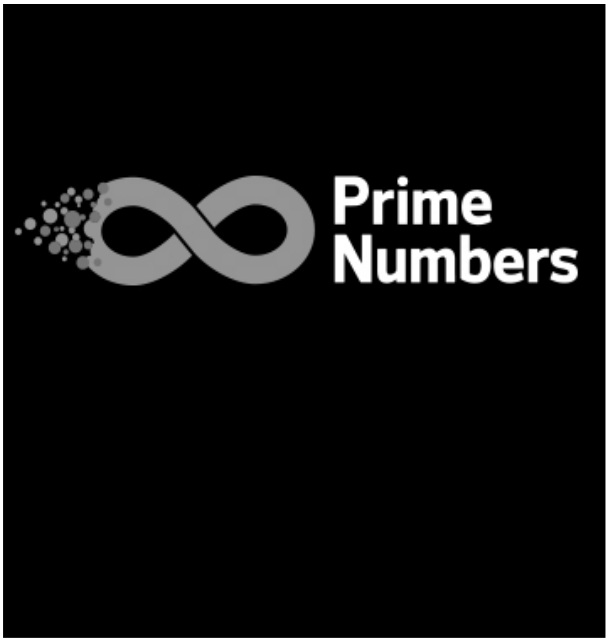

King Abdulaziz University

Faculty of Engineering

Electrical and Computer Engineering Department

Prime Number in Range

EE305-Discrete Mathmatics Project

Instructor:

Dr. Emad Khalaf

Done by:

