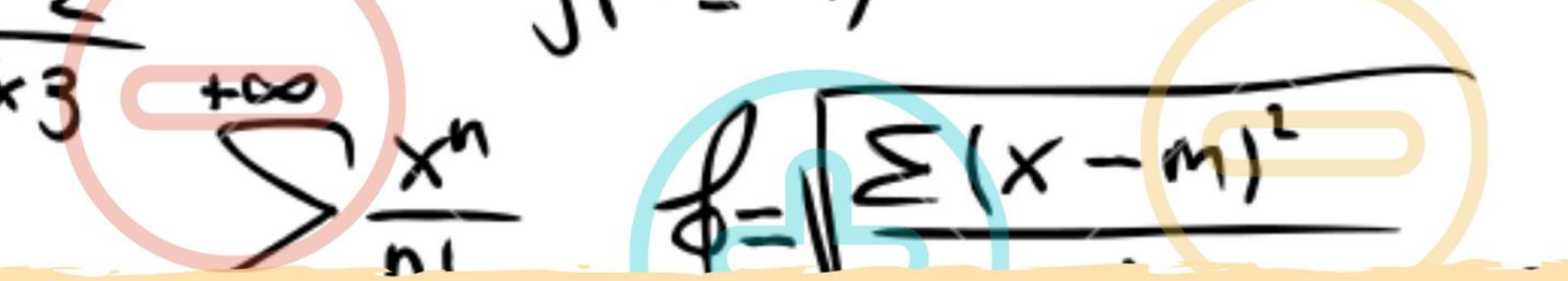
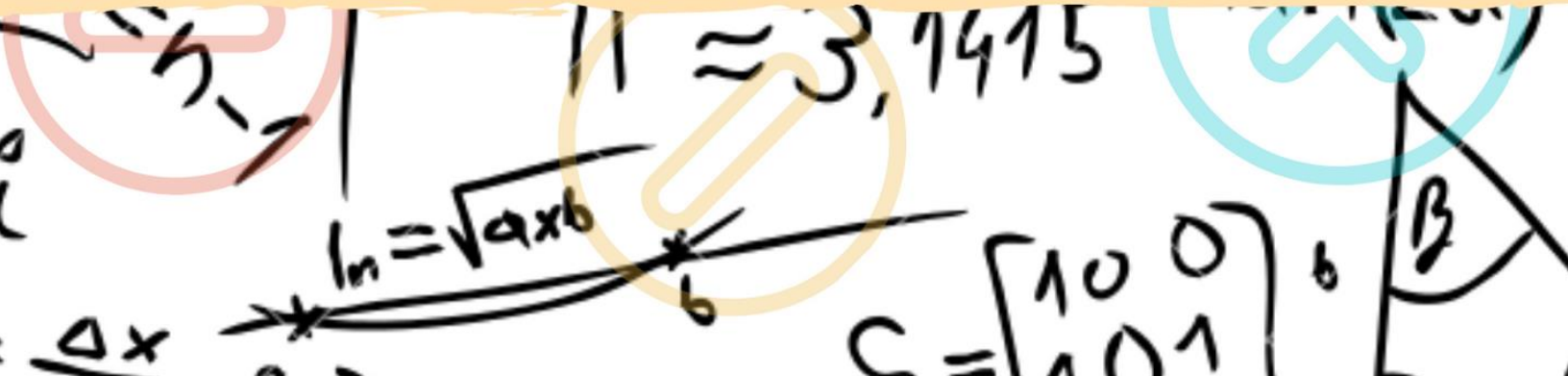


Primality Testing

MATHEMATICS FOR DATA SCIENCE
GROUP PROJECT

Group Epsilon

John Carlo Cueto
Vu Hoang Truong (Vince)
Quang Ngoc Huynh
Minh Thai (Kylie)



Primality Testing

Epsilon Group

John Carlo Cueto
Vu Hoang Truong (Vince)
Quang Ngoc Huynh
Minh Thai (Kylie)

Bachelor of Data Science 2020
S P Jain School of Global Management
Prof. Ashoka Choudhury

December 27, 2020

ACKNOWLEDGEMENTS

This report will not be completed without the assistance of every member in group Epsilon. First and foremost, we give special thanks to our professor Ashoka Choudhury for helping us with guidelines, choosing an interesting topic and assisting in our research. One more time, thank you to all creative team colleagues: Vince, Quang, Kylie and JC Cueto for their inspiration and brilliant ideas.

CONTENTS

Introduction

Prime number

Prime numbers have been studied for thousands of years. Euclid's "*Elements*," published about 300 B.C., proved several results about prime numbers. In Book IX of the "Elements," Euclid writes that there are infinitely many prime numbers. Euclid also provides proof of the Fundamental Theorem of Arithmetic — every integer can be written as a product of primes in a unique way. In "Elements," Euclid solves the problem of how to create a perfect number, which is a positive integer equal to the sum of its positive divisors, using Mersenne primes. A Mersenne prime is a prime number that can be calculated with the equation $2^n - 1$.

In mathematics, prime numbers are whole numbers greater than 1, that have only two factors: 1 and the number itself. Prime numbers are divisible only by the number 1 or itself.

There are dozens of important uses for prime numbers. Cicadas time their life cycles by them, modern screens use them to define color intensities of pixels, and manufacturers use them to get rid of harmonics in their products. However, these uses pale in comparison to the fact that they make up the very basis of modern computational security.

Whatever your thoughts are on prime numbers, you use them every single day and they make up a vital part of our society. All this because they are an irreducible part of the very fabric of the universe.

Primes are of the utmost importance to number theorists because they are the building blocks of whole numbers, and important to the world because their odd mathematical properties make them perfect for our current uses. It is possible that new mathematical strategies or new hardware like quantum computers could lead to quicker prime factorization of large numbers, which would effectively break modern encryption. When researching prime numbers, mathematicians are always being both prosaic and practical.

