الجامعة المصرية اليابانية للعلوم و التكنولوجيا
エジプト日本科学技術大学
EGYPT-JAPAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

Algorithms Course Project

Primality Testing using Miller-Rabin

Submitted by:

Ahmad Mongy (ID 120200033)
Ahmed Abdelkader (ID 120200028)
Mahmoud Akrm (ID 120200045)
Mohammed Ayman (ID 120200081)
Mohanned Ahmed (ID 120200113)
Peter Fayez (ID 120200073)
Yasser Ossama (ID 120190122)
Ziad Hesham (ID 120200078)

Course Instructor:

Dr. Walid Gomaa

Contents

1 Introduction

Prime numbers have fascinated mathematicians for centuries and have played a crucial role in modern cryptography. However, determining whether a given number is prime or composite can be a computationally challenging problem, particularly for large numbers. Over the years, numerous algorithms have been developed to efficiently test the primality of numbers, ranging from ancient methods to the most cutting-edge techniques of today.

Some of the oldest primality testing algorithms include the trial division and the Sieve of Eratosthenes. These methods involve dividing the number by all smaller integers to determine whether it is prime. While these algorithms are simple to understand and implement, they become impractical for very large numbers as the running time grows exponentially.

More sophisticated algorithms, such as the Miller-Rabin and the AKS algorithm, have been developed to overcome this limitation. The Miller-Rabin algorithm is a probabilistic algorithm that performs a series of modular exponentiations to determine whether a number is likely to be prime. On the other hand, the AKS algorithm is a deterministic polynomial-time algorithm that can test the primality of any number. However, it is not used in practice because its runtime is long.

In recent years, researchers have developed even faster algorithms for primality testing, such as the elliptic curve primality proving algorithm and the number field sieve. These algorithms have significantly reduced the time required to test the primality of large numbers.

In this report, we will explore the various primality testing algorithms, from the ancient to the modern, and examine their running times for different sizes of numbers. We will also discuss the advantages and limitations of each algorithm and compare their performances to help identify the best algorithm for a given situation.

2 Primality Testing Algorithms

2.1 Deterministic Algorithms

2.1.1 Naive Approach

The naive algorithm involves dividing the number by all smaller integers to determine whether it is prime or composite.

```
def test_primality(n):
    for i in range(2,n):
        if n % i == 0:
            return False
    return True
```

Total Complexity: $\mathcal{O}(n)$

Comments: While this method is straightforward, it becomes impractical for very large numbers as the running time grows exponentially.

2.1.2 Trial Division

The naive approach could be made more efficient by observing that the maximum factor for any number n is always less than or equal to the $\sqrt{n}$

```
def trial_division(n):
    for i in range(2, sqrt(n) + 1):
        if n % i == 0:
            return False
    return True
```

Total Complexity: $\mathcal{O}(\sqrt{n})$

Comments: This method reduces the run-time of the naive algorithm. Still, this algorithm is exponential in terms of number size (in binary representation). Also, we often assume that division and modulo operations are constant time operations. However, this is not always the case, especially with large numbers.

2.1.3 Sieve of Eratosthenes

This algorithm is used mainly to generate all the primes $\leq n$. The algorithm first filters all even numbers less than n and marks their smallest prime factor (SPF) as 2. The algorithm does so for 3 and all the larger numbers. Therefore, the remaining integers are guaranteed to be primes.

```
def sieve_of_eratosthenes(n):
    prime = [True for i in range(n+1)]
    p = 2
    while p * p <= n:
        if prime[p]:
            for i in range(p * p, n + 1, p):
                prime[i] = False
        p += 1
```

Total Complexity: The complexity analysis is done by observing that the inner loop is only executed when the number p is prime. Therefore, an upper-bound for the complexity is $\sum_{\substack{p \text{ prime} \\ p \leq n}} \frac{1}{p} = \mathcal{O}(n \log(\log n))$

Comments: This algorithm is not used mainly for primality testing.

2.1.4 Foolproof primality test

The main idea of the this algorithm to test the primality of a number n is to expand the polynomial $(x-a)^n - (x^n - a)$ and see if all of its coefficients are divisible by n, where a is coprime to n (chosen as 1 for small integers). That is, we check the congruence relation $(X + a)^n \equiv X^n + a$. To give an example, consider the number 3. By expanding the given polynomial, we get a second degree polynomial as follows:

$$x^3 - 3x^2 + 3x - 1 - x^3 + 1 = -(3)x^2 + (3)x$$

where the coefficients are parenthesized. As both coefficients are divisible by 3, we conclude that 3 is in fact a prime number. However, in case of 4, the polynomial expansion goes as follows:

$$x^4 - 4x^3 + 6x^2 - 4x + 1 - x^4 + 1 = -(4)x^3 + (6)x^2 - (4)x + 2$$

As one of the coefficients is not divisible by 4, we conclude that 4 is in fact a composite integer.

Total Complexity: $\mathcal{O}(n^2)$

Comments: This algorithm gets very computationally heavy for big numbers.

2.1.5 Agrawal–Kayal–Saxena test (AKS)

The first deterministic algorithm to have a polynomial upperbound in terms of its size in bits. Also, it has four features that are not found grouped in any other algorithm, which are:

- **Unconditionally correct:** Its correctness is not dependent on any side unproven hypothesis. However, algorithms like the deterministic version of Miller-Rabin (See 2.2.2) depend on some unproven hypothesis (the generalized Riemann Hypothesis).
- **General:** It can work with any general number. Certain algorithms (e.g. Lucas-Lehmer test [3.2.1]) assume numbers with some special properties.
- **Polynomial:** Its running time is polynomial in terms of input bit size.
- **Deterministic:** It is guaranteed to give the correct answer for both primes and composites.

The algorithm is as follows:

1- If ($n = a^b$ for $a \in \mathbb{N}$ and $b > 1$): print **COMPOSITE**.

2- Find the smallest r such that $\text{ord}_r(n) > \log^2(n)$.

3- $\forall a, 2 \leq a \leq \min(r, n-1)$, if $(a \mid n)$, print **COMPOSITE**.

