INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY
HYDERABAD

Course Project

Advanced Problem Solving

Fall 2018

Comparison of Probabilistic and Deterministic algorithms for Primality Test

Author:
Tarun Vatwani
Vivek Patare

Supervisor:
Prof. Venkatesh Chopella
Prof. Vikram Pudi

November 12, 2018

Abstract

Primality testing is one of the important problems in number theory. Unlike Prime factorization, primality test only verifies whether a number is prime or not, that is why primality testing a relatively easier problem than factorization which is a computationally difficult problem. On the other hand primality test is a polynomial time solvable problem.

There exist several algorithms in theory to check whether a number is or not, but not all of them are deterministic in nature. Deterministic tests determine absolutely whether a given number is prime or not. Probabilistic tests may, with some small probability, identify a composite number as a prime, although not vice-versa.

Aim of this report is to explain basic idea of some of the deterministic and probabilistic algorithms and present a comparison analysis of those algorithms.

Contents

Chapter 1

Introduction

Prime numbers are natural numbers with no positive divisors other than 1 and the number itself. To check whether a natural number is a prime number or not is a very important problem in number theory. Apart from being a fundamental problem it has various practical applications such as Cryptography, Hash Functions, Pseudo Random Number generators, etc.
In Cryptography prime numbers are used to compute the cipher text of given plain text through encryption

$$CipherText = (PlainText^e) \bmod n \tag{1.1}$$

where n is the product of two large distinct prime numbers and e is the public key [1].
Whereas in Hash functions prime numbers are used to compute hash value of a data

$$h_{a,b}(x) = [((ax + b) \bmod p)] \bmod n \tag{1.2}$$

where p is a prime number [2].
Also the Linear congruential generator which is used to generate pseudo random numbers has the following recurrence form

$$X_{n+1} = [aX_n + c] \bmod m \tag{1.3}$$

where c and m are relatively prime, which implies that distinct prime numbers can be chosen for c and m [3].

1.1 Motivation

The need to explore the probabilistic and deterministic algorithms arises from the fact that the naive trial division method to check whether a number is prime or not is not very efficient. The trial division method is mentioned below:

Algorithm 1 Trial Division

```
INPUT: Integer $n \geq 2$
OUTPUT: prime:0, composite:1
1: integer : $i \leftarrow 2$
2: while $i^2 \geq$n do
3:     if $i$ divides $n$ then
4:         return 1;
5:     $i \leftarrow i + 1$
6: return 0;
```

The above algorithm takes a natural number n as a input and loops it for maximum $\sqrt{n}$ iterations if the number is prime. So typically it takes around O($\sqrt{n}$) time to check whether the number is prime or not. Which seems plausible for small input sized numbers.

The need for such testing large prime numbers arises in application such as RSA where we require prime numbers consisting of few 100 digits which would require around 10^{50} operations.

To keep with needs of Data Science and Cryptography various probabilistic methods were proposed. Using the extended ideas of Fermat's theorem, Sollovay and Strassen, in 1977, and Miller and Rabin, in 1980, probabilistic primality tests were developed. Until 2002 primality problem belonged to NP-class of problems. In 2002 the a deterministic algorithm to check primality was proposed. Based on these previous attempts, we decided to implement the improved methods and compare their performance for further study. This forms the main idea of the project.

1.2 Objectives

The purpose of this project is to compare Probabilistic and Deterministic algorithms for Primality Testing, explain how these algorithm work, and present their performance. Most of the algorithms and study references are based on book *Primality Testing in Polynomial Time: From Randomized Algorithms to "PRIMES is in P"*[4]
Objectives are:

- Understand the mathematical basics in detail.
- Fit the relevant knowledge into the algorithms.
- Produce functions to test input numbers, based on the pseudo-codes.
- Evaluate the performance of different methods by various approaches.
- Find speed-accuracy trade-off between Probabilistic and Deterministic algorithms if possible.

1.3 Report Structure

- Chapter 2 investigates three typical randomized algorithms (Lehman's Test, Solovay Strassen Test, Miller-rabin Test), and explaining how they work in brief.
- Chapter 3 talks about Deterministic Algorithm(AKS Test) and basic idea behind it.
- Chapter 4 talks about implementation details of each of these four algorithms, libraries used for it and details user-end testing of algorithms and considers statistics gathered from the running of previous algorithms.
- Chapter 5 concludes the report with inferences drawn from statistics gathered.