Primality tests

In 200 B.C., Eratosthenes created an algorithm that calculated prime numbers, known as the Sieve of Eratosthenes. This algorithm is one of the earliest algorithms ever written. Eratosthenes put numbers in a grid, and then crossed out all multiples of numbers until the square root of the largest number in the grid is crossed out.

A primality test is an algorithm, the steps of which verify that given some integer n, we may conclude "n is a prime number." A primality proof is a successful application of a primality test.

Such tests are typically called true primality tests to distinguish them from probabilistic primality tests which can only conclude that "n is prime" up to a specified likelihood. Some primality tests that will be discussed in this text are **Trial Division**, **Wilson's Theorem**, **Fermat Primality Test** and **Miller-Rabin Primality Test**.

Trial division is the most laborious but easiest to understand of the integer factorization algorithms. The essential idea behind **trial division** tests to see if an integer n, the integer to be factored, can be divided by each number in turn that is less than n.

Wilson's theorem states that a natural number $n > 1$ is a prime number if and only if the product of all the positive integers less than n is one less than a multiple of n. That is using the notations of modular arithmetic, the factorial $(n-1)! = 1 \times 2 \times 3 \times ... \times (n-1)$ satisfies $(n-1)! \equiv -1 \pmod{n}$ exactly when n is a prime number. In other words, any number n is a prime number if, and only if, $(n-1)! + 1$ is divisible by n.

Fermat primality test is a primality test, giving a way to test if a number is a prime number, using Fermat's little theorem and modular exponentiation (see modular arithmetic). **Fermat's Little Theorem** states that if a is relatively prime to a prime number p, then $a^{p-1} \equiv 1 \pmod{p}$.

The **Miller–Rabin primality test** is a probabilistic primality test: an algorithm which determines whether a given number is likely to be prime, similar to the **Fermat primality test** and the **Solovay–Strassen primality test**. It is of historical significance for the research of a polynomial-time deterministic primality test. Its probabilistic variant remains widely used in practice, as one of the simplest and fastest tests known.

A concept used frequently in primality testing is the notion of a **sieve**. A "**sieve**" is a process to find numbers with particular characteristics by searching among all integers up to a prescribed bound and eliminating invalid candidates until only the desired numbers remain. **Eratosthenes** proposed the first sieve for finding primes.

Preliminary

Definition P.1: Let $p > 1$ is an integer. p is **prime** if it has only two positive factors: 1 and itself. An integer $n > 1$, which has more than two positive factors is called **composite**

Definition P.2: Let p and q be integers. The notation $k = \gcd(p, q)$ is called the **greatest common divisor** of p and q, which means $k|\mathrm{p}$ and $\mathrm{k}|\mathrm{q}$, where k is the largest positive integer.

Definition P.3: Two integers p and q are **coprime** if and only if $\gcd(p, q) = 1$

Definition P.4: Let $m, n\ and\ p$ be integers. Then m is **congruent** to n modulo p if and only if $m \equiv n \pmod{p}$ or $p \mid \mathrm{m} - \mathrm{n}$.

Definition P.5: An odd composite integer that passes some certain conditions of probabilistic primality test is called a **probable prime**.

Definition P.6 Deterministic is an algorithm that output can be predicted given a particular input, and the machine always conducts the same calculation in the same sequence of states. **Probabilistic** algorithm is opposite of deterministic, which release an output with some probability of being successful.

Definition P.7 The number of time that it takes to execute an algorithm, expressed using notation $O(n), O(nlogn)$, ... is **time complexity**.

Definition P.8 Euler's totient function $\phi(n)$: is a function that counts the number of positive numbers such that $\gcd(a, n) = 1$. For example: $\phi(9) = 6$

Definition P.9 In mathematics, the **factorial** of a positive integer n, denoted by $n!$, is the product of all positive integers less than or equal to n:

$$n! = n \cdot (n-1) \cdot (n-2) \cdot \ldots \cdot 3 \cdot 2 \cdot 1$$

Definition P.10 If a is a natural number less than m, then a number a^{-1} is called a **reciprocal** or **multiplicative inverse** of a modulo m if $aa^{-1} = a^{-1}a = 1 \pmod{m}$.

Theorem: Let m, x be positive integers such that $\gcd(m, x) = 1$. Then x has a multiplicative inverse modulo m, and it is unique (modulo m).

Part 1

Simple methods

Let us get to understand the methods of primality testing by tackling each one of it in depth but in a simple process and getting to know more of its value.

1.1 Trial Division

1.1.1 What is Trial Division

This is one of the simplest deterministic primality tests, by naively checking the condition of a number being prime. It uses the fact that a prime number is not divisible by any other positive integer other than itself and 1, so we can turn it into its contrapositive: if a number is divisible by some other positive integer besides itself and 1, it is composite. However, it is not necessary to check all numbers from 2 to $n - 1$.

1.1.2 Concept

Suppose that n is composite, so $n = pq$ for $2 \leq p, q \leq n - 1$. We claim that at least one of p, q is not greater than $\sqrt{n}$ Indeed, if both are greater than $\sqrt{n}$ then $pq > \sqrt{n} \cdot \sqrt{n} = n,$ a contradiction. Thus, whenever n is composite, one of its factors is not greater than $\sqrt{n}$ so we can modify the range endpoint at *1.1.3 Algorithm (Python Code 1)*

For *1.1.3 Algorithm (Python Code 2)* we loop through i for $\sqrt{n}$ times, so the time complexity is $O(\sqrt{n})$, multiplied the time complexity of division (about as fast as multiplication at around $O(\boldsymbol{lgnlglgn})$, using Newton-Raphson division algorithm). This gives poor performance for large values of n.