1 – Yazeed Alzughaibi

2 – Abdulrahman Aljuhani

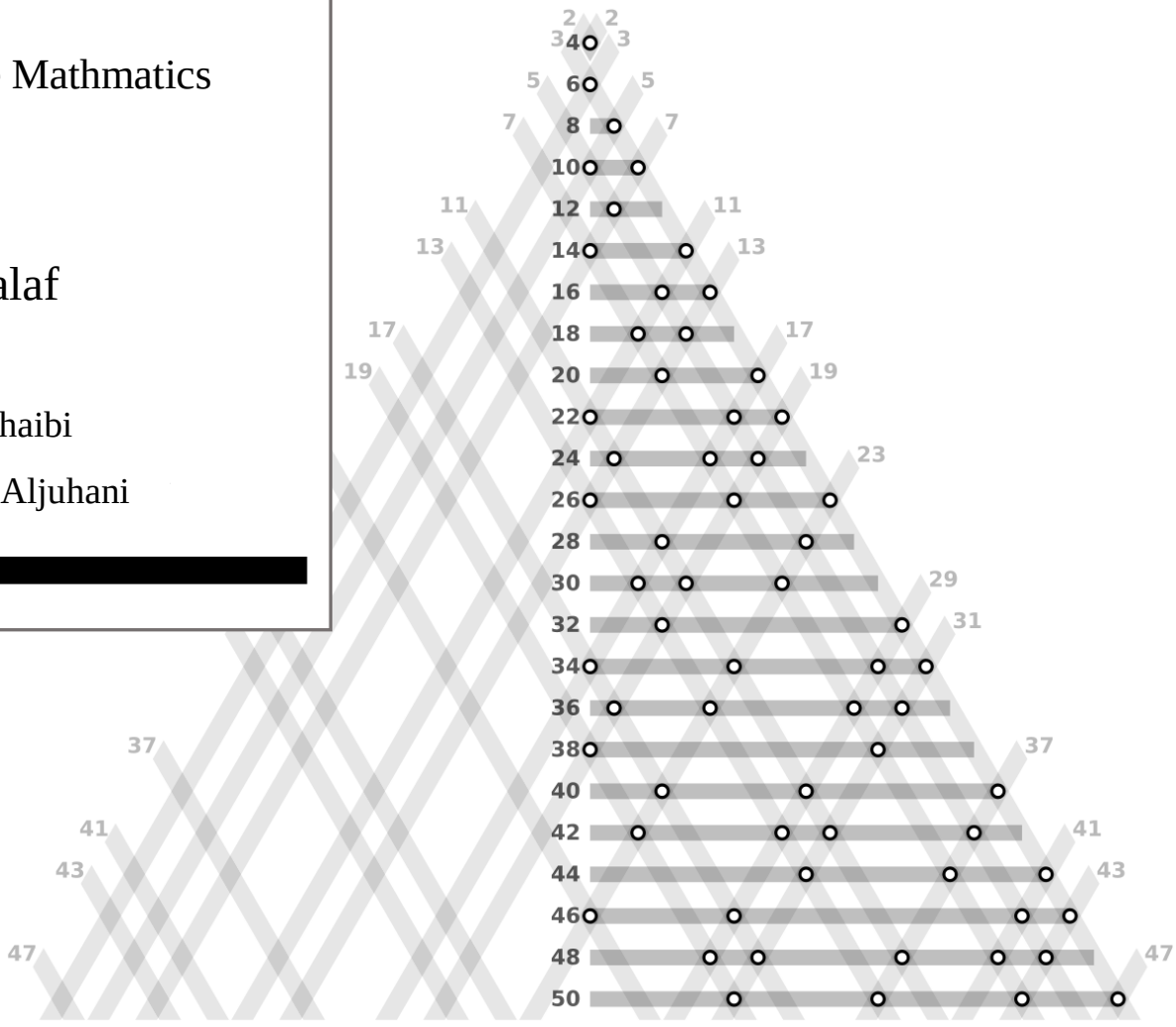

Table of Contents

Table of Figures

Introduction:

For this course Discrete mathematics (EE305), we study the mathematical structure that are fundamentally discrete rather than continuous and describe objects and problems in branches of computer science, such as computer algorithms, software development and programming languages [1]. the best way to understand something is implementation, that is why we were assigned to do this project. The goal of the project is to understand the material of the course and expand our knowledge in python programming and how to implement these algorithms. In this project it is expected from us to create a program using any language we prefer to implement any subject from the course. So, we chose to implement prime numbers using python programming language. In this report, we will discuss what are prime numbers, how they are checked for, why we chose python programming language, explain each algorithm is detail, demonstrate the python program, how it works and the final output.

Background info:

To begin with, we ask ourselves; what are prime numbers? After a short research session, we found that prime numbers are numbers that have only two factors: 1 and themselves. For example, the first and the most recognized prime numbers are 2, 3, 5 & 7. On the contrary, the numbers that have more than 2 factors are called composite numbers. To identify a prime number, there multiple techniques for doing so [1].

Applications of prime numbers:

There are multiple applications of prime numbers in real-life such as encryption and gears teeth. A real-life practical example is when designing a gearbox, different sized gears have certain numbers of teeth, usually the numbers of teeth comes from prime numbers to avoid any gear from hitting any other individual gear more often in case of small imperfections or gears stuck on gears [2]. in **Figure-1** we can see an example of designing a gear with prime number teeth.

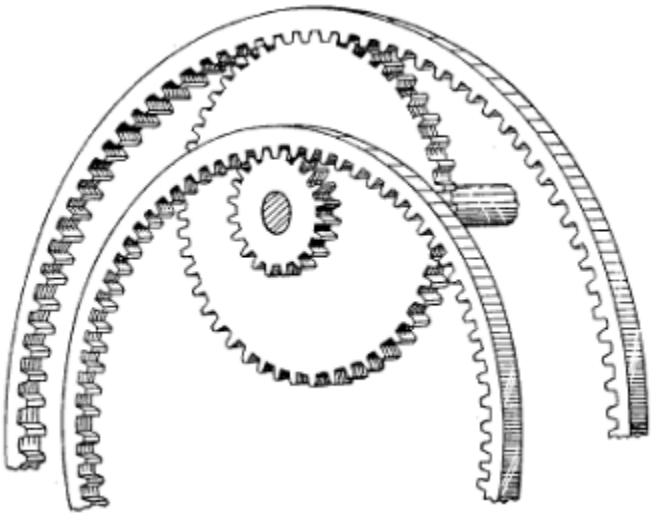

Figure 1 - application of prime numbers in real life.