4- If $(n \leq r)$ print **PRIME**.

5- For $a = 1$ to $\left\lfloor \sqrt{\varphi(r)}\log_2(n) \right\rfloor$ do:

if $(X + a)^n \neq X^n + a \pmod{X^r - 1,\ n}$, print **COMPOSITE**.

6- print **PRIME**.

Notes:

- $\mathbf{ord_r(n)}$ is the multiplicative order of n mod r.

- $\varphi(\mathbf{r})$ is the euler totient function of r.

- The bottleneck of the algorithm is step 5, where it was done in exponential time in the Foolproof algorithm (See 2.1.4).

Total Complexity: $\mathcal{O}(d^{12})$, where d is the number of digits of the input number.

Comments: This algorithm gets very computationally heavy for big numbers, that is why it is never used in practice and is considered a galactic algorithm.

2.2 Probabilistic Algorithms

2.2.1 Fermat Primality Test

This algorithm is based upon Fermat's little theorem which states that that for a prime number p and a coprime integer a the following equation holds:

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

In general, this equivalence doesn't hold for composite numbers. This can be used to create a primality test. We pick an integer $2 \leq a \leq p-2$, and check if the equation holds or not. If it doesn't hold (i.e. $a^{p-1} \not\equiv 1 \bmod p$), we know that p cannot be a prime. In this case, we call the base a a Fermat witness for the compositeness of p. However it is also possible, that the equation holds for a composite number. So, if the equation holds, we don't have a proof for primality. We can only say that p is probably prime. If it turns out that the number is actually composite, we call the base a a Fermat liar.
By running the test for all possible bases a, we can actually prove that a number is prime. However this is not done in practice, because this is a lot more computation than doing Trial Division (See 2.1.2). Instead, the test will be repeated multiple times with random choices for a. If we find no witness for the compositeness, it is very likely that the number is in fact a prime.

```
    def Fermat_Test(n, k):
        # Higher values of k indicates higher
          probability of correct results for
          composite integers
```

```
        if n == 1 or n == 4:
            return False
        elif n == 2 or n == 3:
            return True
        else:
            for i in range(k):
                a = randint(2, n-2)
                if pow(a, n-1, n) != 1:
                    return False
            return True
```

Total Complexity: $\mathcal{O}(k \log n)$

Comments: Please note that this complexity is unachievable without the use of fast modular exponentiation to calculate $a^{n-1} \pmod{n}$ which runs in $\mathcal{O}(\log n)$ time. Also, as stated before, note that the above method may fail even if we increase the number of iterations (higher k). Fermat's primality test is often used if a rapid method is needed for filtering, for example in the key generation phase of the RSA public key cryptographic algorithm.

2.2.2 Miller-Rabin Primality Test

The Miller-Rabin test extends the same idea from the Fermat test. For an odd number n, $n-1$ is even and we can factor out all powers of 2 as follows:
$n - 1 = 2^s \cdot d$, with d being an odd number.
This consequently allows us to factorize the equation of Fermat's little theorem:

$$\begin{aligned} a^{n-1} \equiv 1 \bmod n &\iff a^{2^s d} - 1 \equiv 0 \bmod n \\ &\iff (a^{2^{s-1}d} + 1)(a^{2^{s-1}d} - 1) \equiv 0 \bmod n \\ &\iff (a^{2^{s-1}d} + 1)(a^{2^{s-2}d} + 1)(a^{2^{s-2}d} - 1) \equiv 0 \bmod n \\ &\vdots \\ &\iff (a^{2^{s-1}d} + 1)(a^{2^{s-2}d} + 1) \cdots (a^{d} + 1)(a^{d} - 1) \equiv 0 \bmod n \end{aligned}$$

Therefore, if n is prime, n has to divide one of these factors. In the Miller-Rabin primality test, for a base $2 \leq a \leq n-2$ we check if either

$a^d \equiv 1 (\text{mod } n)$ or $a^{2^r \cdot d} \equiv -1 (\text{mod } n)$ holds for some $0 \leq r \leq s-1$.
If we found a base a which doesn't satisfy any of the above equalities, then we found a witness for the compositeness of n. In this case we have proven that n is not a prime number.

Total Complexity: $\mathcal{O}(k \log n)$

Deterministic Version
Miller proved that possible to make the algorithm deterministic by only checking all bases $a \leq \mathcal{O}(\ln x^2)$. It was later proved a concrete bound of $2\ln(x)^2$. In [4], it was proved that for testing numbers up to $3 \cdot 10^9$ it is only necessary to check the first 4 prime bases: {2, 3, 5, 7}. The smallest composite number that fails this test is $3,215,031,751 = 151 \cdot 751 \cdot 28351$. Moreover, for testing numbers up to 10^{19} it is enough to check the first 12 prime bases {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, and 37}.

3 Prime properties

3.1 Twin Primes

Twin primes are a pair of primes that has a prime gap of two. For example, both (17, 19) and (41, 43) are twin prime pairs.
First few twin prime pairs are: (3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73), (101, 103).

Note that 5 is the only prime that belongs to two pairs, as all of the following pairs are of the form $(6n-1)$ and $(6n+1)$, i.e. their sum is divisible by 12.

3.1.1 Current Research

Twin Prime Conjecture
This is a famous open question in mathematics. It remains unproven whether that there exist an infinite number of twin primes.

De Polignac's Cojecture
This is the more general form of the twin prime conjecture. It asks if for every natural k, there exists a pair of primes with a prime gap of $2k$. The case with $k = 1$ is the twin prime conjecture.

3.1.2 Recent work

There was a recent paper proving that for some integer $x < 7 \cdot 10^6$, there are an infinite number of primes that differ by x [1]. This bound was later reduced to 246 [2].

3.2 Mersenne Primes

A mersenne prime is a prime p of the form $2^x - 1$ for some positive integer x. Notably, if x is a composite integer, so is $2^x - 1$. Also, it is easy to prove that if $2^p - 1$ is prime, then p is prime.
Mersenne primes studied because of their relation to perfect numbers [1]. Euclid proved that if p is a Mersenne prime, then $(2^p - 1)^2$ is a perfect number. Also, a converse statement was proved by Leonhard Euler, that is, all even perfect numbers have that form.