Chapter 2

Probabilistic Algorithms for primality test

In this chapter we will discussing about several non-deterministic algorithms and intuitions behind working of these algorithms. The probabilistic algorithms that we will discuss are based on the idea of Fermat's little theorem which is described as follows:

Theorem 1. Fermat's Theorem
Let p be a prime number, and let x be an integer such that gcd(x, p) = 1 then $x^{p-1} \equiv 1 (\text{mod } p)$

So Fermat's theorem is only a necessary condition for proving primality of a number. It correctly identifies all the prime numbers but fails for certain composite numbers which exhibit the same property as prime numbers.

Similarly the probabilistic methods that are discussed in this section does not guarantee the correct classification but the probability of giving a wrong answer can be calculated for all the methods.

2.1 Lehmann's Test

This algorithm was proposed by DJ Lehmann in 1982. This is a very simple algorithm and its correctness and error probability is based on the concepts of algebraic structures in number theory [5]. Mentioned below is the algorithm for Lehman's Test:

Algorithm 2 Lehmann's Test

```
INPUT: Odd Integer ≥ 3, Integer l ≥ 2
OUTPUT: prime:0, composite:1
Integer: $a, c$;
Array of Integers: $b[1 : l]$
for $i = 0; i < l; i++$ do
    Let $a$ be randomly chosen from $[1, ..., n-1]$;
    $c \leftarrow a^{(n-1)/2} \bmod n$
    if $c \notin (1, n-1)$ then
        return 1;
    else
        $b[i] \leftarrow c$

for $i = 0; i < l; i++$ do
    if $b[i] == n-1$ then
        return 0;
return 1;
```

The above algorithm can be summarized as:

1. A random integer a is selected from the range $[1 : n-1]$ so that $\gcd(a, n)$ is 1

2. Calculate $a^{(n-1)/2} \bmod n$ and assign it to c
3. If $c = 1$ or $n - 1$, then store value of c in array otherwise break the loop and return n as composite
4. Run the loop l times with different random values of a
5. If any element in array is not equal to 1, return n as prime. Otherwise, return n as composite

In the above method if n is a prime number then the probability of wrong output is exactly 2^{-l} whereas if n is odd composite the probability of having false prime is no more than 2^{-l}. The running time of above algorithm is $O(l(\log n)^3)$ [4]

2.2 Solovay Strassen Test

This algorithm was proposed by R. Solovay and V. Strassen in 1977. Unlike Lehmann's test it is capable of identifying primes with a probability of 1 but it is capable of composite numbers with a probability of 0.5[6] To build a basic understanding of the algorithm we introduce Jacobi Symbols

Definition 2.2.1. Jacobi Symbols
Let $n \geq 3$ be an odd integer with prime decomposition $n = p_1.p_2.p_3....p_r$ we define $(a/n) = (a/p_1)....(a/p_r)$ as the Jacobi Symbol of $aandn$, where $(a/p_1), (a/p_2)...., (a/p_r)$ are called the Legendre Symbols [4]

Properties of Jacobi Symbols are summarized as Quadratic Reciprocity Law, which is as follows:

Theorem 2. Quadratic Reciprocity Law
If $a, n \geq 3$ are odd integers then (a/n) can be calculated by the following rules:

1. If a is not in the interval $1, ..., n - 1$ the result is Jacobi of $a \bmod n$ and n
2. If $a = 0$ then result is 0
3. If $a = 1$ result is 1
4. If $4dividesa$ then result is Jacobi of $a/4$ and n
5. If $2dividesa$ the result is Jacobi of $a/2$ and n if $n \bmod 8 \in (1, 7)$, and Jacobi of $-a/2$ and n if $n \bmod 8 \in (1, 7)$
6. If $a \equiv 1$ or $n \equiv 1(\text{mod}4)(a > 1)$, the result is Jacobi of $n \bmod a$ and n
7. If $a \equiv 3$ or $n \equiv 3(\text{mod}4)(a > 1)$, the result is Jacobi $-n \bmod a$ and n

Now the Algorithm of Solovay Strassen is as following:

Algorithm 3 Solovay Strassen's Test

INPUT: Odd Integer ≥ 3
OUTPUT: prime:0, composite:1

```
1: Integer: $a, c$;
2: Let $a$ be randomly chosen from $[2, ..., n - 1]$;
3: $c \leftarrow a^{(n-1)/2}.(a/n) \bmod n$
4: if $b \neq 1$ then
5:     return 1;
6: else
7:     return 0;
```

In the above method if n is odd composite the probability of having false prime is no more than 2^{-l} and it correctly classifies all the prime numbers as prime numbers i.e there are no false composites. The running time of above algorithm is $O(c(\log n)^3)$ [4]

2.3 Miller-Rabin Test

Miller-Rabin is non deterministic primality testing algorithm first version of which is discovered by Gary L. Miller which was originally designed to be deterministic but depended on unproven hypothesis. Michael O. Rabin then converted it to probabilistic version.
Algorithm is as follows:

Algorithm 4 Miller-Rabin Test

```
INPUT: Odd Integer $\geq 3$
OUTPUT: prime:0, composite:1
1: Find $u$ and $k$ satisfy $n = u \cdot 2^{\mathrm{k}}$
2: Let $a$ be randomly chosen from $[2, ..., n-2]$;
3: $b \leftarrow a^{\mathrm{u}} \bmod n$
4: if $b \in (1, n-1)$ then
5:     return 0;
6: for $i = 1; i < k; i++$ do
7:     $b \leftarrow b^2 \bmod n$
8:     if $b = n-1$ then
9:         return 0;
10:    if $b = 1$ then
11:        return 1;
12: return 1;
```

Algorithm can be summarized as:

1. Line 3:
   Set b as $a^{\mathrm{u}} \bmod n$.
2. Line 4-5:
   If $b = 1$ or $n-1$, which means $(a^{\mathrm{u}} - 1) or (a^{\mathrm{u}} + 1) = 0(\bmod n)$, returned n as an unsure prime.
3. Line 6-11:
   Repeat squaring b, $b_{\mathrm{i}} = (a^{u \cdot 2^i}) \bmod n$. Check whether $b_{\mathrm{i}} = -1$, then n is unsure prime. Otherwise the equation $b_{\mathrm{i}} = 1$ held for n is composite.
4. Line 12:
   After $k-1$ times repeating, we can make sure $a^{u \cdot 2^i} - 1 \neq 0(\mathrm{mod} n)$, return composite.

Based on this implementation, time complexity of Miller-Rabin Test is $O(c(\log n)^3)$

Chapter 3

Deterministic Algorithm for primality test

A deterministic algorithm is an algorithm that, always give the same answer with the same input on different runs. When dealing with the primality problem, the deterministic algorithms will never give erroneous outputs. In other words, if we input a prime number, the deterministic algorithms will never output it as a composite. Otherwise the deterministic algorithms will never output composite for a prime input.
In this section we are going to discuss deterministic polynomial time primality test. The first deterministic algorithm was published by M. Agrawal, N. Kayal, and N. Saxena on 2002, in a paper titled ”PRIMES is in P”. This algorithm is also known as the AKS test.

3.1 The AKS Test

AKS algorithm was first algorithm which was proposed in 2004 in paper ’Primes is in P’ [7]. The basic theorem on which the test is based can be stated as follows:

Theorem 3. Let n $>$ 2 be arbitrary, and let a $<$ n be an integer that gcd(a, n) $=$ 1. Then n is prime if and only if $(\mathrm{X} + \mathrm{a})^{\mathrm{n}} \equiv \mathrm{X}^{\mathrm{n}} + \mathrm{a}$ (mod n), for all values of x.