Quick explanation about algorithms:

One of the methods to calculate the number to check if a number is a prime number or not. First where we divide the number by every number from 2 to the number before it as shown in Algorithm (1):

$$\text{Algorithm (1): } n = n + 1\,, where\ 1 < n < x$$

and x is the number we want to check if it is a prime number or not, then we will check if any of the numbers are smaller than x result a natural number or not, if not then it is a prime number. for example, if we took the number 9 and we check:

$$n = 1 + 1 = 2, \qquad 9 \div 2 = 4.5\ is\ not\ a\ natural\ number$$

$$n = 2 + 1 = 3, \qquad 9 \div 3 = 3\ is\ a\ natural\ number$$

Since it can by divided by a natural number other than itself and one it is considered a composite number and not a prime number.

The other method is dividing the numbers from 2 to its square root integer instead of the base number itself as the previous method, the new Algorithm (2) is shown below:

$$\text{Algorithm (2): } n = n + 1\,, where\ 1 < n < \sqrt{x}$$

And x the square root of the inputted number, we can take for example the number 9 again and check:

$$n = 1 + 1 = 2, \qquad \sqrt{9} \div 2 = 3 \div 2 = 1.5\ is\ not\ a\ natural\ number$$

$$n = 2 + 1 = 3, \qquad \sqrt{9} \div 3 = 3 \div 3 = 1\ is\ a\ natural\ number$$

Since the final result was 1, x is not considered a prime number but a composite number [1].

Why we chose Python Programming Language:

The reason why we chose Python Programming Language is that the Python Programming Language is one of the most accessible programming languages available and the available learning resources for it is endless, because of its high level and simplified syntax. Due to its ease of learning and its infinite capabilities in AI learning. Also, this semester we registered in EE361 course which requires us to study Python Programming Language for its assignments and projects. [3]

The Algorithms:

The code we have written for both algorithm's functions in a very similar way. The only difference between them is the method of calculating/checking for each number in the range. They both have nested loops; they both measure the CPU execution time the same way and they both output the same way. To differentiate between the two algorithms we measure the CPU execution time using a Python built-in library called "time", we just import the libraries needed and start the timer after the input is entered until the prime numbers are finished in that range of numbers.

Algorithm (1):

In the first algorithm, the user inputs the lower end of the range and the higher end of the range, after that we start the timer (time1), we make a loop that goes through that range of numbers and in that for loop we need to test each number, first check it is greater than one ($x > 1$) and if it is we make a second loop that starts from two to that number (*for y in range (2 : x)*) and see if any number smaller than it that divides it and if not, that number is stored in the (listofprime) list, otherwise we break the second loop and got to the next number in the range. After the loop is finished, we stop the timer and store it for later to show. At the end of the algorithm the output should look. **Figure-2** and **Figure-3** and **Figure-4** explains the process used.

