The Comparison of Prime Number Testing Algorithms: Miller-Rabin, Eratosthenes and Atkin

I. INTRODUCTION

In computer science, prime numbers play a significant role in various applications, particularly in cryptography. Prime numbers are utilized to generate public and private keys in cryptographic systems like RSA. Their unique mathematical properties, especially their indivisibility except by 1 and themselves, make them ideal for secure encryption (Pande, 2015).

To determine prime numbers efficiently, several algorithms have been developed. These include **prime number generation algorithms** such as:

- **The Sieve of Eratosthenes**
- **The Sieve of Atkin**
- **The Sieve of Sundaram**
- **Mersenne Prime Generation** (Maltare & Chudasama, 2016).

In addition to generating prime numbers, **primality testing algorithms** are used to verify whether a given number is prime. These include:

- **Miller-Rabin Primality Test** (probabilistic),
- **Fermat's Little Theorem**,
- **Solovay-Strassen Test**, and
- **AKS Primality Test** (Tarafder & Chakroborty, 2019).

Previous Studies

Several relevant studies have explored prime number generation and primality testing:

1. **Kochar & Goswami (2016)** compared the Sieve of Eratosthenes and the Sieve of Sundaram. The results showed that the **Sieve of Eratosthenes** outperformed the Sieve of Sundaram in terms of speed and simplicity.
2. **Apdilah, Harahap, & Khairina (2017)** generated large prime numbers using the **Mersenne Prime Number algorithm** and validated their results using the **Miller-Rabin Primality Test**. This study demonstrated the importance of large primes in RSA cryptography for secure public and private keys.
3. **Rajput & Bajpai (2019)** performed a comparative analysis of **probabilistic and deterministic primality testing algorithms**, including **Fermat, AKS, Miller-Rabin**, and **Solovay-Strassen**. The results highlighted that the **AKS algorithm** is the most efficient deterministic algorithm.

Objective of the Study

This project aims to implement and compare three prime number testing algorithms:

1. **Miller-Rabin Primality Test**,
2. **The Sieve of Eratosthenes**, and
3. **The Sieve of Atkin**.

The implementation will allow users to:

- Select a specific algorithm from a simple user interface.
- Test the primality of a given large number.

The comparative analysis will focus on the **performance** (execution time) and **accuracy** of these algorithms for different inputs. Results will be presented in the form of tables and graphs.

Fermat Primality Test Algorithm

Testing of prime numbers can use several algorithms, namely by testing prime numbers with Fermat or using the Miller Rabin algorithm. In this discussion only using prime number testing algorithms using Fermat's Theory. Testing prime numbers with the Fermat algorithm is a probability, so it needs to be tested several times until then it can be ascertained that the prime number is deterministic.

a (n-1) mod n ≡ 1 , where 1 < a < n ..... (1)

a = random number

n = prime number

If the modulo results are ≠ 1, then n is not a prime number (Agrawal, 2006).

Example: Test number 37, whether prime or not prime.

We take a = 2, n = 37, then :

2 (37 – 1) ≡ 1 (mod 37) = 2 36 mod 37 = 1.

Because the modulo results are 1, then number 37 is a prime number.

Miller-Rabin Algorithm for Testing Primality

The following is the process of the Miller-Rabin algorithm in testing whether a number n n is prime:

Step - 1: For a given number n , check if n is 2 or 3. If so, it is prime. If n is less than 2 or even, it is composite.

Step - 2: Decompose $n-1$ into the form $2^s \cdot d$, where d is an odd integer and s is the number of times $n-1$ can be divided by 2.

Step - 3: Choose a random integer a in the range $2 \le a \le n-2$ .This a is called a witness.Compute $x = a^d$ m o d n (modular exponentiation).

Step - 4: Check the following conditions:

If $x \equiv 1 \pmod{n}$ or $x \equiv n-1 \pmod{n}$,n passes this iteration (it may still be prime).