3.2.1 Lucas-Lehmer Test

Verifying Mersenne primes can be done efficiently using the Lucas-Lehmer test LL1. The algorithm verifies the Mersenne prime $M_x = 2^x - 1$. The algorithm starts be defining a sequence of integers S_i recursively as follows:

$$S_i = \begin{cases} 4 & \text{if } i = 0; \\ S_{i-1}^2 - 2 & \text{otherwise.} \end{cases}$$

Then M_x is prime iff. $S_{x-2} \equiv 0 \pmod{M_x}$

```
    def Lucas_Lehmer(x):
        s = 4
        M = pow(2,x) - 1
        for i in range(x - 2):
            s = (s * s - 2) % M
        if s == 0:
            return True
        else:
            return False
```

[1] Perfect numbers are positive integers that can be represented as the sum of their divisors (e.g. $6 = 1 + 2 + 3$)

3.3 Sophie-Germain primes

A prime number p is a Sophie-Germain prime if $2p - 1$ is also a prime. Sophie-Germain primes have a wide-range of applications in cryptography and pseudo-random numbers generation.

3.4 Pythagorean primes

A prime p of the form $4x + 1$ is called a pythagorean prime for a positive integer x. A pythagorean prime p appear as the hypotenuse in two pythagorean triplets as p and $\sqrt{p}$. For example, $\sqrt{5}$ is the hypotenuse for the triangle with side lengths of 1 and 2. Also, 5 is the hypotenuse of the triangle with side length of 3 and 4.

4 Results

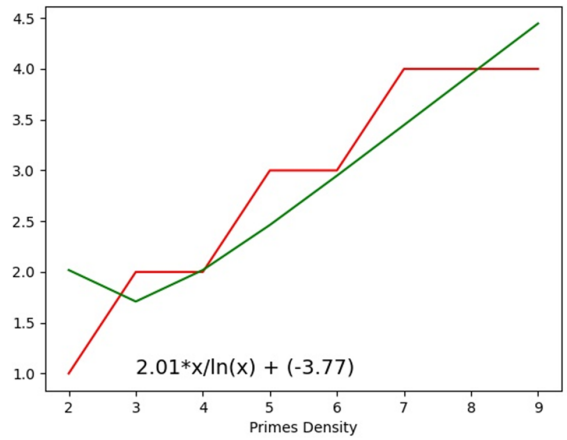
After implementing the deterministic version of Miller-Rabin, we counted prime numbers up to 10^{10} using a constant sampling interval of 10^5 to make a total of 10^5 data points. Additionally, we tested the prime properties mentioned above using the same sampling interval using the criterion listed for every property. After that, we used different fitting least-square functions to fit the densities of each type of primes. The objective functions used for every type of primes will be listed as well as the reasons accompanying the choice of each objective function.

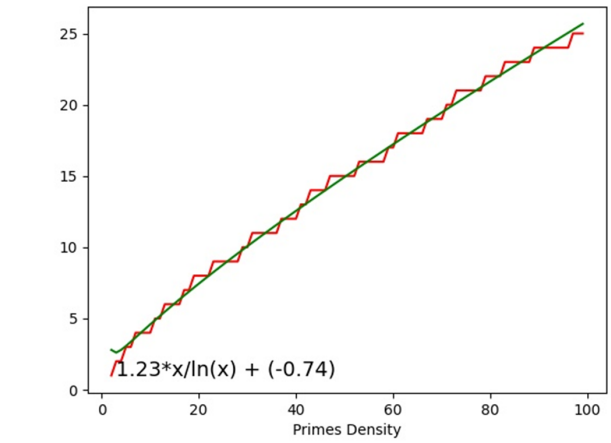
4.1 Primes Density

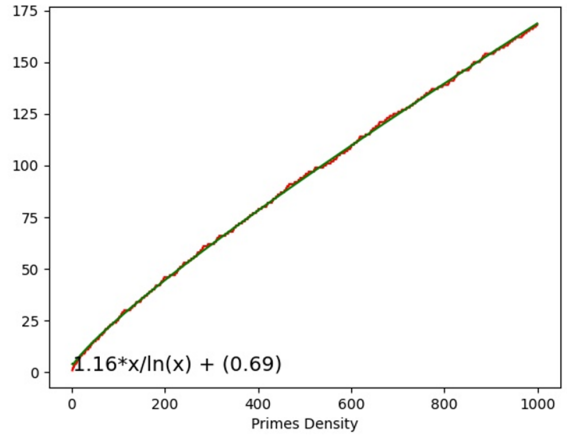
According to the prime number theorem, the prime counting function $\pi(n)$, which is the number of primes less than or equal to n, could be approximated by $\frac{n}{\ln n}$. By the prime number theorem, this statement is true:

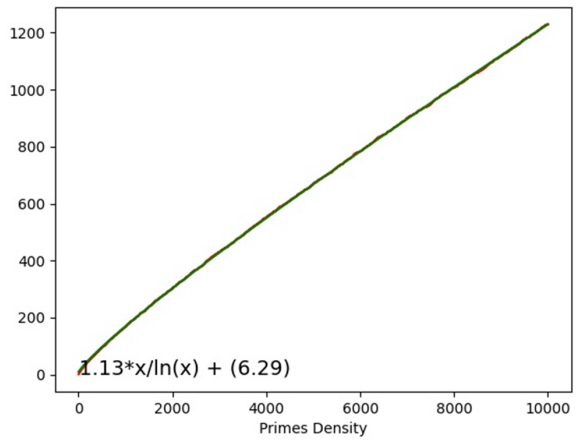
$$\lim_{x \to \infty} \frac{\pi(x)}{\left[\frac{x}{\log(x)}\right]} = 1,$$