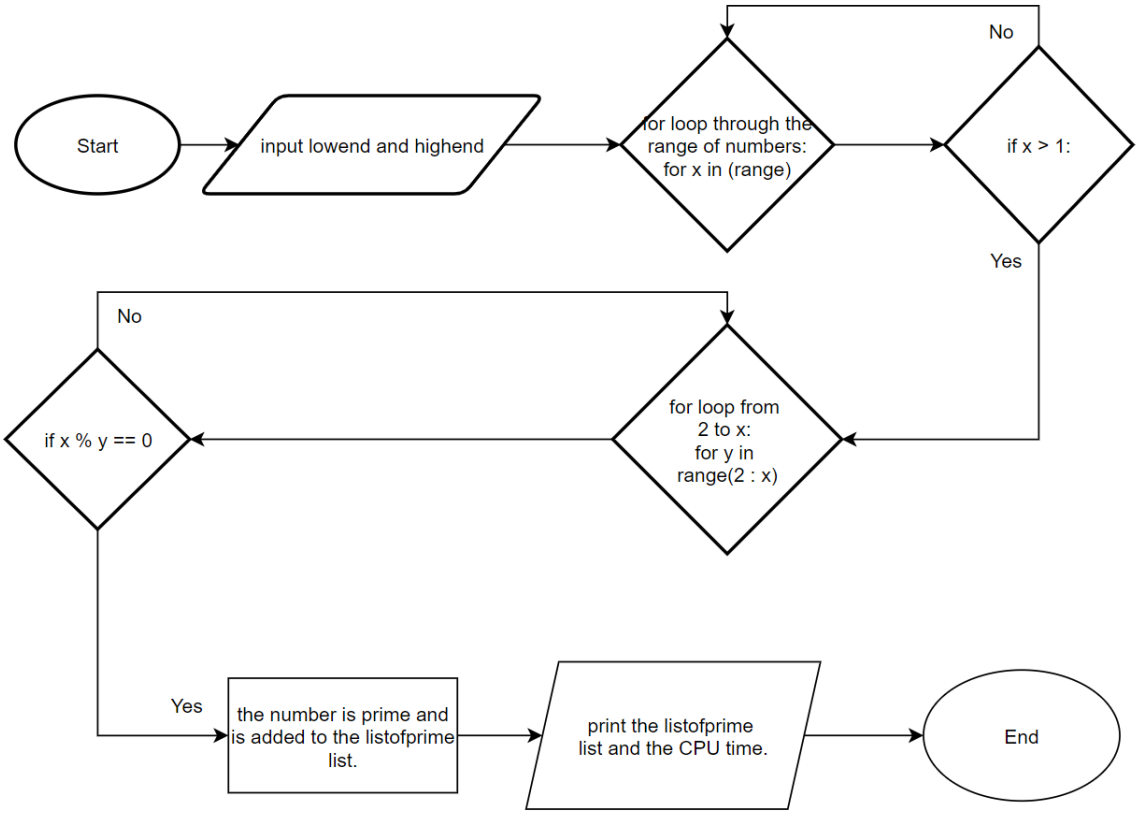

Figure 2 - flowchart for algorithm1

```
print('Algorithm 1: ')
lowend = int(input('Enter the lower end of the range: '))
highend = int(input('Enter the higher end of the range: '))
time1 = time.time() #save time stamp after input is done
listofprime = [] #list to hold the prime numbers later on
for x in range(lowend, highend + 1): #loop to go through the range of numbers
    if x > 1: #check if its greater than one
        for y in range(2, x):
            if x % y == 0: #if that specific number in the range can be divided by any number then it's not a prime number
                break
        else:
            listofprime.append(x) #add that number to the list to display later
calctime1 = time.time() - time1 #calculate the CPU time for the algorithm
#final print
print('the prime numbers between ' + str(lowend) + ' and ' + str(highend) + ' are:')
print(listofprime)
print(calctime1)
```

Figure 3 - code for algorithm1

```
Algorithm 1:
Enter the lower end of the range: 1
Enter the higher end of the range: 1000
the prime numbers between 1 and 1000 are:
[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, 101, 103, 107, 109, 113, 127
, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263
, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419
, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577
, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739
, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911
, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
0.00446319580078125
```

Figure 4 - output for algorithm1 with range (1 : 1000) as example

Algorithm (2):

The second algorithm is not much different than the first one, the only difference I as we said before is the way we check each number in the second for loop; instead of dividing the number by every number smaller than it until we reach two, we divide it by its square root until we reach 2 *(for y in range (2 : √ (x))*). After that it's the same as the first algorithm, we store the CPU execution time, and display the output. **Figure-5** and **Figure-6** and **Figure-7** explains the process used.