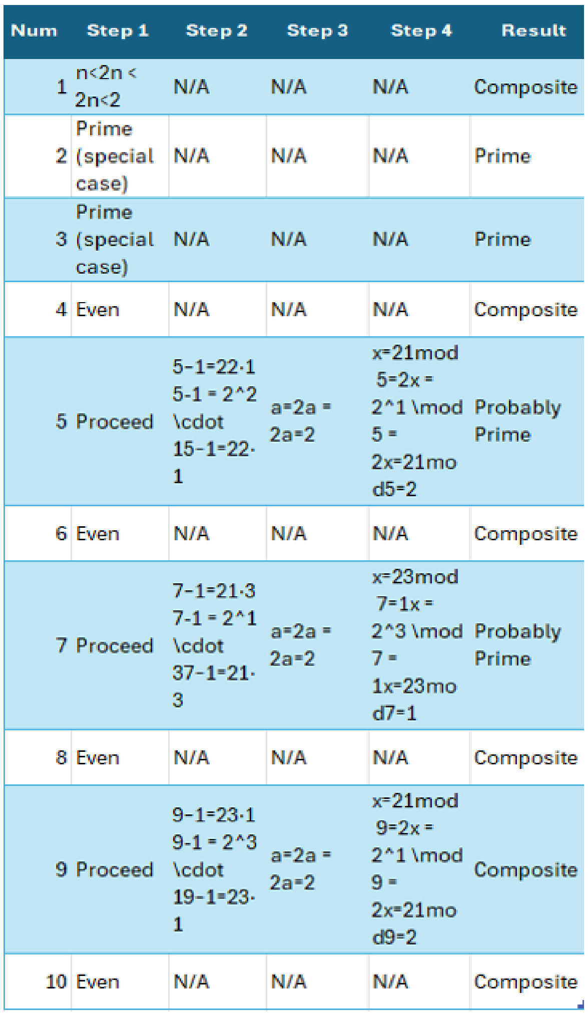
Otherwise, repeatedly square x (up to $s-1$ times) and check:

If $x^2 \equiv n-1 \pmod{n}$, then n passes this iteration.

If $x \equiv 1 \pmod{n}$ during squaring, n is composite.

Step - 5: Repeat Steps 3 and 4 for k k iterations (where k k is the number of witnesses chosen).

If n n fails any iteration, it is composite. If n n passes all k k iterations, n n is considered probably prime.



Num	Step 1	Step 2	Step 3	Step 4	Result
1	n<2n < 2n<2	N/A	N/A	N/A	Composite
2	Prime (special case)	N/A	N/A	N/A	Prime
3	Prime (special case)	N/A	N/A	N/A	Prime
4	Even	N/A	N/A	N/A	Composite
5	Proceed	5-1=22·1 5-1 = 2^2 \cdot 15-1=22·1	a=2a = 2a=2	x=21mod 5=2x = 2^1 \mod 5 = 2x=21mod5=2	Probably Prime
6	Even	N/A	N/A	N/A	Composite
7	Proceed	7-1=21·3 7-1 = 2^1 \cdot 37-1=21·3	a=2a = 2a=2	x=23mod 7=1x = 2^3 \mod 7 = 1x=23mod7=1	Probably Prime
8	Even	N/A	N/A	N/A	Composite
9	Proceed	9-1=23·1 9-1 = 2^3 \cdot 19-1=23·1	a=2a = 2a=2	x=21mod 9=2x = 2^1 \mod 9 = 2x=21mod9=2	Composite
10	Even	N/A	N/A	N/A	Composite

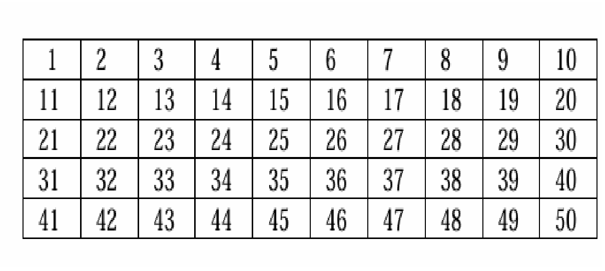
Sieve of Eratosthenes

The following is the process of the Sieve of

Eratosthenes algorithm in generating prime numbers:

Step - 1 :

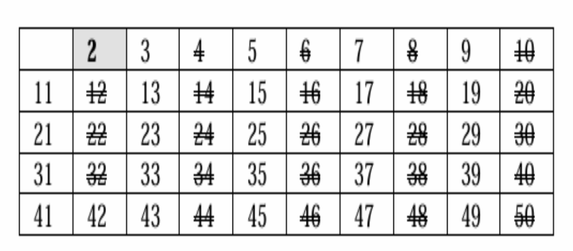
Make a list of numbers from 1 to n



1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Step - 2 :