1.1.3 Algorithm

Input:

- n: a number that needs testing primality

Output:

- Print True if the number is a prime, otherwise, print False

Python Code 1:

```
def isPrime_1(n):
    if (n < 2):
        return False
    for i in range(2, n):       # from 2 to n-1
        if (n % i == 0):        # n is divisible by i
            return False
    return True

print(isPrime_1(int(input("Enter a number: "))))
```

Python Code 2:

```
def isPrime_2(n):
    if (n < 2):
        return False
    for i in range(2, int(sqrt(n)) + 1):    # from 2 to sqrt(n)
        if (n % i == 0):                    # n is divisible by i
            return False
    return True

print(isPrime_2(int(input("Enter a number: "))))
```

1.2 The Sieve of Eratosthenes

1.2.1 What is Sieve of Eratosthenes

Sieve of Eratosthenes is a simple and ancient mathematical algorithm of finding prime numbers up to any given limit. Its model works by eliminating given numbers which do not meet a certain criterion. In this case, the pattern removes multiples of known prime numbers.

1.2.2 Algorithm

Eratosthenes's method to find all prime numbers less than or equal to n:

- ➢ *Step 1:* Create a list of consecutive integers from 2 to n: $(2, 3, 4, \dots, n)$
- ➢ *Step 2:* Initially, let $i = 2$, which is the smallest prime number.
- ➢ *Step 3:* Mark all multiples of i from i^2 to n as composite (there will be $i^2,\ i^2 + i,\ i^2 + 2i, \dots$)
- ➢ *Step 4:* Find the smallest number in the list that is not marked and assign to i.
- ➢ *Step 5:* Repeat step 3 until $i \leq \sqrt{n}$.

1.2.3 Python Code for Sieve of Eratosthenes

```
from math import *

n = int(input("Enter a number: "))

a = [True for i in range(n+1)]
a[0] = a[1] = False
a[2] = True

for i in range(2, int(sqrt(n))+1):
    if (a[i]):
        for j in range(i*i, n+1, i):
            a[j] = False

print("Prime numbers which is less than", n, ":")
for i in range(n+1):
    if (a[i]):
        print(i, end=", ")
```

1.2.4 Analyze Source Code

First, we create a list containing integers from 2 to n. To take full advantages of the list, we suppose the index is integers and its value is either True or False. True is prime while False indicates composite numbers. We assume all values of the list are True

```
a = [True for i in range(n+1)]
```

Because 0 and 1 are neither prime nor composite, so we assign False for the first two values. Besides, $a[2]$ must be True since 2 is the smallest prime number.

```
a[0] = a[1] = False
a[2] = True
```

Then, we begin the process of marking composite numbers. For this, it iterates all over numbers from 2 to $\lfloor\sqrt{n}\rfloor + 1$. If the current number $\boldsymbol{i}$ is prime, it marks all multiples of $\boldsymbol{i}$ as composite (`a[j] = False`) starting from $\boldsymbol{i^2}$. This is also an optimal implementation as all multiples of $\boldsymbol{i}$ less than $\boldsymbol{i^2}$ have already sifted False earlier by previous primes.

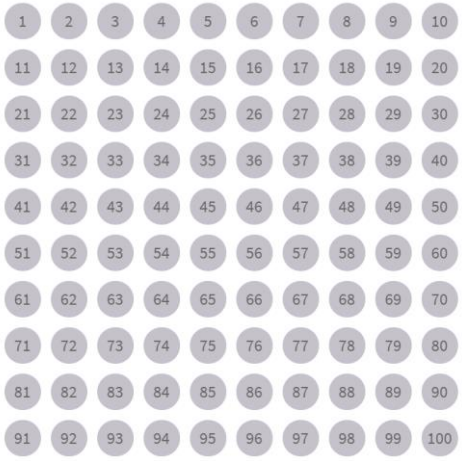
```
for i in range(2, int(sqrt(n))+1):
    if (a[i]):
        for j in range(i*i, n+1, i):
            a[j] = False
```

After the iteration, all indices of composite numbers are marked False, while the rest of the list, which is prime numbers, remain True. Finally, we just print it out from the list.

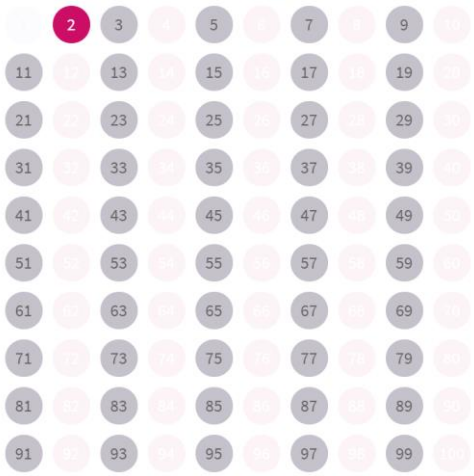
```
for i in range(n+1):
    if (a[i]):
        print(i, end=", ")
```

1.2.5 Example

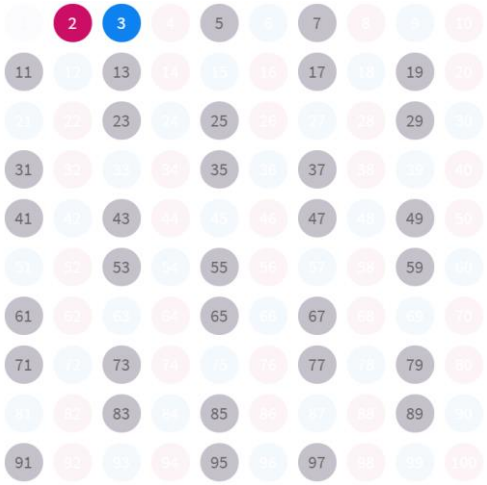
- We apply Eratosthenes's idea from finding prime numbers less than 100.
- Initially, we write 100 numbers on a paper.