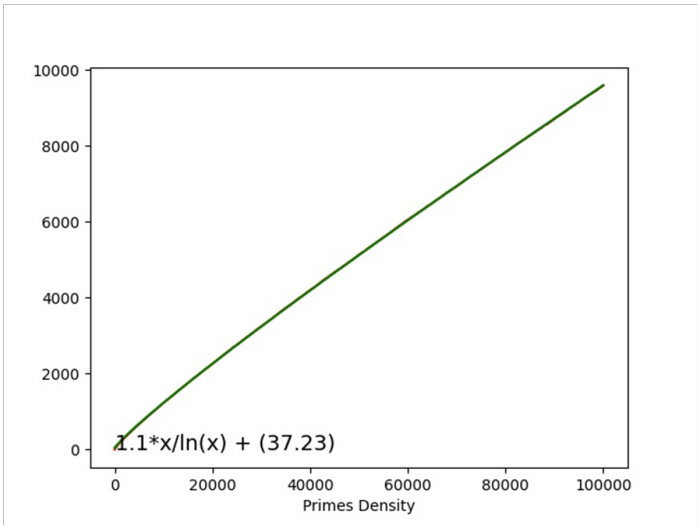
Therefore, our first analysis was to determine a fitting function $a \cdot (\frac{x}{\ln x}) + b$ using least-square regression to determine the parameters a and b. The implementation of the least-square regression is implemented in SciPy library in [5]. The results of the analysis is listed in the following graphs. The first graph is the analysis on the first 10^1 integers, the second is the analysis on the first 10^2 integers, and so on. As we see, the higher we get in the number range, the more we approach the function $\frac{n}{\ln n}$.



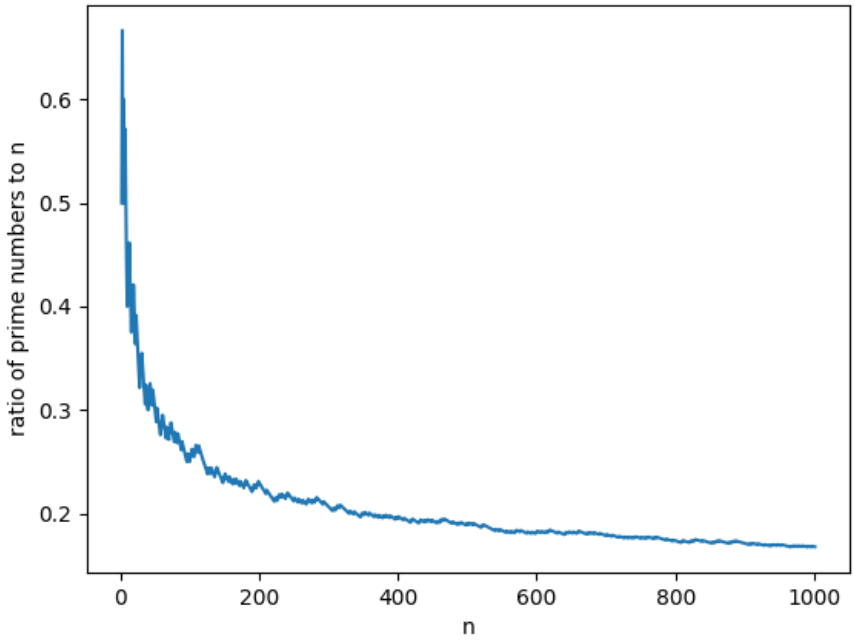








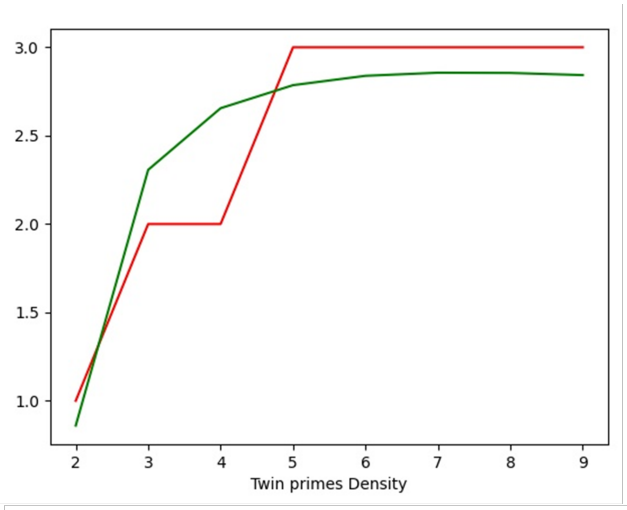
Also, we show in the next graph the ratio of prime numbers density in the set of integers as n approaches infinity.

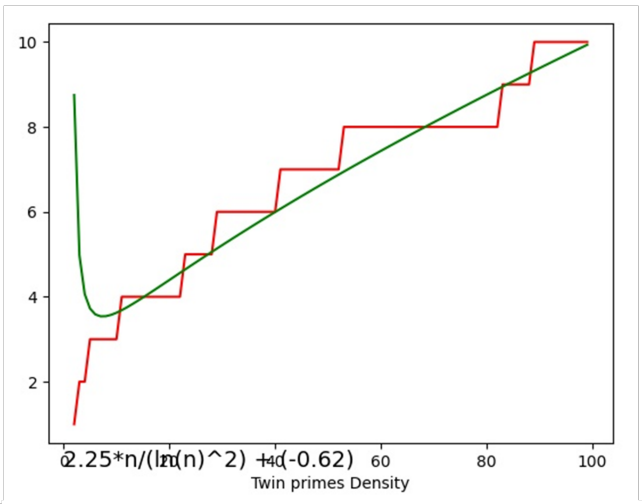


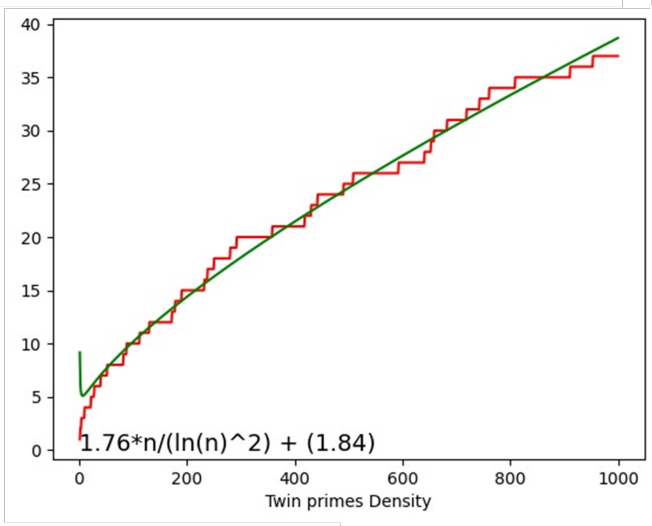
As expected, the behaviour of the prime density matches the behaviour of the function $\frac{1}{\ln x}$ as stated in the prime number theorem.