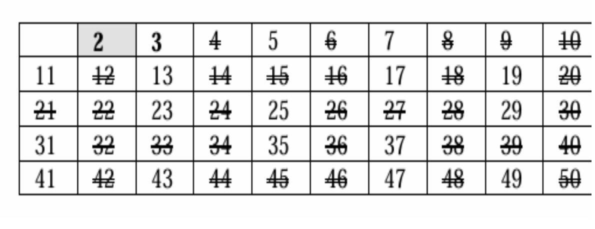
Mark the first prime number in number 2 (Pande, Prime Generating Algorithms by Skipping Composite Divisors, 2014), then mark and cross all numbers that are multiples of number 2 (E.O'Neill, 2009).



	2	3	~~4~~	5	~~6~~	7	~~8~~	9	~~10~~
11	~~12~~	13	~~14~~	15	~~16~~	17	~~18~~	19	~~20~~
21	~~22~~	23	~~24~~	25	~~26~~	27	~~28~~	29	~~30~~
31	~~32~~	33	~~34~~	35	~~36~~	37	~~38~~	39	~~40~~
41	42	43	~~44~~	45	~~46~~	47	~~48~~	49	~~50~~

Step - 3 :

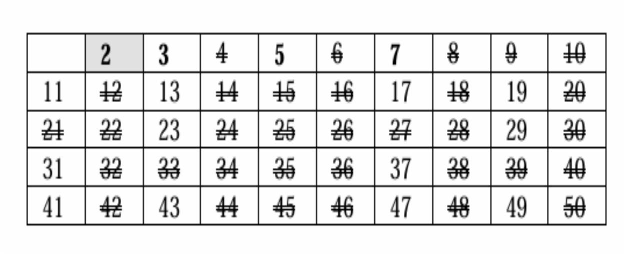
Mark the next prime number in number 3 then mark and cross all numbers which are multiples of 3.



	2	**3**	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	25	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	35	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~

Step - 4 :

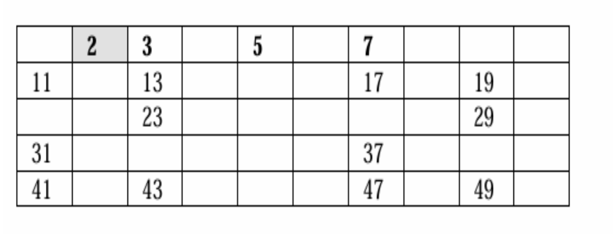
Repeat steps - 2 and step - 3 until all multiples of numbers are crossed.



	2	**3**	~~4~~	**5**	~~6~~	**7**	~~8~~	~~9~~	~~10~~
11	~~12~~	13	~~14~~	~~15~~	~~16~~	17	~~18~~	19	~~20~~
~~21~~	~~22~~	23	~~24~~	~~25~~	~~26~~	~~27~~	~~28~~	29	~~30~~
31	~~32~~	~~33~~	~~34~~	~~35~~	~~36~~	37	~~38~~	~~39~~	~~40~~
41	~~42~~	43	~~44~~	~~45~~	~~46~~	47	~~48~~	49	~~50~~

Step – 5 :

Prime numbers are numbers from 2-n that are not crossed during the multiple deletion process of the previous number (Apdilah, Harahap, & Khairina, 2017)



	2	**3**		**5**		**7**			
11		13				17		19	
		23						29	
31						37			
41		43				47		49	

The series of prime numbers between 2 - 50 are : 2, 3,5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

The Sieve of Atkins

Following is the process of The Sieve of Atkins algorithm in filtering prime numbers:

Step – 1 :

Determine the search for prime numbers of numbers from 1 – n, where in this study we will generate prime numbers from 1- 50, so that n = 50.

Step – 2 :

Mark numbers 2, 3 and 5 as prime numbers.