This provides us a way of testing the primality of n, but this way needs to evaluate n coefficients in the worst case, so the time complexity will be O(n). In order to speed up this, the AKS algorithm extends the congruence in Theorem 1 to below:

$$(X + a)^{\mathrm{n}} \equiv X^{\mathrm{n}} + a(modX^{\mathrm{r}} - 1, n) \tag{3.1}$$

where r is an integer. This computation reduces highest maximum degree to r. Based on this, basic algorithm can be written as:

Algorithm 5 AKS Test

INPUT: Integer $n \geq 2$ - decision tree
OUTPUT: prime:0, composite:1

```
 1: if $n = a^{\mathrm{b}}$ for some $a, b \geq 2$ then
 2:     return 1;
 3: $r \leftarrow 2$
 4: while $r < n$ do
 5:     if $r$ divides $n$ then
 6:         return 1;
 7:     if $r$ is a prime number then
 8:         if for all $i$, $(1 \leq i \leq 4\lceil \log n \rceil^2)$ then
 9:             break;
10:     $r \leftarrow r + 1$
11: if $r = n$ then
12:     return 0;
13: for $a$ from 1 to $2\lceil \sqrt{r} \rceil . \lceil \log n \rceil$ do
14:     if $(X + a)^{\mathrm{n}} (\mathrm{mod} X^{\mathrm{r}} - 1) \neq X^{\mathrm{n mod\ r}} + a$ then
15:         return 1;
16: return 0;
```

This can be summarized into five main steps:

1. Line 1:
   Test whether n is a power of an integer, if so return composite.

2. Line 5:
   For all r (r $<$ n) test whether r divides n, if so return composite.

3. Line 7-9:
   For a prime number r, i will be $\mathrm{ord}_{\mathrm{r}}(\mathrm{n})$ if $\mathrm{n}^{\mathrm{i}} \equiv 1$ (mod r). In other words, if we cannot find $i(1 \leq i \leq 4(logn)^2)$ satisfies this equation, then $ord_{\mathrm{r}}(n) > 4(logn)^2$. Break the while loop with the value of r.

4. Line 11-12:
   If $r \geq n$, return prime.

5. Line 13-15:
   For $a = 1$ to $2\sqrt{r}\log n$, check $(X + a)^{\mathrm{n}} \neq X^{\mathrm{n}} + a(\mathrm{mod} X^{\mathrm{r}} - 1, n)$. If any a satisfies this, return composite.

Based on this implementation, running time of AKS comes down to $O(\mathrm{c.logn}^{10.5})$ which is significantly higher than previously discussed probabilistic algorithms. Its implications will be further discussed in Testing and Evaluation.

Chapter 4

Testing and Evaluation

In this chapter we test on the implemented algorithms, in order to make sure the programs perform correctly and meet the relevant requirements. The results are given by figures and tables, together with the comparisons between different algorithms. Further, we evaluate the algorithms; the evaluation employs the results of testing. Tests are carried out from three aspects: run time and accuracy shown by different algorithms.

4.1 Implementation

All the pseudo-codes mentioned above are implemented in C++. When trying to run with largest C++ datatype possible, there were limitations on the size of input possible. To remedy that, all algorithms are implemented using mpz module GMP Library [8] which is free library for arbitrary precision arithmatic. This Makes testing algorithms on virtually any number possible. All the implemented source codes are available at github repository:
https://github.com/vivek-2018201078/APS_PROJECT_2018 [9].
You can test any arbitrary number having any digit. make sure you have installed gmp library in your machine before running code. Command for which is:

```
$ sudo apt-get install libgmp3-dev
```

For compiling any code make sure to use -lgmp flag such as:

```
$ g++ aks.cpp -lgmp -o aks
```

You can give arbitrary input number and code will return if number is prime or not such as:

```
$ ./aks
AKS Primality test
number:12354532527675755756757547547
composite
time = 0.042260
```