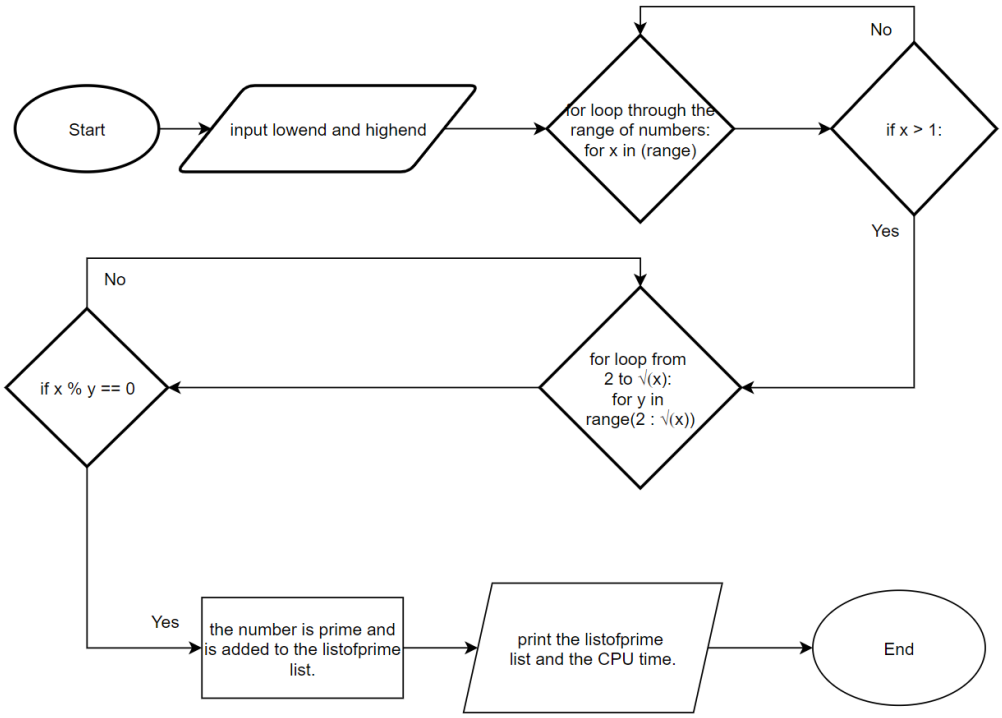

Figure 5: flowchart for algorithm2

```
lowend = int(input('Enter the lower end of the range: '))
highend = int(input('Enter the higher end of the range: '))
time2 = time.time() #save time stamp after input is done
listofprime = [] #list to hold the prime numbers later on
for x in range(lowend , highend + 1):  #loop to go through the range of numbers
    if x > 1: #check if its greater than one
        for y in range (2, int(math.sqrt(x)) + 1):
            if x % y == 0: #if that specific number in the range can be divided by any number then it's not a prime number
                break
        else:
            listofprime.append(x) #add that number to the list to display later
calctime2 = time.time() - time2 #calculate the CPU time for the algorithm
#final print
print('the prime numbers between ' + str(lowend) + ' and ' + str(highend) + ' are:')
print(listofprime)
print(calctime2)
```

Figure 6 - code for algorithm2

```
Algorithm 2:
Enter the lower end of the range: 1
Enter the higher end of the range: 1000
the prime numbers between 1 and 1000 are:
[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, 101, 103, 107, 109, 113, 127
, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263
, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419
, 421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577
, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739
, 743, 751, 757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911
, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
0.0009918212890625
```

Figure 7 - output for algorithm2 with range (1 : 1000) as example

Demonstration of how the program works and the output:

The final program is just a big while loop to keep receiving input from the user until the user exits with 0 as input. We just separated each algorithm to their corresponding number; input = 1 uses Algorithm (1) and if input = 2 uses Algorithm (2). We used two Python built-in libraries, math for square root function and time for time CPU execution time. When the program is run the user should start by choosing the Algorithm (1) or (2), after that they enter the range they want to find the prime numbers in, then the program outputs the list of prime numbers in that range and the CPU execution time and then loops back to the first step until the user inputs 0 to end the program. **Figure-8** shows the full code, and **Figure-9** shows the program running.

```python
import math
import time
'''
Names:
1- Yazeed Mohammed Alzughaibi - 1847186
2- Abdulrahman Sameer Aljuhani - 1847693
This is EE305 project
this python written program has two differant algorithms
that will recive a range of numbers from the lowend to
the highend and outputs all the prime
numbers in between them
'''
choice = 88
while (choice != 0):
    choice = int(input('Enter choice of algorithms to calculate prime numbers 1 or 2 & 0 for exiting:'))
    if choice == 1:
        print('Algorithm 1: ')
        lowend = int(input('Enter the lower end of the range: '))
        highend = int(input('Enter the higher end of the range: '))
        time1 = time.time() #save time stamp after input is done
        listofprime = [] #list to hold the prime numbers later on
        for x in range(lowend, highend + 1): #loop to go through the range of numbers
            if x > 1: #check if its greater than one
                for y in range(2, x):
                    if x % y == 0: #if that specific number in the range can be divided by any number then it's not a prime number
                        break
                else:
                    listofprime.append(x) #add that number to the list to display later
        calctime1 = time.time() - time1 #calculate the CPU time for the algorithm
        #final print
        print('the prime numbers between ' + str(lowend) + ' and ' + str(highend) + ' are:')
        print(listofprime)
        print(calctime1)
    elif choice == 2:
        print('Algorithm 2: ')
        lowend = int(input('Enter the lower end of the range: '))
        highend = int(input('Enter the higher end of the range: '))
        time2 = time.time() #save time stamp after input is done
        listofprime = [] #list to hold the prime numbers later on
        for x in range(lowend , highend + 1):  #loop to go through the range of numbers
            if x > 1: #check if its greater than one
                for y in range (2, int(math.sqrt(x)) + 1):
                    if x % y == 0: #if that specific number in the range can be divided by any number then it's not a prime number
                        break
                else:
                    listofprime.append(x) #add that number to the list to display later
        calctime2 = time.time() - time2 #calculate the CPU time for the algorithm
        #final print
        print('the prime numbers between ' + str(lowend) + ' and ' + str(highend) + ' are:')
        print(listofprime)
        print(calctime2)
print('Program exiting...')
```