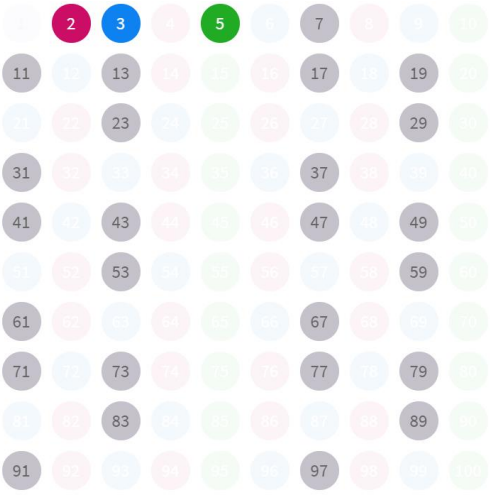
○ Then, we delete 1 as it is not a prime. We already know the smallest prime is 2, thereby removing any multiple of 2 from 4 to 100.



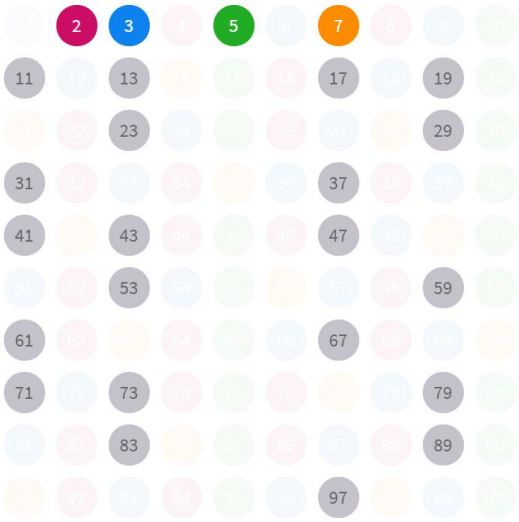
○ The next number is 3, which is not marked. Therefore, we delete all multiples of 3 as well.



○ The next number, 4 is already marked as a composite so we move to 5. It is a prime number and again we cross out all multiples of 5.



○ 7 is the next number after 5. One more, we remove 49 as it is divisible by 7.



○ The cursor goes to 11. Notice, $11^2 = 121$ is greater than 100 so we stop here. Therefore, all remaining numbers on the paper must be prime.

○ There are 25 prime numbers less than 100.

2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97

When we run the code in Python, we also get the same answer.

```
Enter a number: 100
Prime numbers which is less than 100 :
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59,
61, 67, 71, 73, 79, 83, 89, 97,
```

1.3 Wilson's Theorem

1.3.1 What is Wilson's Theorem

Wilson's Theorem states that a natural number n $>$ 1 is a prime number if and only if the product of all positive integers less than n is one less than multiple of n.

To be specific, n is a prime number if and only if

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

1.3.2 Proof

● First, we prove that if $(n-1)! + 1$ is divisible by n, then n is a prime number.

Indeed, when $(n-1)! + 1 = k \cdot n$ or $2 \cdot 3 \cdot \ldots \cdot (n-1) + 1 = k \cdot n$, it means n is coprime with all integers from 1 to $n-1$. Thus, n has no divisors except for 1 and n. Therefore, n is prime.

● For the inverse direction, we must prove if n is a prime, $(n-1)! \equiv -1 \pmod{n}$

○ It is easy to check $n = 2$ satisfy the theorem.

○ We assume $n > 3$. Because n is a prime number, so each of integer less than n is relatively prime to n. In other words, $\gcd(a, n) = 1$ for a running from 0 to $n-1$.

○ As mentioned above, when $\gcd(a, n) = 1$, then a has a multiplicative inverse modulo n. It means that for each of integers a, there is another b less than n such that $ab \equiv 1 \pmod{n}$ apart from 1 and $n-1$ (since the reciprocal of 1 and $n-1$ is the number itself).

○ Now, if we omit 1 and $n-1$, then the others can be grouped into pairs whose product is one showing

$$2 \cdot 3 \cdot 4 \cdot \ldots \cdot (n-2) \equiv 1 \pmod{n}$$

○ Or more simply, $(n-2)! \equiv 1 \pmod{n}$.

○ Finally, we multiply this equality by $n-1$ to complete the proof.

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

● The proof above is also the algorithm for the code of Wilson's Theorem.

1.3.3 Python Code for Wilson's Theorem

```
def Wilson(n):
    r = 1
    for i in range(1, n):
        r = (r*i) % n

    if (r + 1 == n):
        return True
    else:
        return False

n = int(input("Enter a number: "))

if (Wilson(n)):
    print(n, "is a prime number")
else:
    print(n, "is a composite number")
```

When we run the code and enter 997 as input, we got the result:

```
Enter a number: 997
997 is a prime number
```

1.3.4 Flaws

While Wilson's Theorem is beautiful in many ways, however, one downside is that it takes much time to determine whether one integer is a prime or not, in particular, $n-1$ iterations will be executed. The larger n is, the more time the code will take.

