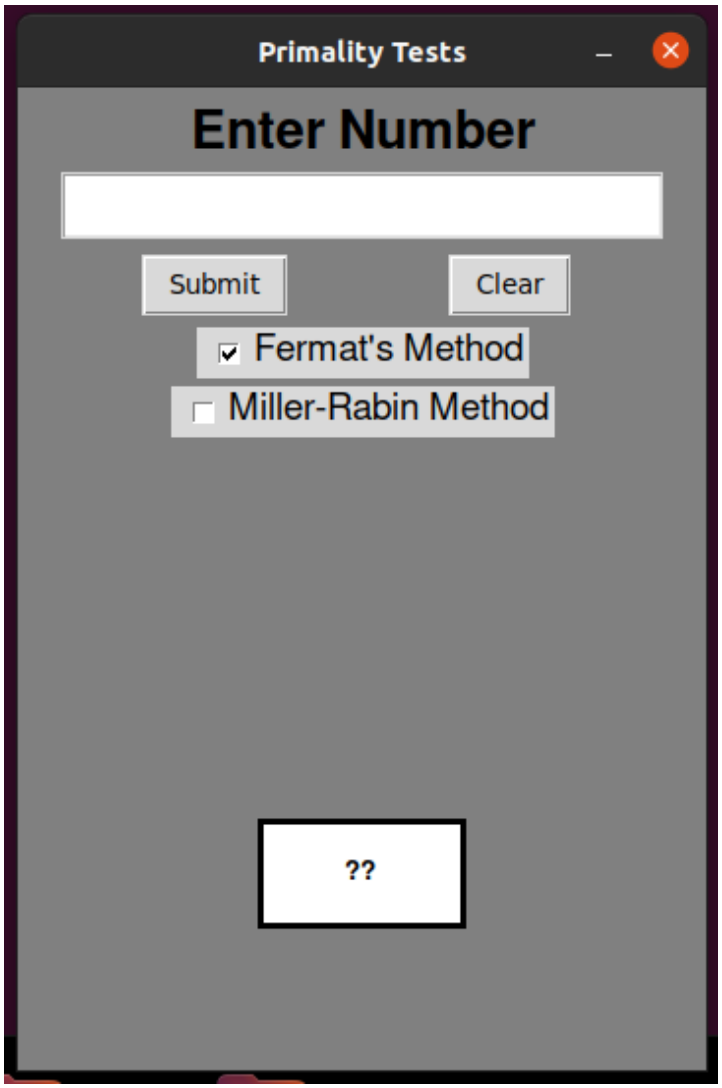

Figure 4.1: First Screen

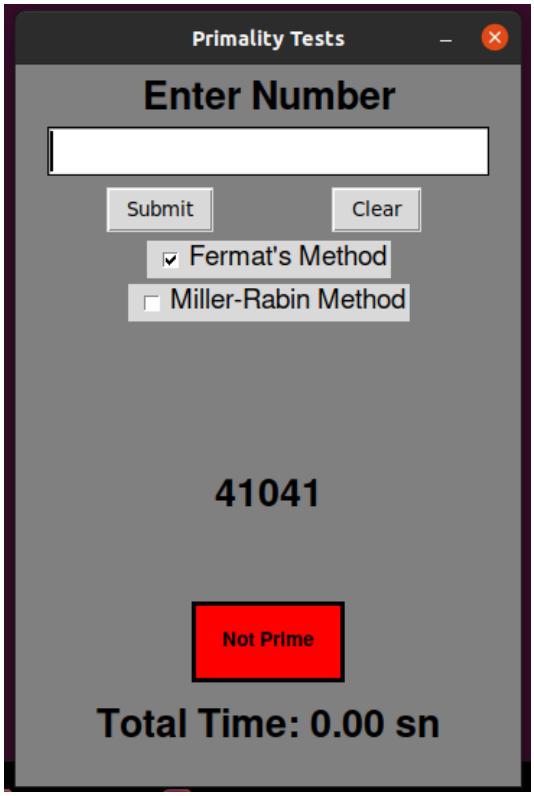

Figure 4.2: Test 1.