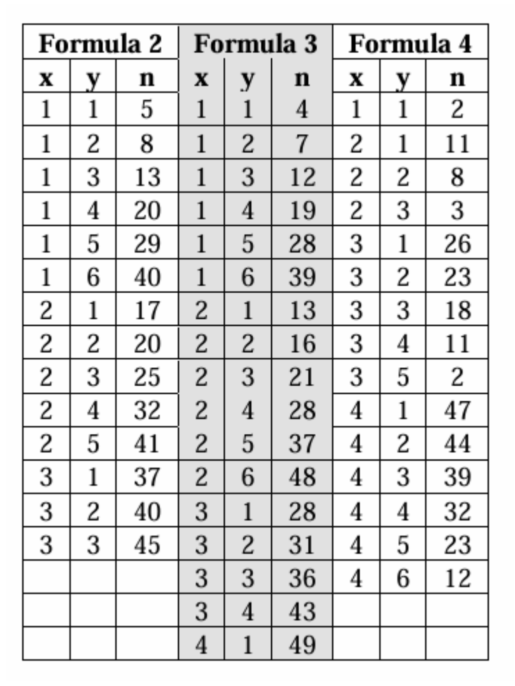
Step – 3 :

Calculate the equation based on the following 3 blocks:

4x2 + y2 mod 4 => 1 or 5 4x2 + y2 < n....(2)

3x2 + y2 mod 6 => 1   3x2 + y2 < n ...... (3)

3x2 – y2 mod 12 => 11   0 < 3x2 – y2 < n .. (4)



Formula 2			Formula 3			Formula 4		
x	y	n	x	y	n	x	y	n
1	1	5	1	1	4	1	1	2
1	2	8	1	2	7	2	1	11
1	3	13	1	3	12	2	2	8
1	4	20	1	4	19	2	3	3
1	5	29	1	5	28	3	1	26
1	6	40	1	6	39	3	2	23
2	1	17	2	1	13	3	3	18
2	2	20	2	2	16	3	4	11
2	3	25	2	3	21	3	5	2
2	4	32	2	4	28	4	1	47
2	5	41	2	5	37	4	2	44
3	1	37	2	6	48	4	3	39
3	2	40	3	1	28	4	4	32
3	3	45	3	2	31	4	5	23
			3	3	36	4	6	12
			3	4	43			
			4	1	49			

Step – 4 :

In this step, there are two steps, namely:

a. Delete the value of n which the module results do not match with the requirements of modulo formulas 2, 3 and 4.

b. Delete numbers that are not square-free numbers

 Explanation of square-free numbers:

Square-free numbers are integer integers that cannot be divided by perfect squaresother than 1. That is, the main factorization has exactly one factor for each prime that appears in it.

As an example, 10 = 2 · 5 is quadratic free, but 18 = 2 · 3 · 3 is not square-free numbers, because 18 can be divided by 9 = 2.

Can be seen in the table below, numbers that are not square-free numbers are 25, 45, 49.

Proof that 3 numbers below are not square-free are:

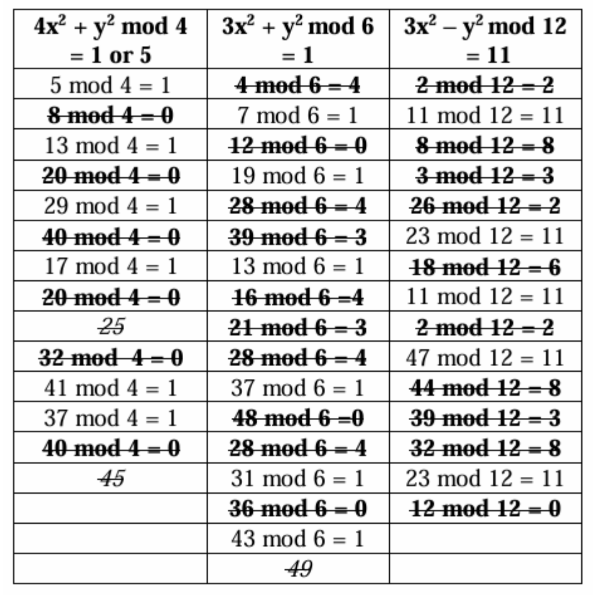
25 => is divided by 25 which is the square of 5.

Example → 25 : 52 = 1 45 => is divided by 45 which is the square of 3.

