Halmstad
KATTEGATTGYMNASIET

Understanding How the Largest Primes Came to Be

Ibrahim Abdulhussein,
Robin Brusbo

April 20, 2020

$$\pi(x) - \frac{x}{\ln x}$$

$$\int_2^x \frac{1}{\ln t}\,\mathrm{d}t - \pi(x)$$

1

---
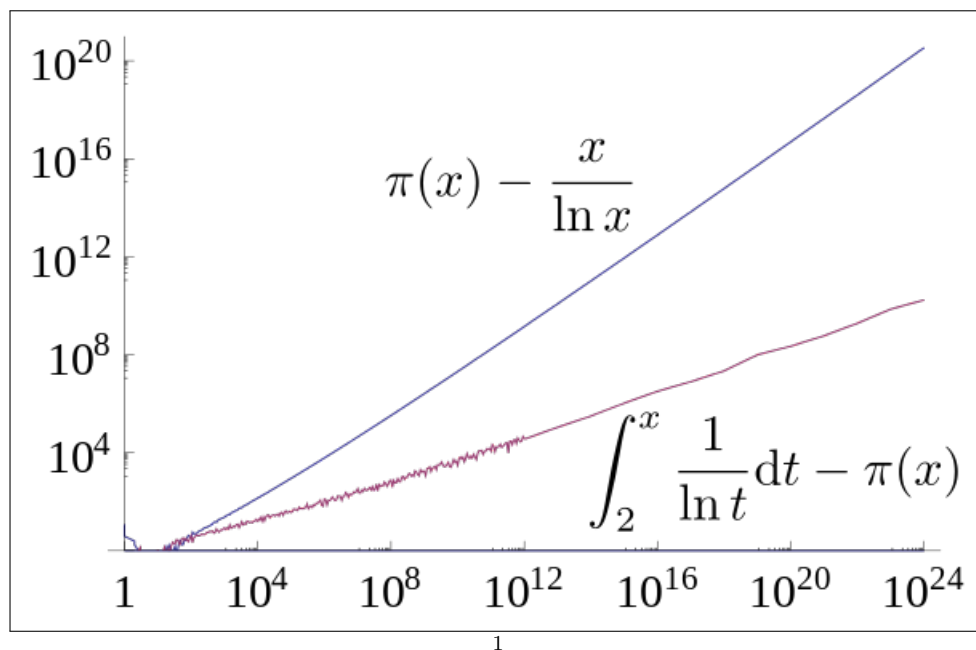
[1]The Prime Number Theorem [1]

Abstract

Write later

Contents

1 Introduction

1.1 History

Prime numbers have been known to mankind since Ancient Greece, [2]

1.2 Aim

The purpose of this essay is to investigate prime numbers. Investigating all the aspects of prime numbers would be a near impossible task; therefore, this essay will focus more on learning about the methods and algorithms of finding new primes. The study will also present ways to prove that a given number, n, is indeed prime. Primes are without a doubt beautiful mathematically, but what real life application do primes have?

The relevant questions for this study are the following:

1. What methods can be used to find new primes?

2. What are some algorithms for primality testing?

3. Are there "accurate" ways to measure the amount of primes under a given number, N?

4. Which algorithm is the most efficient and why?

5. Why are primes even relevant to us?

2 Definitions

In order to fully understand this essay, some key concepts have to be adressed and understood.

2.1 What is a prime?

Prime numbers are defined as *positive integers* which only have the factors 1 and itself. Thus 4 is not a prime since $4 = 2 * 2$. On the other hand 5 is a prime since the only divisors of 5 is 1 and 5.

If a number, n, is not prime, it is refered to as a *composite number*.

2.2 The Fundamental Theory of Arithmetic

The Fundamental Theory of Arithmetic [3] states that all integers greater than 1 is either a prime, or can be expressed as a product of primes in a unique way. This means that all natural numbers, except for 1, has its own factorization containing only primes, unless it is a prime itself.