1.4 Graphs

1.4.1 Running Time for Trial Division

By using Python to calculate the running time of Trial Division, we got the table below:

Test case	7	97	997	9973	99991	999983	9999991	99999989	999999937	9999999967	99999999977
Running Time	0	0	0	0	0.000987	0.000999	0.001	0.002	0.004	0.016	0.044

Figure 1.1 *Table of running time for Trail Division*

We also plot the data on a graph for analyzing purpose

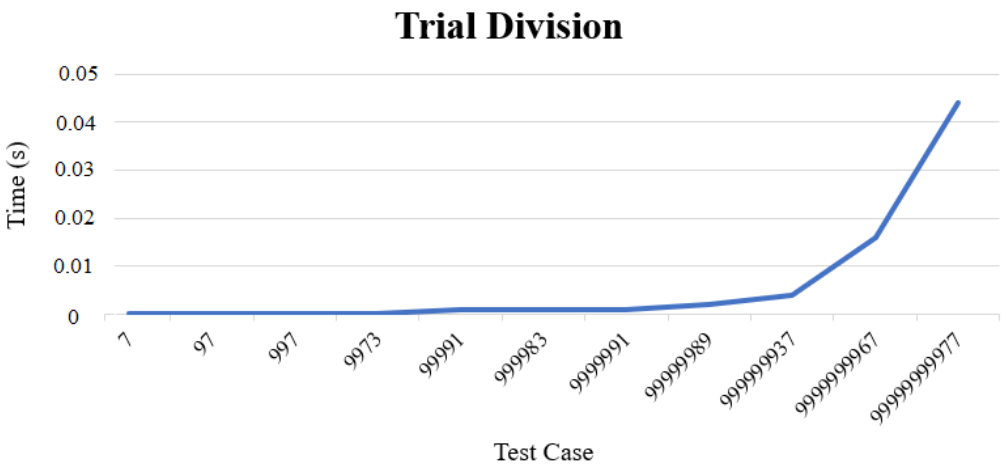


Figure 1.2 *Line graph of running time for Trail Division*

The time complexity for Trial Division is $O(\sqrt{n})$.

Looking at the line graph, if the input number is less than 10000, the running time is negligible. Its pattern experiences a significant growth when the number is increasing. However, the running time is still fast as it takes only 0.044s to check whether an 11-digit number is prime or not. This is why Trial Division is considered as an efficient way to determine prime numbers.

1.4.2 Running Time for Wilson's Theorem

Wilson's Theorem is suitable for modular arithmetic but it takes much time to identify a prime number.

Test Case	7	97	997	9973	99991	999983	9999991	99999989	999999937	9999999967
Running Time	0	0	0.000997	0.00199	0.0169	0.1226	1.293	15	140	5235

Figure 1.3 *Table of running time for Wilson's Theorem*

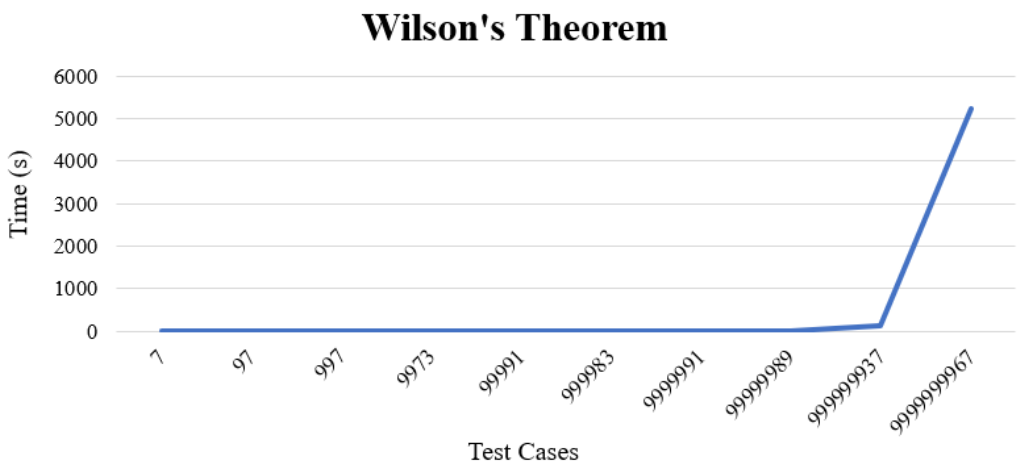


Figure 1.4 *Line graph of running time for Wilson's Theorem*

The time complexity for Wilson's Theorem is $O(n)$.

Unlike Trial Division, the running time for Wilson's Theorem is greater. It takes around 1 second to execute 6 and 7-digits while that of 8 and 9-digits are 15 and 140 seconds respectively. This figure for over 1 billion numbers is 37 times higher, which takes more than 1.5 hours to print out the result.

1.4.3 Number of Primes

The table below illustrates how many numbers of prime in specific range by using Eratosthenes's method,

Range	1-digit	2-digit	3-digit	4-digit	5-digit	6-digit	7-digit	8-digit
Number of primes	4	21	143	1061	8363	68906	586081	5096876

Figure 1.5 *Number of primes in particular range*

In reality, humans use powerful computer to identify prime numbers. By 2017, the largest prime has been found is $2^{77,232,917} - 1$, having 23,249,425 digits.

Part 2

Probabilistic Tests

Definition. Probabilistic tests. Probabilistic tests attempt to determine the primality of a number n by performing a check of some equality involving a random number a. The usual probabilistic tests report compositeness with absolute accuracy and primality with high accuracy. Reduction of probability of error can be achieved by repeating the tests with several different values of a.

2.1 Fermat Test

Fermat test is the simplest among probabilistic tests.

2.1.1 Concept

Fermat test is the simplest among probabilistic tests and is based on Fermat's little theorem.

Theorem. Fermat's little theorem. If n is a prime number and a is any integer,

$$a^{n-1} \equiv 1 \pmod{n}$$

Proof by induction:

$$a^{n-1} \equiv 1 \pmod{n}$$
$$\Rightarrow a^n \equiv a \pmod{n}$$

- Induction hypothesis:
  - n is a prime number
  - a is any integer in the range $[2, p-2]$
  - $a^n \equiv a \pmod{n}$

- Base case: $1^n = 1 \pmod{n}$ is obviously True.