Example → 45 : 32 = 1

49 => is divided by 49 which is the square of 7

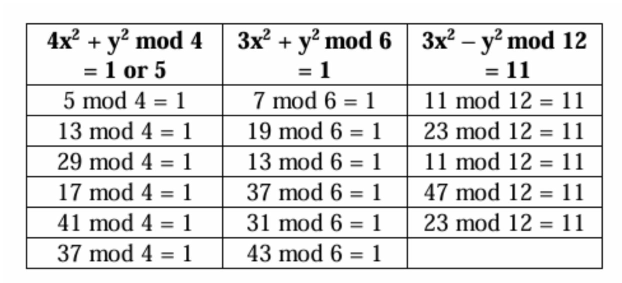
Example → 49 : 72 = 1



$4x^2 + y^2$ mod 4 = 1 or 5	$3x^2 + y^2$ mod 6 = 1	$3x^2 - y^2$ mod 12 = 11
5 mod 4 = 1	~~4 mod 6 = 4~~	~~2 mod 12 = 2~~
~~8 mod 4 = 0~~	7 mod 6 = 1	11 mod 12 = 11
13 mod 4 = 1	~~12 mod 6 = 0~~	~~8 mod 12 = 8~~
~~20 mod 4 = 0~~	19 mod 6 = 1	~~3 mod 12 = 3~~
29 mod 4 = 1	~~28 mod 6 = 4~~	~~26 mod 12 = 2~~
~~40 mod 4 = 0~~	~~39 mod 6 = 3~~	23 mod 12 = 11
17 mod 4 = 1	13 mod 6 = 1	~~18 mod 12 = 6~~
~~20 mod 4 = 0~~	~~16 mod 6 =4~~	11 mod 12 = 11
~~*25*~~	~~21 mod 6 = 3~~	~~2 mod 12 = 2~~
~~32 mod 4 = 0~~	~~28 mod 6 = 4~~	47 mod 12 = 11
41 mod 4 = 1	37 mod 6 = 1	~~44 mod 12 = 8~~
37 mod 4 = 1	~~48 mod 6 =0~~	~~39 mod 12 = 3~~
~~40 mod 4 = 0~~	~~28 mod 6 = 4~~	~~32 mod 12 = 8~~
~~*45*~~	31 mod 6 = 1	23 mod 12 = 11
	~~36 mod 6 = 0~~	~~12 mod 12 = 0~~
	43 mod 6 = 1	
	~~*49*~~	

Step – 5 :

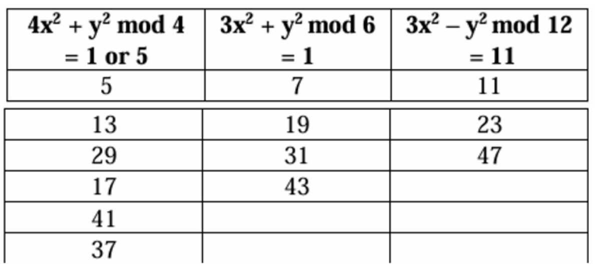
After step - 4, list the remaining selection numbers:



$4x^2 + y^2$ mod 4 = 1 or 5	$3x^2 + y^2$ mod 6 = 1	$3x^2 - y^2$ mod 12 = 11
5 mod 4 = 1	7 mod 6 = 1	11 mod 12 = 11
13 mod 4 = 1	19 mod 6 = 1	23 mod 12 = 11
29 mod 4 = 1	13 mod 6 = 1	11 mod 12 = 11
17 mod 4 = 1	37 mod 6 = 1	47 mod 12 = 11
41 mod 4 = 1	31 mod 6 = 1	23 mod 12 = 11
37 mod 4 = 1	43 mod 6 = 1	

Step – 6 :

Delete repeated numbers in step number list - 5, and recreate a new list.



$4x^2 + y^2$ mod 4 = 1 or 5	$3x^2 + y^2$ mod 6 = 1	$3x^2 - y^2$ mod 12 = 11
5	7	11
13	19	23
29	31	47
17	43	
41		
37		

Arrange the prime numbers obtained in this step fromthe smallest to the largest number.