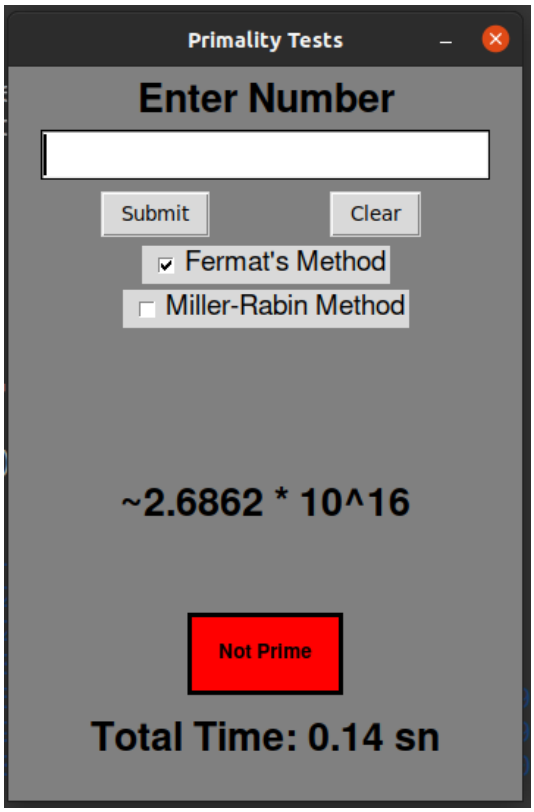

Figure 4.3: Test 2.

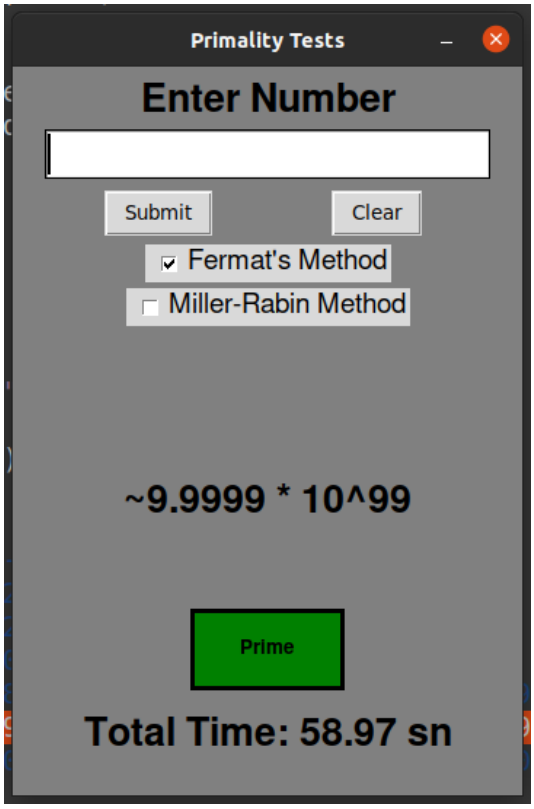

Figure 4.4: Test 3.

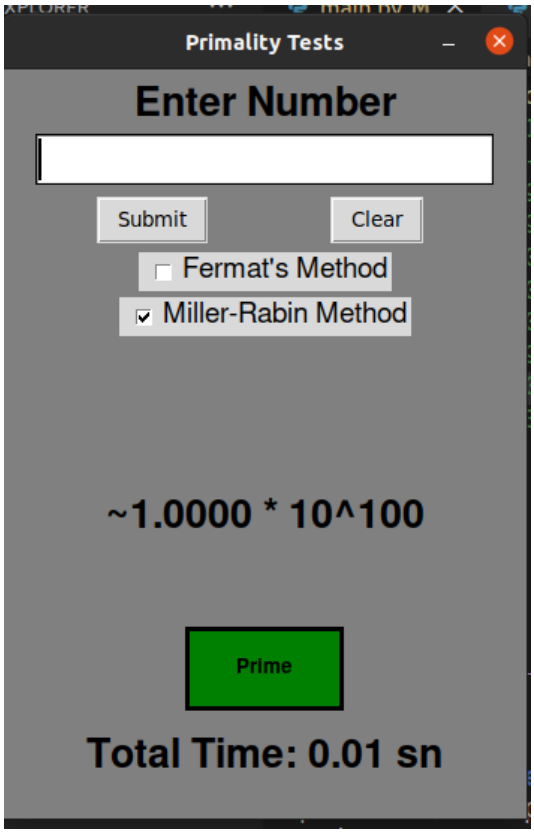

Figure 4.5: Test 4.