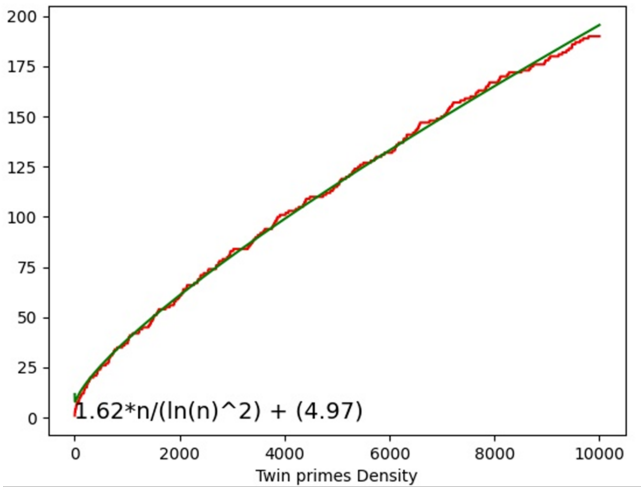
4.2 Twin Primes

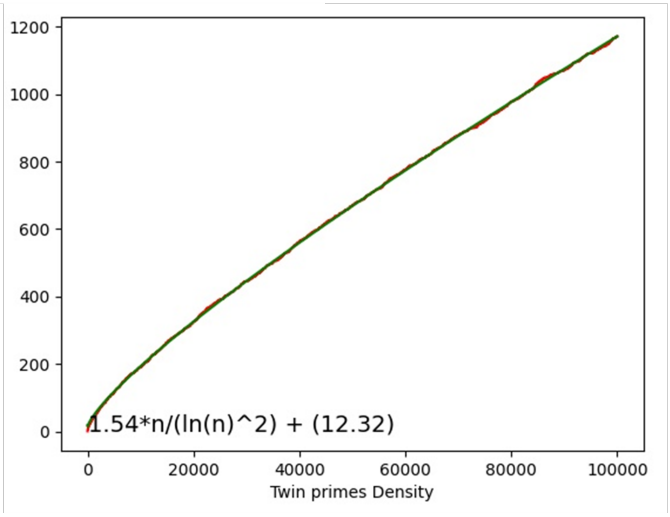
We analyze the density of twin prime pairs the same way we have done in the density section.



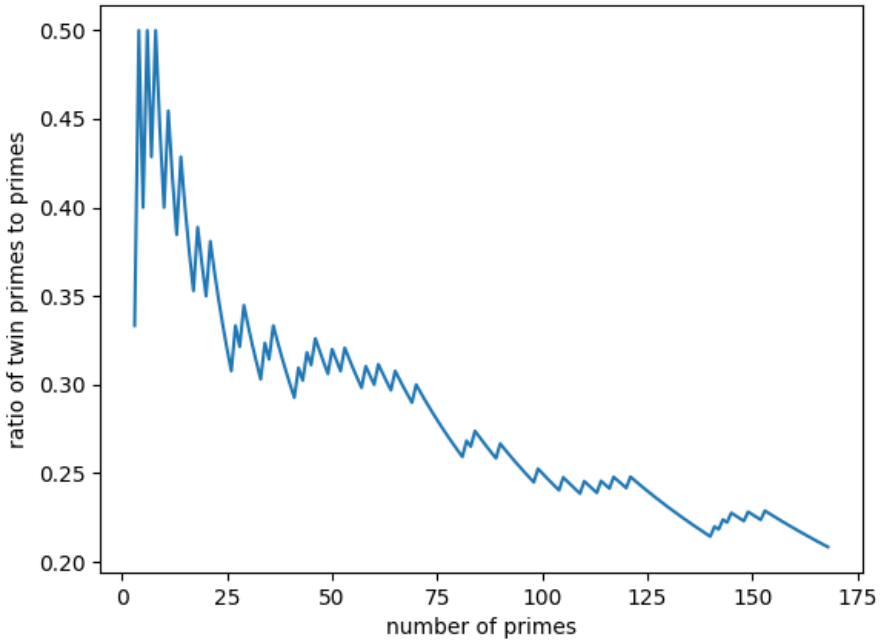






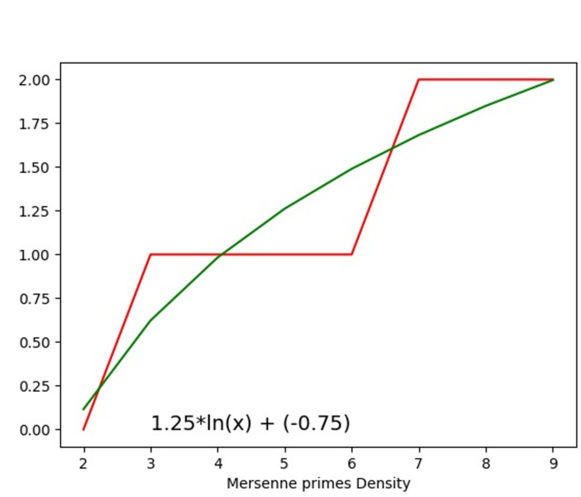


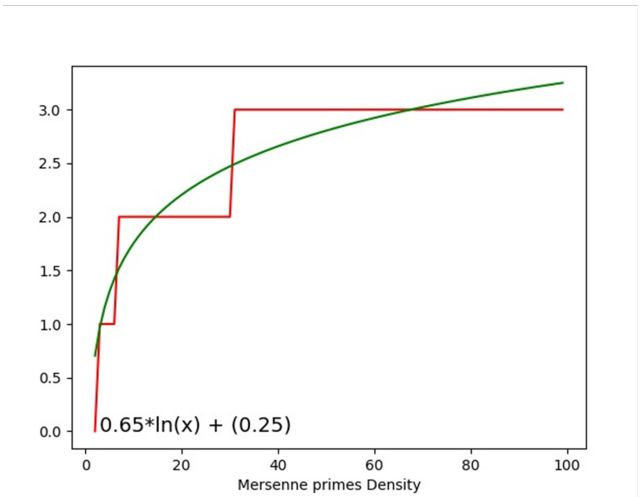
Also, we show in the next graph the ratio of Twin Primes to prime numbers density in the set of integers as n approaches infinity.