Here is the intermediate prime number 1-50 : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47

OBSERVATIONS AND ANALYSIS

Miller-Rabin Algorithm

- **Nature of Algorithm**:
  Miller-Rabin is a **probabilistic primality test**. It verifies whether a given number nnn is prime by testing it against randomly chosen "witnesses" aaa. The algorithm works by decomposing n−1n-1n−1 into the form 2s·d2^s \cdot d2s·d and using modular exponentiation to identify nontrivial square roots of 1 or violations of Fermat's theorem.
- **Execution Time**:
  - Small inputs (e.g., n=7n = 7n=7) take microseconds (~ **0.000008s** to **0.000009s**).
  - Large inputs (e.g., n=104729n = 104729n=104729 or n=99999989n = 99999989n=99999989) remain efficient (~ **0.000021s** to **0.000030s**) due to logarithmic modular exponentiation.
  - Complexity: O(klog3n)O(k \log^3 n)O(klog3n), where kkk is the number of iterations (default k=5k = 5k=5).
- **Strengths**:
  - Extremely fast for testing a single large number.
  - Scales efficiently for 64-bit or 128-bit numbers, making it ideal for cryptographic applications.
- **Limitations**:
  - Probabilistic in nature, leading to a small chance of **false positives** (composite numbers identified as prime).
  - Does not generate all prime numbers up to a limit nnn.

Sieve of Eratosthenes

- **Nature of Algorithm**:
  The Sieve of Eratosthenes is a **deterministic algorithm** that generates all prime numbers up to nnn. It works by iteratively marking multiples of known primes as composite in a boolean array.
- **Execution Time**:
  - Small inputs (e.g., n=7n = 7n=7) are processed in milliseconds (~ **0.000003s** to **0.000004s**).
  - Large inputs (e.g., n=10472 or n=99999989) take significantly longer (~ **0.008s** to **10.76s**)

because the algorithm processes all numbers up to nnn.

- Complexity: O(nloglogn)O(n \log \log n)O(nloglogn).

- **Strengths**:
  - Simple and intuitive implementation.
  - Guarantees 100% accuracy for generating all prime numbers.
- **Limitations**:
  - Inefficient for testing a **single large number** since it computes primes for the entire range.
  - Memory-intensive for very large nnn.

Sieve of Atkin

- **Nature of Algorithm**:
  The Sieve of Atkin is an **optimized version** of the Sieve of Eratosthenes. It uses quadratic equations 4x2+y24x^2 + y^24x2+y2, 3x2+y23x^2 + y^23x2+y2, and 3x2−y23x^2 - y^23x2−y2 to mark numbers as prime or composite.
- **Execution Time**:
  - Small inputs (e.g., n=7) are slightly slower (~ **0.000005s** to **0.000009s**) due to additional quadratic checks.
  - Large inputs (e.g., n=104729or n=99999989) take longer (~ **0.051s** to **5.97s**) compared to Eratosthenes due to its computational overhead.
  - Complexity: O(n/loglogn)O(n / \log \log n)O(n/loglogn).
- **Strengths**:
  - Optimized for generating primes in large ranges, making it more efficient for very large nnn.
- **Limitations**:
  - Not efficient for single-number testing.
  - Complex to implement, especially for smaller inputs.

KEY TRENDS AND OBSERVATIONS

1. **Miller-Rabin Dominates for Single-Number Testing**:
   - Miller-Rabin is much faster because it directly tests nnn, avoiding computations for all numbers up to nnn.
   - Its time complexity is logarithmic, making it ideal for large inputs.
2. **Sieves (Eratosthenes and Atkin) are Overkill for Single-Number Testing**:
   - Both algorithms compute primes for the entire range

up to nnn, which is unnecessary when testing just one number.

3. **Performance Comparison**:
   - **Small Inputs**: Differences between the algorithms are negligible.
   - **Large Inputs**:
     - **Miller-Rabin**: Microseconds (e.g., 0.000021s).
     - **Sieve of Eratosthenes**: Slower(e.g.,10.76s)
     - **Sieve of Atkin**: Slightly slower than Eratosthenes (e.g., 5.97s).
4. **Complexity Growth**:
   - **Miller-Rabin**: O(klog3n)O(k \log^3 n)O(klog3n) → Efficient for single-number testing.
   - **Eratosthenes**: O(nloglogn)O(n \log \log n)O(nloglogn) → Becomes slow for large nnn.
   - **Atkin**: O(n/loglogn)O(n / \log \log n)O(n/loglogn) → Optimized for generating all primes but computationally intensive for smaller ranges.

RESULT AND DISCUSSION

The comparison in this discussion focuses on the **accuracy** and **speed** of the three algorithms to determine large prime numbers.

Accuracy Comparison

Accuracy was tested using the Fermat Primality Test as a benchmark. Results showed:

- **Miller-Rabin**: Small chance of false positives.
- **Eratosthenes & Atkin**: 100% accurate.