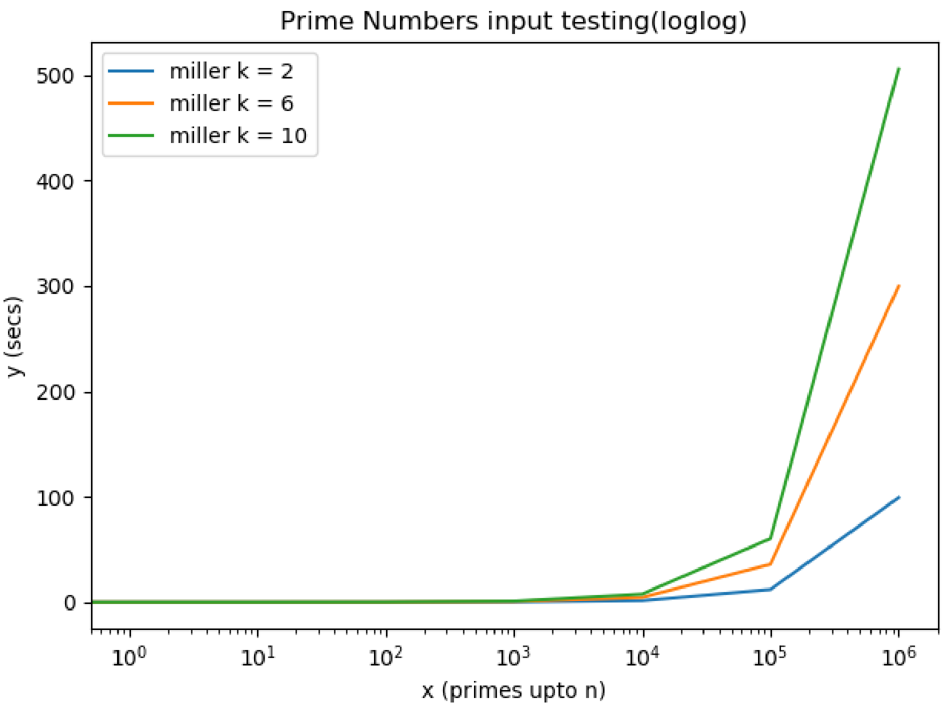
or you can give file input as argument with file consisting of numbers separated by newline example for file consisting of all numbers 2, 4, 5 ,7 ,8, 9:

```
$ ./aks input.txt
AKS Primality test
2: prime
4: composite
5: prime
7: prime
8: composite
9: composite
total time = 0.200113
total primes = 4
```

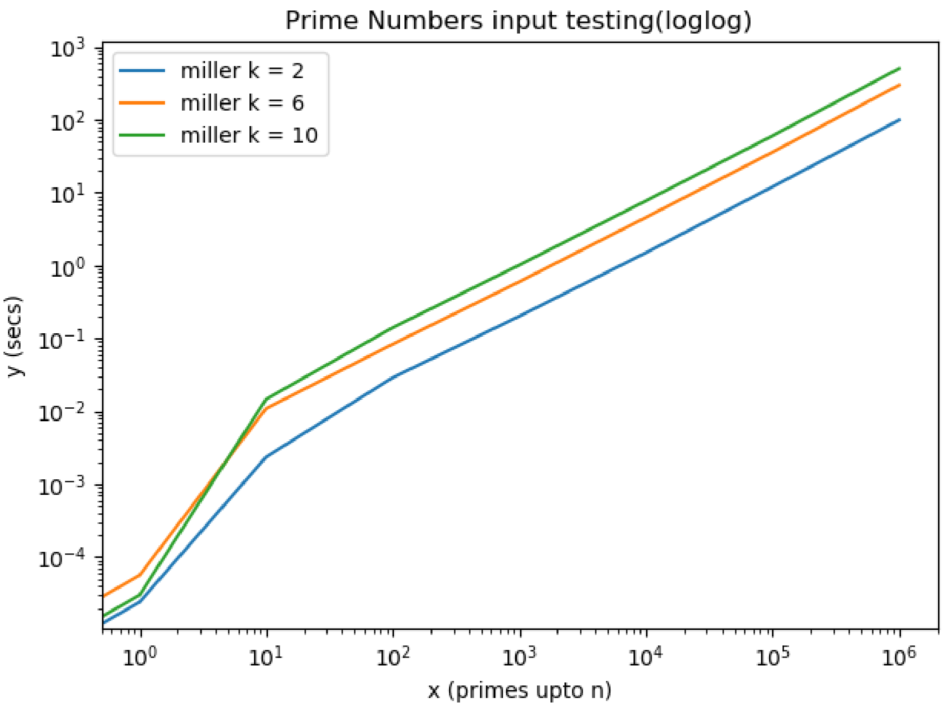
Based on these implementations test cases and performance statistics are measured.