Important to know is that there is an infinite amount of primes. The proof is a quite easy by contradiction, but nonetheless beautiful:

> Assume that there is a finite amount of primes and make a list of them:
> $$p_1, p_2, p_3, p_4, p_5, ...$$
>
> Let the constant Q be the product of all the primes in the list and add 1:
> $$Q = p_1 * p_2 * p_3, ... + 1$$
>
> According to the fundamental theorem of arithmetic, Q must be a prime since none of the primes in the list divide Q evenly because of the 1; therefore making the list incomplete and proving that you cannot make a finite list of all primes.

2.3 The Prime Number Theorem

The Prime Number Theorem [1] describes approximately how many primes there are less than or equal to a given number. The function $\pi(N) \sim \frac{N}{ln(N)}$ gives the expected amount of primes below a certain N. Graphing this function shows that primes become less common for greater N.
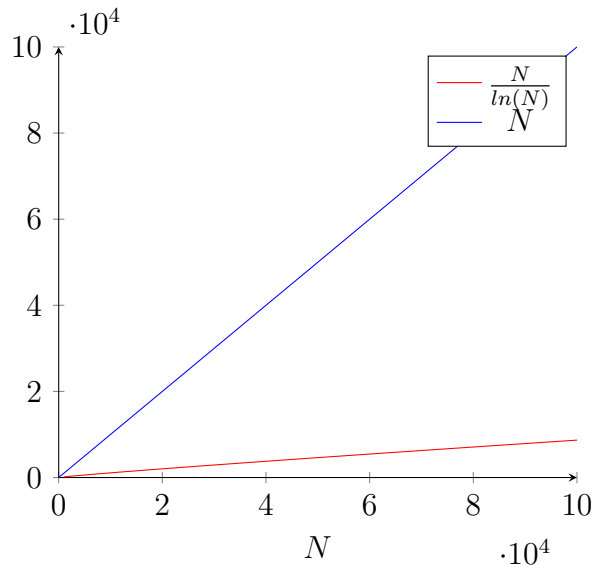
This proves that primes do not show up linearaly, meaning a computer that is twice as powerful will *not* produce twice as many primes. Instead, the most important and crucial part of generating and verifying primes are the *algorithms*.

2.4   Time Complexity

Time complexity [4] is a concept within computer science, which describes the approximate time for a program to complete. The study will make heavy use of the Big O Notation [5], which notates how the run time increases as the input size increases. For example, $O(N)$ will grow linearly with the input size. Increasing the input size by a factor of 10, will also increase the run time by a factor of 10, as such $O(10N)$. On the other hand, $O(log(n))$ grows logarithmically, which is far more efficient for bigger input sizes.

This is very important when it comes to testing large numbers, because it makes it easier to determine whether the program will succeed or run for a *very long time*[2]. It is therefore important to write as efficient algorithms as possible, considering the fact that the largest known prime has $24,862,048$ digits [6].

---

[2]Some programs will not finish until the sun explodes, which is quite impractical.

3 Materials and Methods

3.1 Resources

The most important resource to be used in this study will be Wikipedia and the countless articles written regarding this subject. The programming language used to implement and time the algorithms will be Python[3] [7].

As of today, the largest primes are found using supercomputers running these algorithms. Using a supercomputer is not plausible for this study, therefore the maxmimum limit for primes will be 10^{14}.

3.2 Methods

During the primality testing [8] there will be two test cases for the algorithms. The first one will determine if n is a prime or composite, and the second one will generate the largest prime within a time limit. At the start of the code a driver function will be defined to serve as the "starter" code for the program - defining what test to perform. The driver function will call a primality testing function called **primeTest** which will return a boolean[4] value.