Speed Comparison

For generating primes in the range 1–1,000,000:

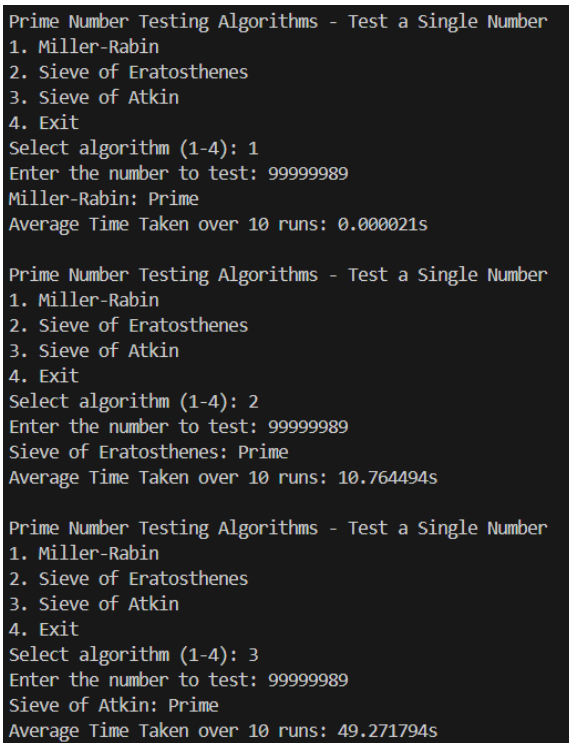
- **Miller-Rabin**: Fastest for single-number testing.
- **Sieve of Eratosthenes**: Faster for generating primes.
- **Sieve of Atkin**: Slightly slower than Eratosthenes.

CONCLUSION

1. **Miller-Rabin** is the best choice for testing the primality of a single large number.
2. **Sieve of Eratosthenes** is more efficient for generating all primes up to nnn.
3. **Sieve of Atkin** performs well for large ranges but is computationally complex for smaller inputs.

Output examples

Testing prime number 99999989

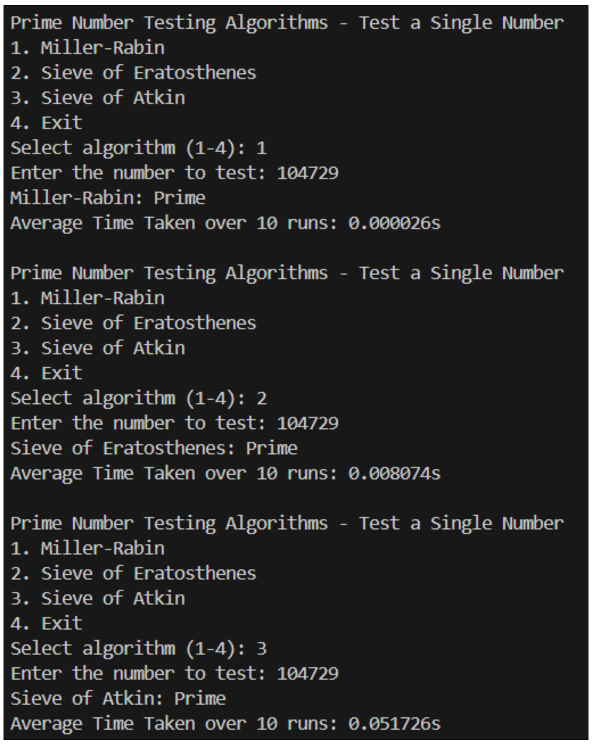


```
Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 1
Enter the number to test: 99999989
Miller-Rabin: Prime
Average Time Taken over 10 runs: 0.000021s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 2
Enter the number to test: 99999989
Sieve of Eratosthenes: Prime
Average Time Taken over 10 runs: 10.764494s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 3
Enter the number to test: 99999989
Sieve of Atkin: Prime
Average Time Taken over 10 runs: 49.271794s
```