- Inductive step:
  - Suppose $a^n \equiv a \pmod{n}$ is True. Then, by the binomial theorem,

$$(a+1)^n = a^n + \binom{n}{1} a^{n-1} + \cdots + \binom{n}{n-1} a + 1$$

  - Taken mod n, we are then left with $(a+1)^n \equiv a^n + 1 \pmod{n}$.

Therefore, to test whether n is prime, one can pick any integer a indivisible by n to see if the congruence above is True. If it is, then n is a probable prime. Consult the table below for further information:

Congruence	*n is actually*	*n is then called*	*a is then called*
True	composite	a Fermat pseudoprime to base a	a Fermat liar
False	composite	still a composite number	a Fermat witness for the compositeness of n

Figure 2.1 *Fermat liar and witness*

For a more optimum way of selecting a, it will be randomly chosen in the interval $[2, p-2]$ (removing 1 and $p-1$), for

- $1^{n-1} \equiv 1 \pmod{n}$ holds trivially,
- If n is odd and $a \equiv -1 \pmod{n}$, the congruence also holds trivially.

2.1.2 Example

$$n = 15;\ a = 4$$
$$a^{n-1} = 4^{14} \equiv 1 \pmod{15}$$

With this result, either 15 is a prime or 4 is a Fermat liar.

To make sure, we choose another a, say 6: $a^{n-1} = 6^{14} \not\equiv 1 \pmod{15}$

So we can now conclude:

- 15 is composite
- 4 is a Fermat liar
- 6 is a Fermat witness for the compositeness of 15

2.1.3 Algorithm

Input:

- n: a number that needs testing primality
- k: number of iterations. The higher it is, the more accurate the conclusion is.

Output:

- n is composite or probably prime

Repeat k times:

- Randomly pick a value for a in the range $[2, n-2]$
- If $a^{n-1} \not\equiv 1\ (mod\ n)$, return composite and end the code
- If the above never happens, then return probably prime

The specific code for Python is as follow:

```
from random import *
from math import *

def get_coprime(n):
    while True:
        coprime = randrange(2, n-1)
        if (gcd(coprime, n) == 1):
            return coprime
```

```
def Fermat(n, r):
    for i in range(r):
        a = get_coprime(n)
        if ((a ** (n-1)) % n != 1):
            return False
    return True

n = int(input('Enter a number to test: '))
r = int(input('Enter a range: '))

if Fermat(n, r):
    print(n, 'is probably prime')
else:
    print(n, 'is composite')
```

2.1.4 Time complexity

Using fast algorithms for modular exponentiation and multiprecision multiplication, the running time of this algorithm is $O(k \log^2 n \ \log\log n) = \tilde{O}(k \log^2 n)$, where k is the number of times we test a random a, and n is the value we want to test for primality.

2.1.5 Flaws

First of all, there is an infinite number of Fermat pseudoprimes. A more serious flaw is due to the existence of what is called the Carmichael numbers.

Consider $n = 561$, check its primality. Applying Fermat's Little Theorem, we choose $a = 20$, we have: $\gcd(20, 561) = 1$

$$20^{560} \equiv 1 \pmod{561}$$

We conclude 561 is a probable prime number. However, 561 is actually composite and can be factorized as $561 = 3 \cdot 11 \cdot 17$. Then, we call 561 a Fermat pseudoprime and 20 a Fermat liar. In fact, when $n = 561$, all a's are Fermat liars no matter what value is assigned to it. Numbers like 561 have a special name: Carmichael numbers.

Definition. Carmichael number. A Carmichael number n is an odd composite number which passes the Fermat primality test. In other words, it satisfies the modular congruence:

$$a^{n-1} \equiv 1 (mod\ n) \ \text{ where } \ \gcd(a, n) = 1 \ \text{ for all } \ a \ \text{ with } \ 1 \leq a \leq n - 1$$

561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, … are some of the first Carmichael numbers.

In fact, there is an alternative way to define the Carmichael Number raised by Korselt. Following this criterion, a Carmichael number n has to satisfy either of two conditions:

i) n is a square-free, which means for p is a prime factor of n then $n \equiv 0 \pmod{p}$ but $n \not\equiv 0 \pmod{p^2}$.

ii) p is a prime factor of n then $n - 1 \equiv 0 \pmod{p - 1}$.

We take $41041 = 7 \cdot 11 \cdot 13 \cdot 41$ as an example, we easily witness that 41041 is square-free and 41040 is divisible by 6, 10, 12 and 40. Then we can conclude 41041 is a Carmichael number.

Based on these definitions, many mathematics have put forward some properties of this sequence. Carmichael numbers have at least three prime factors. Moreover, for those which have exactly three prime factors and be written as the form $(6k + 1) \cdot (12k + 1) \cdot (18k + 1)$ should be a Carmichael number. For instance: we take $1729 = 7 \cdot 13 \cdot 19 = (6k + 1) \cdot (12k + 1) \cdot (18k + 1)$ with $k = 1$. Then, we conclude 1729 is a Carmichael number. Indeed, by simple calculation, we see that:

$$a^{1728} \equiv 1 \pmod{1729} \text{ with } \gcd(a, 1729) = 1$$

While Carmichael numbers are very rare, even more so than prime numbers, they are not rare enough for the Fermat test to be implemented often, at least in the original form. Instead, one of the more powerful extensions of it – the **Miller-Rabin test**, is more commonly used.

2.2 Miller-Rabin Test

This probabilistic test was discovered by Gary L. Miller in 1976. This version is deterministic, yet the correctness of the test depends on extended Riemann hypothesis (ERH). However, its probability of success is greater than that of the Fermat test.