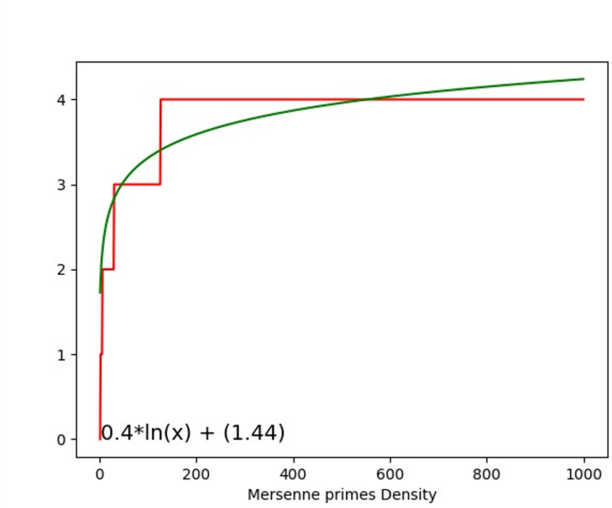


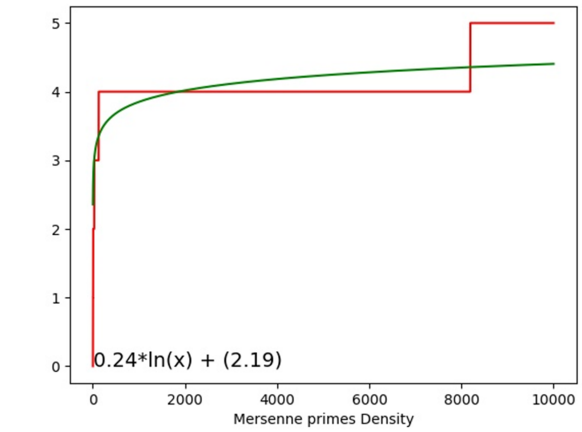
4.3 Mersenne Primes

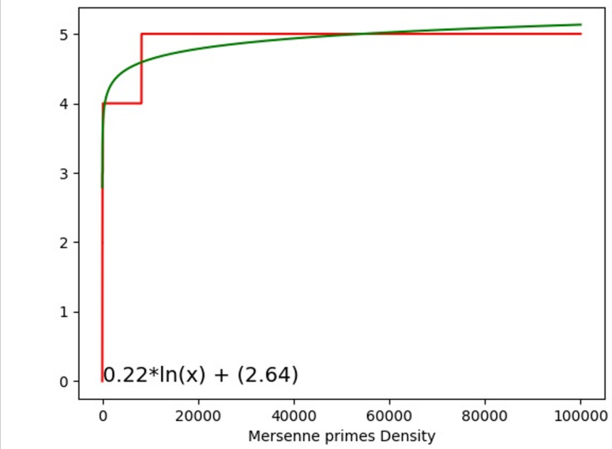
We analyze the density of Mersenne prime pairs the same way we have done in the density section.



