

MAIN TITLE HERE
SUBTITLE HERE

Created by Aiden Taylor and Noah Pinel

Contents

1	Table of contents	2
2	Abstract	3
3	Implementation of The Sieve of Eratosthenes in Python	4
3.1	Overview	4
3.2	The Implementation	4
3.3	Conclusion	5

Abstract

Give a general Overview of our project, explain high level what we each did...

3 Implementation of The Sieve of Eratosthenes in Python

3.1 Overview

The Sieve of Eratosthenes is a type of prime sieve. Prime sieves are algorithms that are very good at generating prime numbers up to some upper bound n , where $n \in \mathbb{Z}^+$. The following sections explain the Implementation process of simulating the Sieve of Eratosthenes in the programming language Python.

3.2 The Implementation

My first run in with the sieve of Eratosthenes was during my number theory course, the mechanical nature of the sieve seemed to be a perfect way to utilize a computer to do some interesting math. The entirety of the program relies on the algorithm below, so here are the meat and potatoes of the whole thing,

Algorithm 1 : An algorithm for The Sieve of Eratosthenes

Require: : $n > 0$

Ensure: : List of prime numbers 2, ..., n

```

1: Sieve( $n$ ) :
2: Populate a boolean list P size 2, ...,  $n$ .
3: Initialize all index values to True.
4: for  $i = 2, \dots, n + 1$ : do
5:   if P[ $i$ ] TRUE
6:     for  $j = i^2, i^2 + 1, i^2 + 3, \dots, n$ : do
7:       set P[ $j$ ] False
8:     end for
9:   end for
10: Return P

```

Simply put, we have a list of size 2, ..., n . We set all values of our list to True, then starting at 2 we loop through each index setting multiples of the respected current position to false. After this has run, we are left with a list where all prime values are unchecked, that is, they remain True while composite numbers are false. Below is a visual of the algorithm working for a list of size 10.

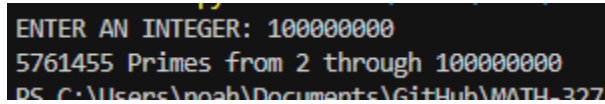
```

ENTER AN INTEGER: 10
0: -->  T,T,T,T,T,T,T,T,T,T
1: -->  T,T,F,T,F,T,F,T,F,T
2: -->  T,T,F,T,F,T,F,T,F,T
3: -->  T,T,F,T,F,T,F,T,F,T
4: -->  T,T,F,T,F,T,F,T,F,T
5: -->  T,T,F,T,F,T,F,T,F,T
6: -->  T,T,F,T,F,T,F,T,F,T
7: -->  T,T,F,T,F,T,F,T,F,T
8: -->  T,T,F,T,F,T,F,T,F,T

```

We can see that after 8 iterations of following the algorithm the only index values set True are 2, 3, 5, 7, i.e., the first 4 primes. Now as cool as it is to visualize the algorithm I feel like finding the

number of primes \leq some large n is a lot more interesting. To help with run time I disabled the visual aspect of the algorithm and was able to find a value for the primes up to 100,000,000.



```
ENTER AN INTEGER: 100000000
5761455 Primes from 2 through 100000000
PS C:\Users\nash\Documents\GitHub\MATH_327
```

Using the forbidden site Wolfram Alpha to verify, there are indeed 5,761,455 primes in between 2 and 100,000,000.

3.3 Conclusion

The last calculation presented took just about an epsilon under a minute to calculate, which for an algorithm that has been around for thousands of years I feel is very impressive, thus, Eratosthenes sieve is a powerful tool to find primes up to some appropriate positive integer. This problem also displays the interplay between Mathematics and Computer Science and how they complement each other very well. The source code for Sieve.py will be included in our submission with documentation if you're wanting to poke around.