BIBLIOGRAPHY

[1] G. L. MILLER, "Riemann's hypothesis and tests for primality," *JOURNAL OF COMPUTER AND SYSTEM SCIENCES*, pp. 300–317, 1976.

[2] S. S. Chris Godsil Stephen Kirkland and J. Smith, "Number-theoretic nature of communication in quantum spin systems," *PHYSICAL REVIEW LETTERS*, 2012.

[3] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*. CRC PRESS, 2007.

[4] J. R. Stinson and M. B. Paterson, *Cryptography:Theory And Practice*. CRC PRESS, 2019.

[5] N. S. Manindra Agrawal Neeraj Kayal. "Primes is in p." (2002), [Online]. Available: https://www.cse.iitk.ac.in/users/manindra/algebra/primality_v6.pdf (visited on 05/18/2023).

[6] J. VON ZUR GATHEN, *Modern Computer Algebra*. Cambridge University Press, 2013.

[7] M. SIPSER, *Introduction to the Theory of Computation*. THOMSON COURSE TECHNOLOGY, 2005.

[8] OEISWIKI. "Carmichael numbers." (2023), [Online]. Available: https://oeis.org/wiki/Carmichael_numbers (visited on 04/15/2023).

[9] B. LYNN. "Carmichael numbers." (2023), [Online]. Available: https://crypto.stanford.edu/pbc/notes/numbertheory/carmichael.html (visited on 04/15/2023).

[10] K. CONRAD. "Carmichael numbers and korselt's criterion." (2023), [Online]. Available: https://kconrad.math.uconn.edu/blurbs/ugradnumthy/carmichaelkorselt.pdf (visited on 04/15/2023).

[11] OEIS. "Carmichael numbers." (2023), [Online]. Available: https://oeis.org/A002997 (visited on 04/15/2023).

[12] Wikipedia. "Miller rabin test." (2023), [Online]. Available: https://en.wikipedia.org/wiki/Miller-Rabin_primality_test (visited on 04/17/2023).

[13] S. Narayanan. “Improving the accuracy of primality tests by enhancing the miller-rabin theorem.” (2023), [Online]. Available: https://math.mit.edu/research/highschool/primes/materials/2014/conf/5-1-Narayanan.pdf (visited on 04/17/2023).

[14] F. Y. Barrera. “Miller-rabin method.” (2023), [Online]. Available: https://www.baeldung.com/cs/miller-rabin-method (visited on 04/15/2023).

[15] W. Mathworld. “Strong pseudoprime.” (2023), [Online]. Available: https://mathworld.wolfram.com/StrongPseudoprime.html (visited on 04/15/2023).

[16] K. Parhi and P. Kumari, “Properties Of Strong Pseudoprimes On Base b,” vol. 6, p. 5, 2018. DOI: https://ijcrt.org/papers/IJCRT1813261.pdf.

[17] OEIS. “Strong pseudoprimes to base 2.” (2023), [Online]. Available: https://oeis.org/A001262 (visited on 04/18/2023).

[18] OEIS. “Strong pseudoprimes to base 3.” (2023), [Online]. Available: https://oeis.org/A020229 (visited on 04/18/2023).

[19] OEIS. “Strong pseudoprimes to base 5.” (2023), [Online]. Available: https://oeis.org/A020231 (visited on 04/18/2023).

[20] GeeksForGeeks. “Wilson’s theorem.” (2023), [Online]. Available: https://www.geeksforgeeks.org/wilsons-theorem/ (visited on 05/02/2023).

[21] R. J. McIntosh. “Wolstenholme’s theorem.” (1995), [Online]. Available: http://matwbn.icm.edu.pl/ksiazki/aa/aa71/aa7144.pdf (visited on 04/15/2023).