4.2 Run Time of algorithms

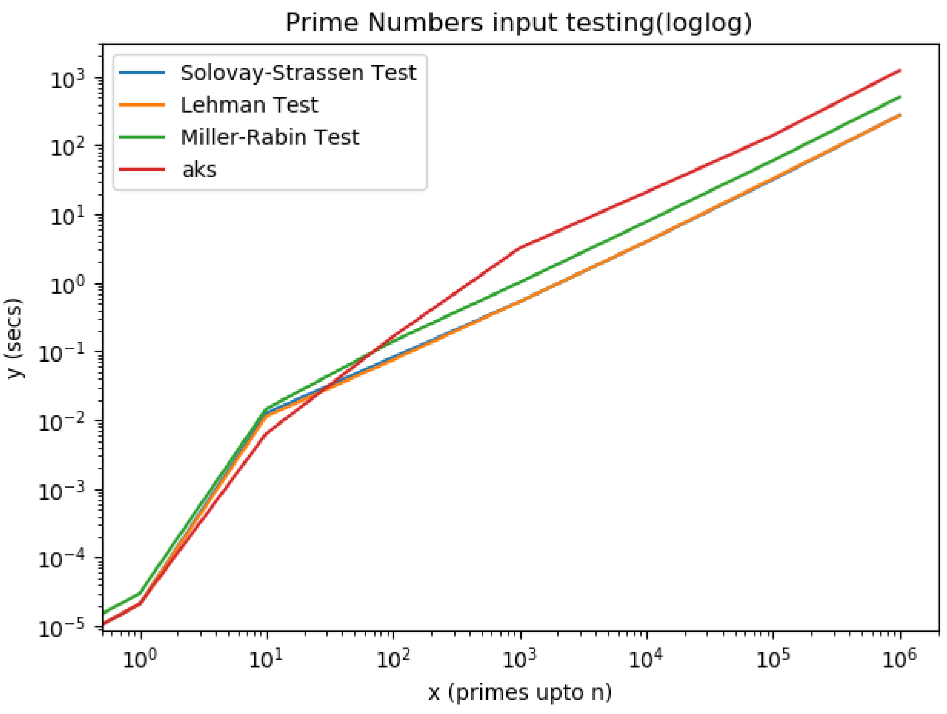
In this section we discuss how testing data was conducted. All the tests were completed by similar methods except some tricky parts. In order to generate enough input data, a program will execute the algorithms repeatedly. Program is tested on multiple values of iteration k.
To get general idea of how value of k affects running time, we first try to compute running time for miller-rabin test with different iteration each time. Testset for this is prime numbers upto digit provided on x axis. Result when plotted out on graph looks like:



According to the time complexity, the relationship between input number and run time must in the form $a \cdot (\log x)^b$ so plotting loglog graph, new graph looks like:



By calculating slope of each line, we can interpolate the formula and conclude that running time of Rabin-Miller increases according to value of iterations. Analyzing the influence of iterations for every algorithm is unnecessary. Therefore, we start comparing the run time between algorithms. Iteration time was set as 10 here for more obvious differences. Following is the figure illustrating the run time of four algorithms: Lehmanns test, the Solovay-Strassen test, the Miller-Rabin test and the AKS test.



From this graph, it gives us straightforward results of running time of algorithms. In three non-deterministic algorithms, Lehmanns test is the fastest one and Miller-Rabin test is the slowest under the same input. The AKS test runs much slower than other tests. This means, if the input number has hundreds of digits, the AKS test will use far more time than others.