Testing prime number 104729

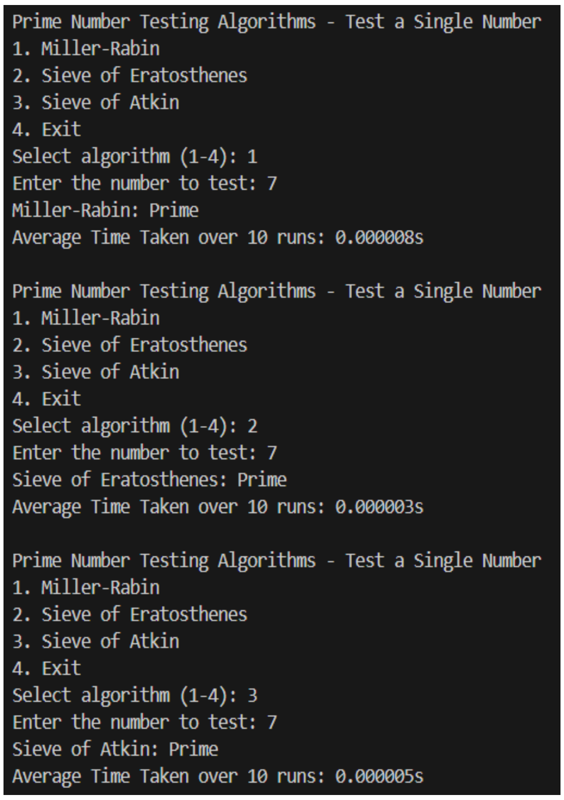


```
Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 1
Enter the number to test: 104729
Miller-Rabin: Prime
Average Time Taken over 10 runs: 0.000026s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 2
Enter the number to test: 104729
Sieve of Eratosthenes: Prime
Average Time Taken over 10 runs: 0.008074s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 3
Enter the number to test: 104729
Sieve of Atkin: Prime
Average Time Taken over 10 runs: 0.051726s
```

For small prime number 7

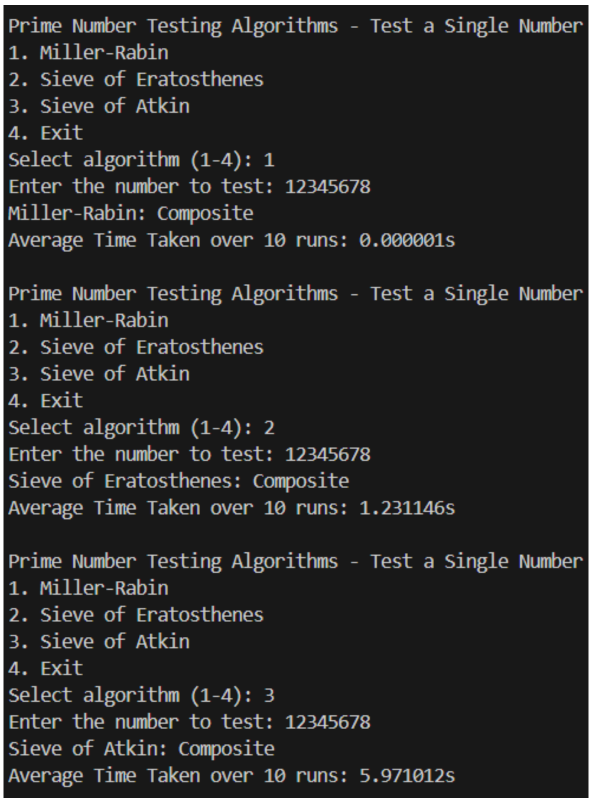


```
Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 1
Enter the number to test: 7
Miller-Rabin: Prime
Average Time Taken over 10 runs: 0.000008s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 2
Enter the number to test: 7
Sieve of Eratosthenes: Prime
Average Time Taken over 10 runs: 0.000003s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 3
Enter the number to test: 7
Sieve of Atkin: Prime
Average Time Taken over 10 runs: 0.000005s
```

For big non-prime number 12345678



```
Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 1
Enter the number to test: 12345678
Miller-Rabin: Composite
Average Time Taken over 10 runs: 0.000001s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 2
Enter the number to test: 12345678
Sieve of Eratosthenes: Composite
Average Time Taken over 10 runs: 1.231146s

Prime Number Testing Algorithms - Test a Single Number
1. Miller-Rabin
2. Sieve of Eratosthenes
3. Sieve of Atkin
4. Exit
Select algorithm (1-4): 3
Enter the number to test: 12345678
Sieve of Atkin: Composite
Average Time Taken over 10 runs: 5.971012s
```