Two groups of algorithms will be tested: the "simple" methods and the probabilistic tests. The driver function will be called to start a timer and execute one of the following algorithms:

1. "Simple" methods

   (a) Brute-force
   (b) Smart Brute-force
   (c) Sieve of Eratosthenes [9]

2. Probabilistic tests [8]

   (a) Fermat Primality
   (b) Miller–Rabin

The function will be called upon three times to average an "accurate" runtime. The same integers will be used across all tests. These runtimes are noted and later compared in the discussion. The following Python code describes the setup:

---

[3]Although Python is one of the slowest languages, the simplicity of it makes it an invaluable tool for this study.

[4]A function that returns $True$ or $False$

```
    import time

    driver()

    def driver():
        numbers = [...]
        times = []
        for n in numbers:
            start = time.time()
            primeTest(n)
            runTime = time.time() - start
            times.append(runTime)
        print(sum(times) / len(times))

    def primeTest(n):
        # Insert the algorithm here
```

To generate the largest prime within 30 seconds, the **primeTest** function will remain the same, however the driver function will recieve some changes. Study the following code:

```
    def driver():
        primes = []
        timelimit = 30
        start = time.time()
        for n in range(time.time() - start < timelimit):
            if primeTest(n):
                primes.append(n)
        print(max(primes))
```

3.3   Algorithms

Explain the algorithms and "tricks" to make them run faster. The results must only be tables of runtimes of the different algorithms.

Sieve of Erathsoneirnsn is

4 Results

4.1 Primality Testing

There are many different ways to test if a natural number, n, is prime or composite. The most trivial way is to check every number from 2 to $n - 1$ and see whether or not they divide n evenly. If no natural number is a factor of n, the program will return that n is a prime. The described algorithm is usually called *Brute-force*.

```
def primeTest(n):
    for i in range(2, p - 1):
        if n % i == 0:
            return "Composite"
    return "Prime"
```

The run time of this algorithm was XXX seconds. This method runs in $O(n)$ time, which is relatively slow. $O(n)$ lets us check numbers less than or equal to approximately XXX. Compared to other methods, this turned out to be the slowest.

This method can be improved quite a bit though after realizing two important things: 1. We do not need to check all integers up to p, it is sufficient only to test numbers up to sqrt(p) 2. We do not need to check any even numbers, except for the number 2. This is because all even numbers have 2 as a factor, and therefore 2 would already be a known factor of p, thus concluding that p is composite.

Rewriting the program:

```
def primeTest(n):
    if n % 2 == 0 or n % 3 == 0:
        return "Composite"
    for i in range(5, sqrt(n)):
        if n % i == 0:
            return "Composite"
    return "Prime"
```

This resulted in the run time of XXX seconds. The time complexity. The

4.2 Generating Largest Prime

5 Discussion

: (

6 Conclusion

: (((((())))))

7 Acknowledgments

8   References

[1] Wikipedia. *Prime number theorem.* https://en.wikipedia.org/wiki/Prime_number_theorem. Information taken January 3, 2020.

[2] Wikipedia. *Prime Number.* https://en.wikipedia.org/wiki/Prime_number. Information taken December 5, 2019.

[3] Wikipedia. *Fundamental theorem of arithmetic.* https://en.wikipedia.org/wiki/Fundamental_theorem_of_arithmetic. Information taken December 28, 2019.

[4] Wikipedia. *Time complexity.* https://en.wikipedia.org/wiki/Time_complexity. Information taken January 3, 2020.

[5] Wikipedia. *Big O notation.* https://en.wikipedia.org/wiki/Big_O_notation. Information taken January 3, 2020.

[6] Wikipedia. *Largest known prime number.* https://en.wikipedia.org/wiki/Largest_known_prime_number. Information taken December 27, 2019.

[7] Python Software Foundation. *Python.* https://www.python.org/. Information taken January 3, 2020.

[8] Wikipedia. *Primality test.* https://en.wikipedia.org/wiki/Primality_test. Information taken January 7, 2020.

[9] Wikipedia. *Sieve of Eratosthenes.* https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes. Information taken January 7, 2020.