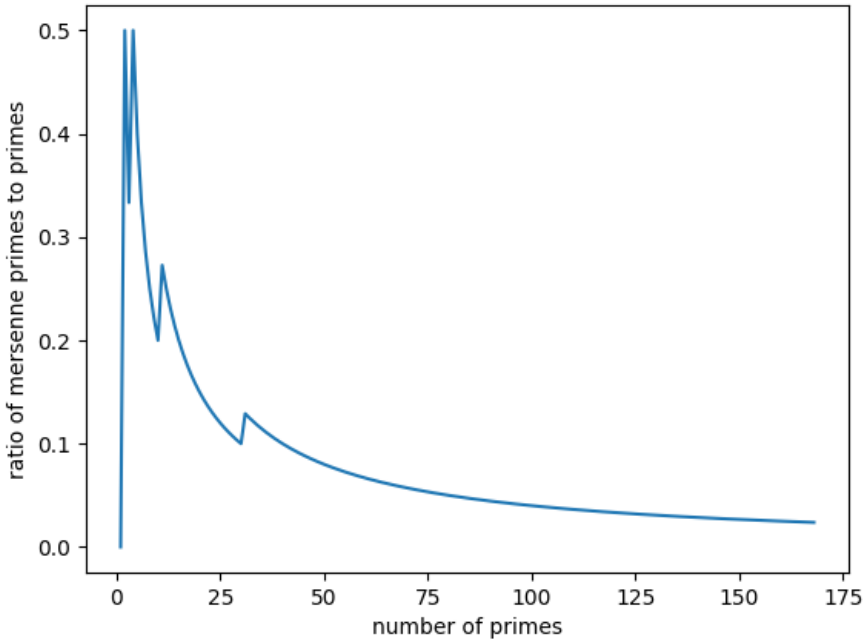






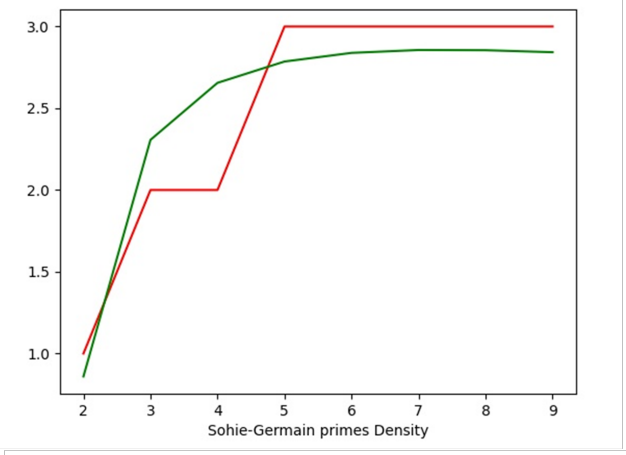


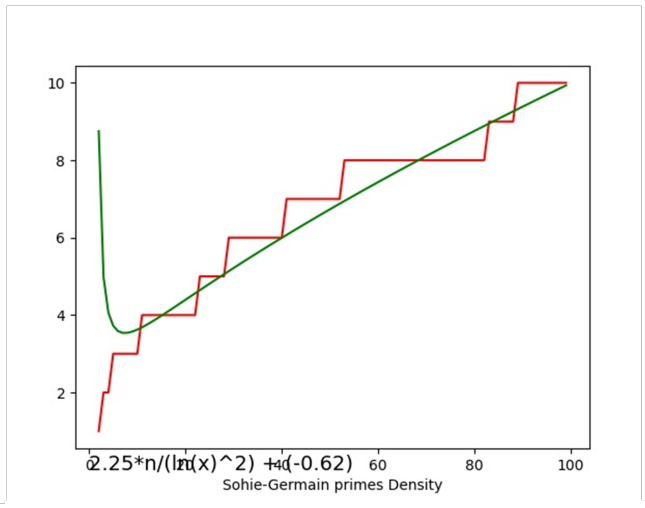
Also, we show in the next graph the ratio of Mersenne Primes to prime numbers density in the set of integers as n approaches infinity.

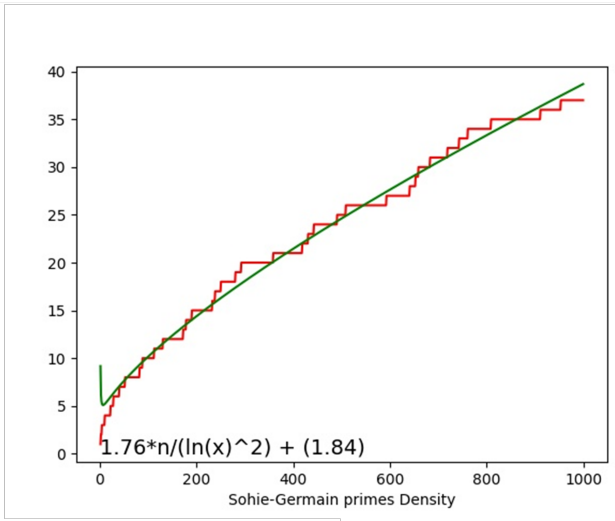


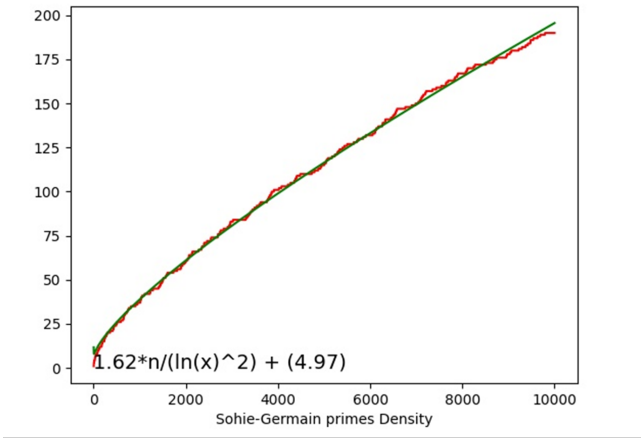
4.4 Germain Primes

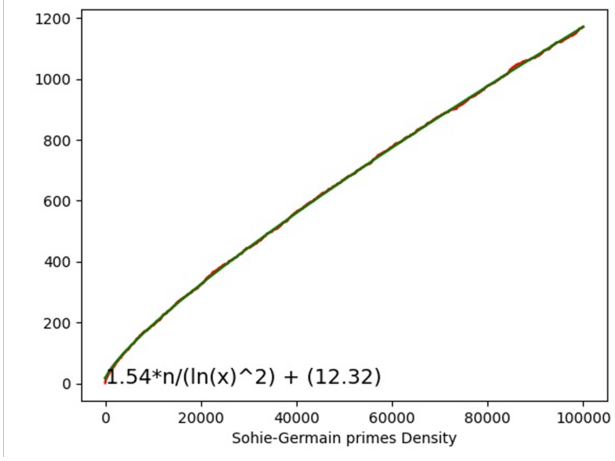
We analyze the density of Germain prime pairs the same way we have done in the density section.