2.2.1 Miller-Rabin Standard

Let p be an odd prime number, we write:

$$p - 1 = 2^s \cdot m \text{ where } s \in N \text{ and } m \text{ is odd}$$

For any number $a \in \{1, 2, \dots, p - 1\}$, then the sequence

$$a^m \pmod{p}, a^{2m} \pmod{p}, a^{2^2 m} \pmod{p}, \dots, a^{2^s \cdot m} \pmod{p}$$

has one of two conditions:

i) $a^m \equiv 1 \pmod{p}$.

ii) $a^{2^k \cdot m} \equiv -1 \pmod{p}$ for some $k \in (0,1, \dots, s - 1)$.

2.2.2 Proof

We have p is an odd prime then $p - 1$ is even, so we can perform

$$p - 1 = 2^s \cdot m$$

Suppose we have a recursive sequence:

$$x_k = a^{2^k \cdot m} \text{ for } k = 0, 1, 2, \dots, s$$

Based on Fermat's Little Theorem:

$$\begin{aligned} & a^{p-1} \equiv 1 \pmod{p} \\ & \Rightarrow a^{2^s \cdot m} \equiv 1 \pmod{p} \\ & \Rightarrow a^{2^s \cdot m} - 1 \equiv 0 \pmod{p} \\ & \Rightarrow (a^{2^{s-1} \cdot m} - 1) \cdot (a^{2^{s-1} \cdot m} + 1) \equiv 0 \pmod{p} \\ & (x_{s-1} - 1) \cdot (x_{s-1} + 1) \equiv 0 \pmod{p} \end{aligned}$$

This means either $x_{s-1} \equiv 1 \pmod{p}$ or $x_{s-1} \equiv -1 \pmod{p}$. If $x_{s-1} \equiv -1 \pmod{p}$, we stop the algorithm, otherwise we continue the Fermat's Little Theorem to find the congruence of x_{s-2} modulo p. Then, after finite steps, we have either of results:

i) $x_k \equiv 1 \pmod{p}$ for $k = 0$.

ii) $x_k \equiv -1 \pmod{p}$ for some k that $0 \leq k \leq s-1$.

If both conditions fail, we then conclude that p is a composite number. Nonetheless, we cannot guarantee p is prime, then we should check for some values of a to increase the certainty.

2.2.3 Algorithms

Input:

- n: an integer greater than or equal to 2.
- k: number of iterations.

Output:

- True if n is prime, False otherwise.

We follow 10 steps below for Miller-Rabin Test:

1. If $n = 2$ return True
2. If n is even, return False
3. Perform $n - 1 = 2^s \cdot m$ with $s \in N$ and m is an odd number.
4. For $i = 0$ to k:
5. Randomly choose a from 1 to $n - 1$, we compute $b = a^m \pmod{n}$
6. If $b \equiv 1 \pmod{n}$ return True. End
7. For $j = 0$ to s:
8. If $b \equiv -1 \pmod{n}$, return True. End
9. Set $b = b^2 \pmod{n}$
10. Return False. End.

Now, we use these algorithms in a Python code:

```
from random import *

def Miller_Rabin(n, k):
    if (n == 2):
        return True
    elif (n % 2 == 0):
        return False

    # write n-1 in form of 2 power s and multiply m
    s, m = 0, n - 1
    while (m % 2 == 0):
        s += 1
        m //= 2
```

```
    # we run the algorithms in k trials
    for i in range(k):
        a = randint(2, n - 1)
        b = (a ** m) % n

        if (b % n == 1):
            return True

        for j in range(s):
            if (b % n == n - 1):
                return True
            b = (b ** 2) % n

    return False

n = int(input("Enter an integer greater than 2: "))
k = int(input("Enter number of iterations: "))
print(Miller_Rabin(n, k))
```

2.2.4 Time complexity

Since this exponentiation has the dominant complexity in the above algorithm, we conclude that the overall time-complexity of the Miller-Rabin algorithm is $O(k \times log\ 3n)$, k being the number of bases we test out, and hence it is indeed polynomial in the bit-size of n.

2.2.5 Probability of success

2.2.5.1 Solving problems by Miller-Rabin Test

Example 1: $n = 233$

- First, we make subtraction: $n - 1 = 233 - 1 = 232$
- Then we write in the form: $n - 1 = 2^s \cdot m \implies 232 = 2^3 \cdot 29$
- We use computer to randomly choose a from 1 to 232, suppose $a = 7$, then:

$$7^{29} \equiv 89 \pmod{233}$$

- Then we take the congruence of $7^{29 \cdot 2} \pmod{233}$, we have:

$$7^{29 \cdot 2} \equiv 89^2 \equiv -1 \pmod{233}$$

- Then we can stop the algorithms and conclude that 233 may be a prime number.

Example 2: $n = 789$

- First, we make subtraction: $n - 1 = 789 - 1 = 788$
- Then we write in the form: $n - 1 = 2^s \cdot m \implies 788 = 2^2 \cdot 197$
- We use computer to randomly choose a from 1 to 232, suppose $a = 53$, then:

$$53^{197} \equiv 353 \pmod{789}$$

○ Then we take the congruence of $53^{197\cdot 2} \pmod{789}$, we have:

$$53^{197\cdot 2} \equiv 353^2 \equiv 736 \pmod{789}$$

○ Then we take the congruence of $53^{197\cdot 2^2} \pmod{789}$, we have:

$$53^{197\cdot 2^2} \equiv 442 \pmod{789}$$

○ Then 789 is a composite number.