Figure 8 - the full program code

```
Enter choice of algorithms to calculate prime numbers 1 or 2 & 0 for exiting:1
Algorithm 1:
Enter the lower end of the range: 10
Enter the higher end of the range: 1000
the prime numbers between 10 and 1000 are:
[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271,
277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431,
433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593,
599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929,
937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
0.004959821701049805
Enter choice of algorithms to calculate prime numbers 1 or 2 & 0 for exiting:2
Algorithm 2:
Enter the lower end of the range: 10
Enter the higher end of the range: 1000
the prime numbers between 10 and 1000 are:
[11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137,
139, 149, 151, 157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257, 263, 269, 271,
277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431,
433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593,
599, 601, 607, 613, 617, 619, 631, 641, 643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863, 877, 881, 883, 887, 907, 911, 919, 929,
937, 941, 947, 953, 967, 971, 977, 983, 991, 997]
0.0009920597076416016
Enter choice of algorithms to calculate prime numbers 1 or 2 & 0 for exiting:0
Program exiting...
```

Figure 9 - output example of the full program with range (10 : 1000) as input

Results and findings:

Table 1 - Testing and results table.

Inputs	Algorithm (1)	Algorithm (2)
1 : 10	15 iterations	8 iterations
1 : 100	1133 iterations	236 iterations
1 : 1000	78022 iterations	5288 iterations

We can see from **Figure-9** that the second algorithm or Algorithm (2) is significantly faster in terms of time in comparison to Algorithm (1). The reason behind that is simply the number of iterations of each algorithm. After some testing, we added a counter for each algorithm, with the range input of (1 : 1000), (1 : 100) and (1 : 10). We found that the number of iterations of each algorithm differs depending on the input range. In every input test, the resulting number of iterations is always greater in the first algorithm, unlike the second algorithm which iterates much less due to its better checking method. Unfortunately, we could not come up with a single equation for the number of iterations for each algorithm. **Table-1** shows the difference between the two algorithms.

Discussion:

In this project we learned and understood the prime numbers and the effective way to calculate them, and their applications in real-life such as encryption and mechanical gears, afterwards by using python programming language with its built-in libraries we implement the methods to calculate prime numbers into two different algorithms and then measuring the execution time on the CPU for both algorithms to find the most efficient algorithm which was Algorithm (2) . To our knowledge, our work is correct to some extent, the IDE we are using couldn't process more than 7-digit prime number in a reasonable time. With this application, we could easily know all the prime numbers in a given range to do our needed work as computer engineering students. This project can help us in the near future for other projects and help us become better future engineers.

Reference:

[1] Course book : Discrete Mathematical Structures. Sixth Edition.

[2] Stibitz, G. (1938). An Application of Number Theory to Gear Ratios. The American Mathematical Monthly, 45(1), 22-31. doi:10.2307/2303469

[3] Top 10 reasons why python is so popular with developers.

https://www.upgrad.com/blog/reasons-why-python-popular-with-developers/#:~:text=The%20python%20language%20is%20one,faster%20than%20other%20programming%20languages.