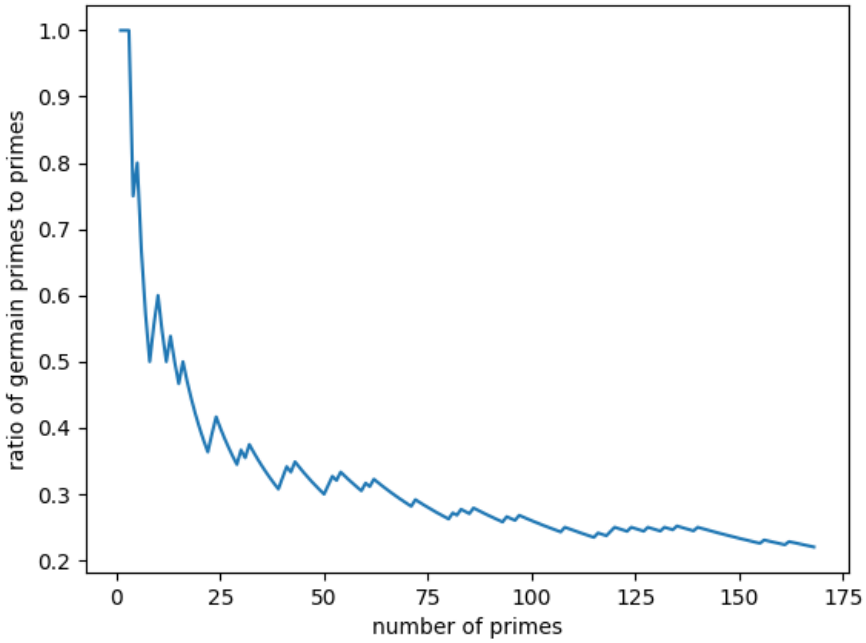






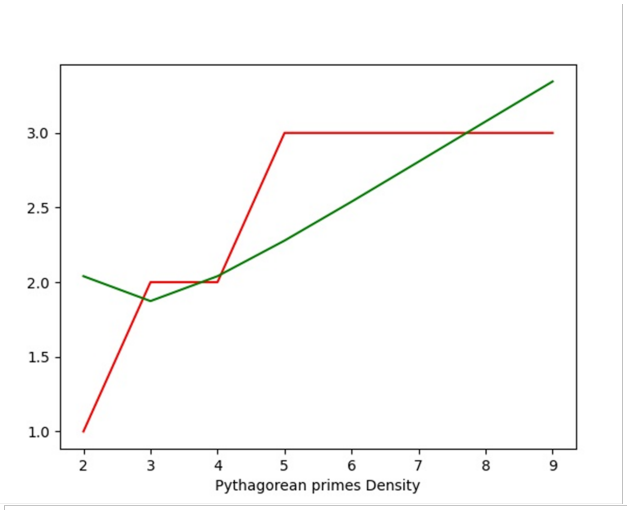


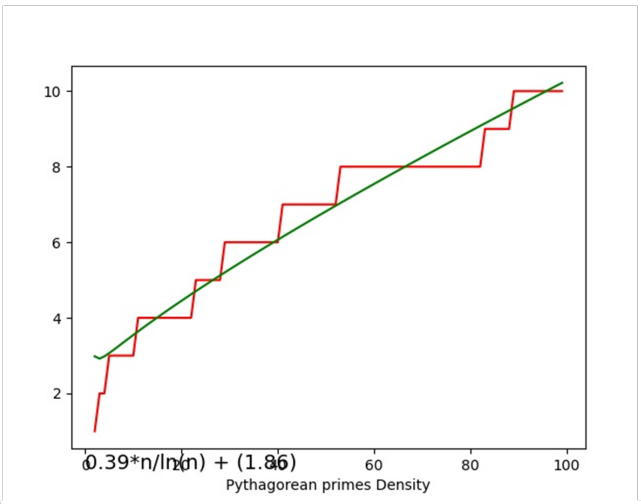
Also, we show in the next graph the ratio of Germain Primes to prime numbers density in the set of integers as n approaches infinity.

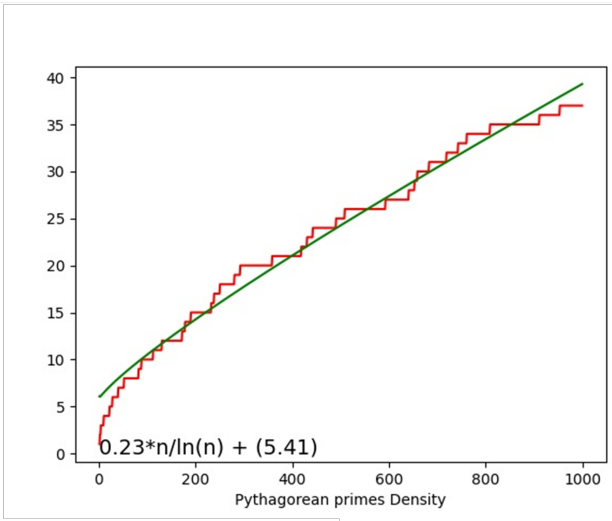


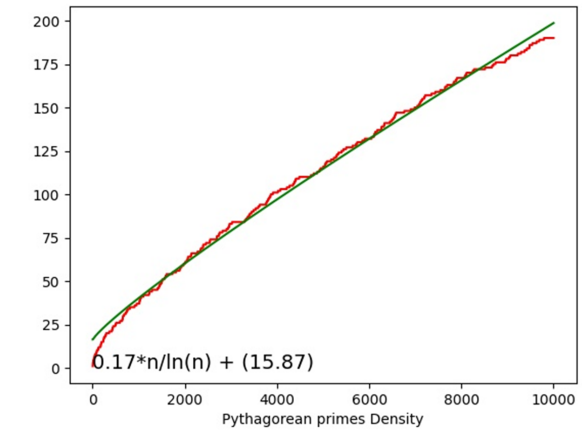
4.5 Pythagorean Primes

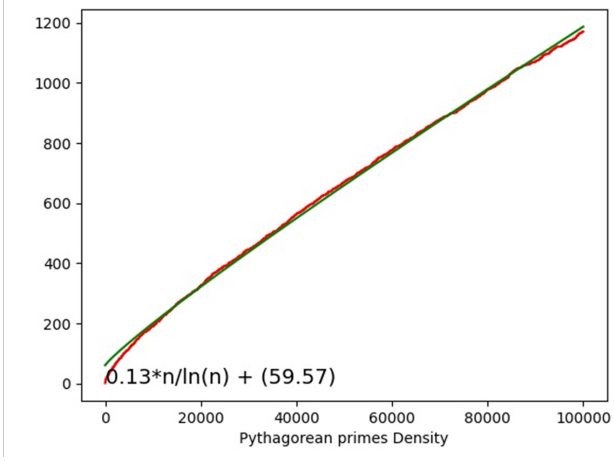
We analyze the density of Pythagorean prime pairs the same way we have done in the density section.











References

[1] McKee, M. First proof that prime numbers pair up into infinity. Nature (2013). https://doi.org/10.1038/nature.2013.12989

[2] Zhang, Y. (2014, May 1). Bounded gaps between primes. Annals of Mathematics. https://doi.org/10.4007/annals.2014.179.3.7

[3] D. H. Lehmer. "Tests for primality by the converse of Fermat's theorem." Bull. Amer. Math. Soc. 33 (3) 327 - 340, May-June 1927.

[4] Jaeschke, G. (n.d.). American Mathematical Society. On strong pseudoprimes to several bases. https://doi.org/10.1090/S0025-5718-1993-1192971-8

[5] https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve_fit.html