2.2.5.2 Failure percentage of Miller Rabin Test

Definition 4.5.1: Let n be an odd composite number, and we have $n-1 = 2^s \cdot m$, with m is odd, and let a is an integer from 0 to $n-1$ and $\gcd(a,n) = 1$. If $a^r \not\equiv 1 \pmod{n}$ and $a^{2^k \cdot m} \not\equiv -1 \pmod{n}$ $\forall k \in (0,1,\dots,s)$, a is called a strong witness for n.

Definition 4.5.2: Let n be an odd composite number, and we have $n-1 = 2^s \cdot m$, with m is odd, and let a is an integer from 0 to $n-1$ and $\gcd(a,n) = 1$. If $a^r \equiv 1 \pmod{n}$ or $a^{2^k \cdot m} \equiv -1 \pmod{n}$ $\forall k \in (0,1,\dots,s)$, a is called a strong liar for n.

Theorem: If n is an odd composite number, the number of strong liars k is such that $k \leq \frac{\phi(n)}{4}$, where $\phi(n)$ is Euler totient function for n.

Because n is composite, we always have $\phi(n) < n-1$, which means that n can pass the test for at most $\frac{n-1}{4}$ values of a $that$ $(1 \leq a \leq n-1)$ and $\gcd(a,n) = 1$. As a result, the probability of a that a is a strong liar is less than $\frac{1}{4}$. If we run the test k times with k different values of a, then the failure probability is $\frac{1}{4^k}$, which is less than that of the Fermat Test. So, we find that Miller Rabin Test can reduce the number of pseudoprimes compared with Fermat Test.

For instance, if we run the algorithm 50 times, then the probability failure is less than 4^{-50}, which is around 10^{-31}.

We take $n = 247$ as an example, write $n-1 = 247-1 = 246 = 2^1 \cdot 123$ $(s = 1, m = 123)$

Let us choose $a = 68$ then $\gcd(68,247) = 1$, we have $68^{123} \equiv 1 \pmod{247}$,

Let us choose $a = 75$ then $\gcd(75,257) = 1$, we have $75^{123} \equiv -1 \pmod{247}$,

Based on two results, we may think 247 is a prime number; however, if

Let us choose $a = 105$ then $\gcd(105,247) = 1$, we have $105^{123} \equiv 27 \pmod{247}$,

we may conclude that 247 is composite. Indeed, $247 = 13 \cdot 19$ then 68 and 75 are strong liars of 247 and 105 is a strong witness for it. If we want to calculate the strong liars of 247, we can see that is the set

$$S = (1, 12, 56, 68, 69, 75, 87, 88, 103, 144, 159, 160, 172, 178, 179, 191, 235, 246)$$

Then $|S| = 18 < \frac{n-1}{4} = 61.5$. From the set above, we also note that if a is a strong liar of composite n, then $n-a$ will be too. Hence, we can skip the test for the base $n-a$.

We have known that Miller-Rabin is a probabilistic test; however, we can get a deterministic version by utilizing Generalized Riemann Hypothesis as stated in the introduction of Miller-Rabin Test. But it is still unproven. The version is that we should test only values of $a < 2\ln^2 n$. If we assume that it is true, this puts the algorithm at a time complexity of $O(\log^4 n)$.

However, some mathematicians have put a theory that when n is small, applying the formula $a < 2\ln^2 n$ is redundant. For instance: if $n < 2{,}047$, only test $a = 2$. We generalize some values of n in the table below:

n for which $n < k$	a values
2,047	2
1,373,653	2, 3
9,080,191	31, 73
25,326,001	2, 3, 5
3,215,031,751	2, 3, 5, 7
4,759,123,141	2, 7, 61
1,122,004,669,633	2, 13, 23, 1662803
2,152,302,898,747	2, 3, 5, 7, 11
341,550,071,728,321	2, 3, 5, 7, 11, 13, 17

Figure 2.2 *a values for specific range of n*

REFERENCE

Mathigon.org, (2020). Mathigon Website. [online] Available at https://vi.mathigon.org/course/divisibility/primes [Accessed 27 Dec. 2020]

Satish R. and David T. (2009). *CS70, Discrete Mathematics and Probability Theory*. 1st ed. [pdf]. Available at http://stanford.edu/~dntse/classes/cs70_fall09/cs70_fall09_5.pdf [Accessed 27 Dec. 2020].

Primes.utm.edu, (2020). The PrimePages Website. [online] Available at https://primes.utm.edu/notes/proofs/Wilsons.html [Accessed 27 Dec. 2020].

Elaine J. Hom (2013). *What is a Prime Number?*. [online] Available at https://www.livescience.com/34526-prime-numbers.html [Accessed 27 Dec. 2020]

Ivan Koswara, et al (no date). *Primality Testing. Briliant.org.* [online] Available at https://brilliant.org/wiki/prime-testing/ [Accessed 27 Dec. 2020]

Richard A. Mollin (2014). *A Brief History of Factoring and Primality Testing B. C.* [pdf] Available at https://sci-hub.se/https://www.jstor.org/stable/3219180?seq=1 [Accessed 27 Dec. 2020]

Bandyopadhyay, S. (2009). *PRIMALITY TESTING A Journey from Fermat to AKS.* Chennai Mathematical Institute, Mathematics and Computer Science. [pdf] Available at https://www.cmi.ac.in/~shreejit/primality.pdf [Accessed 27 Dec. 2020]

Carl Promerance, J. S. (1980). *Mathematics of computation* (Vol. 35). Available at https://math.dartmouth.edu/~carlp/PDF/paper25.pdf [Accessed 27 Dec. 2020]

Worthington, R. (2018). *Primality Testing Theory, Complexity, and Applications.* Whitman College. [pdf] Available at https://www.whitman.edu/documents/Academics/Mathematics/2018/Worthington.pdf [Accessed 27 Dec. 2020]