4.3 Accuracy of Algorithms

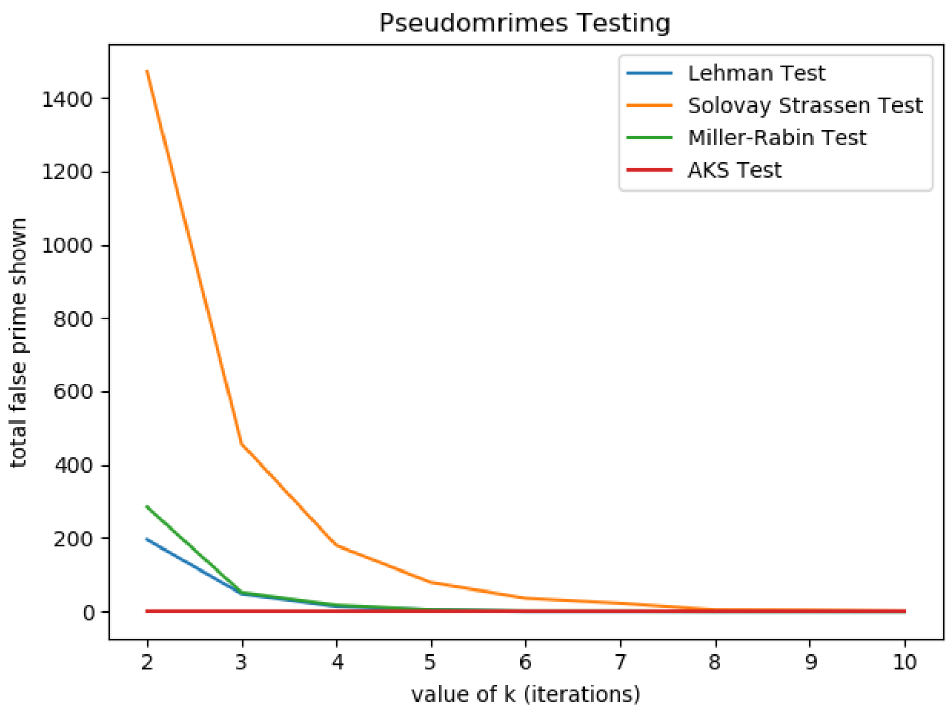
This section will focus on the algorithms accuracy. For primality testing, the accuracy is the degree to which the output is right and consequently we need to count the number of right outputs. By checking the output for upto 3 iterations with known answers, we get following table:

Lehman's Test			
Size	K = 1	K = 2	K = 3
10	8	0	0
100	20	10	3
1000	90	43	16
10000	719	321DZA	121
100000	5427	3528	1254
1000000	400251	25417	11745
10000000	552010	365432	176543

Solovay Strassen's Test			
Size	K = 1	K = 2	K = 3
10	0	0	0
100	2	0	0
1000	8	0	0
10000	19	0	0
100000	70	4	0
1000000	148	9	0
10000000	427	39	2

Miller Rabin's Test			
Size	K = 1	K = 2	K = 3
10	0	0	0
100	0	0	0
1000	5	0	0
10000	24	0	0
100000	53	2	0
1000000	124	4	0
10000000	290	10	1

There is class of numbers known as Pseudo-Primes which are not really prime but tends to fail on many cases. pseudoprimes are essentially product of primes. Making dataset exclusively of pseudoprimes and testing it against all algorithms. Graph of this is drawn below:



From all this data, we can infer that most of the time for practical application, deterministic algorithm such as miller-rabin gives mostly correct results with value of k > 10. For any practical purposes like cryptography, probabilistic algorithm outweighs pros that of deterministic algorithm. Still if need arises of 100% surety, we can switch to AKS like deterministic algorithm which takes considerable amount of time but provide accurate result always.

Chapter 5

Conclusion

This chapter concludes the work that has been done during this project development process. Main achievement and scope of project are met inferences of which can be seen as:

- Probabilistic and Deterministic algorithms are distinguished based on time and accuracy of algorithms
- It can be seen that Probabilistic algorithm takes significantly less time for primality testing application but may produce false result
- Accuracy measure of algorithm can be increased by running probabilistic algorithms multiple time changing random factor.
- For all practical purposes, Miller-Rabin test with some optimization is used over deterministic AKS or any other test barring there run time complexity and Constant factors involved in them.

Bibliography

[1] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[2] J.Lawrence Carter and Mark N. Wegman. Universal classes of hash functions. *Journal of Computer and System Sciences*, 18(2):143 – 154, 1979.

[3] Mihir Bellare, Shafi Goldwasser, and Daniele Micciancio. “pseudo-random” number generation within cryptographic algorithms: The dds case. In Burton S. Kaliski, editor, *Advances in Cryptology — CRYPTO ’97*, pages 277–291, Berlin, Heidelberg, 1997. Springer Berlin Heidelberg.

[4] Martin Dietzfelbinger. *Primality Testing in Polynomial Time.* Springer Berlin Heidelberg, 2004.

[5] Daniel J. Lehmann. On primality tests. *SIAM Journal on Computing*, 11(2):374–375, may 1982.

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

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

[8] The gnu mp bignum library. https://gmplib.org/.

[9] Github - vivek-2018201078/aps_project_2018: Comparisions of probabilistic and deterministic primality testing algorithms based on accuracy and speed. https://github.com/vivek-2018201078/APS_PROJECT_2018.