A method of generating prime numbers

Software Engineering Group 5
Software Engineering Lab 2

Introduction

A prime number is a whole number greater than 1 whose only factors are 1 and itself[1]. This report employs an algorithm, namely the Sieve of Eratosthenes, which was coded in python, to generate a list of all prime numbers less than or equal to an inputted number. Testing will then be done to check the correctness of the output and algorithm use.

Project Requirements

A System needs to be designed, employing an algorithm, and implemented that generates and outputs ta list of prime numbers less than or equal to an imputed number. The following requirements must be met by the final System:

- The System should allow the user the input a number greater than 1.
- The System should store the given number.
- The System should compute a list of prime numbers using the Sieve of Eratosthenes algorithm that are less than or equal to the inputted number.
- The System should output the list generated by the Sieve of Eratosthenes algorithm.
- The System should test that the algorithm produces the correct results during various scenarios.

Project Description

Core Algorithms

Sieve of Eratosthenes

Sieve of Eratosthenes is a simple and ancient algorithm used to find the prime numbers up to any given limit. It is one of the most efficient ways to find small prime numbers[2].

The idea is to find numbers in the table that are multiples of a number and therefore composite, to discard them as prime. The numbers that are left will be prime numbers[3].

Use case narrative

A user opens the "Lab2.py" file using a terminal. The terminal then runs the program, which asks the user to input a number greater than 1 of their choice. Once inputted, the program then runs and computes a list of prime numbers less than or equal to the given inputted number. The output is then shown in an array to the user, using the terminal as an interface.

Testing

Test Description:

Due to the scope of the testing only being one module, it is unnecessary to perform integration testing, as such, unit testing has been performed on this module with checks on:

1. Given valid input, is the correct output produced?

2. Given edge cases as input, is the correct output produced?

3. Given invalid input, are the correct exceptions raised?

Scenarios

1. Non-perfect square X

Due to the optimization made to only search for primes less than the square root of x, it is important to test the edge cases when x is not a perfect square, this detects a potential off-by-one error.

Input given: x = 80;

Expected Output datatype: List

Expected Output: [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79];

2. Input not a number

This scenario tests whether the algorithm can raise the correct exceptions for invalid input.

Input given: '100'

Expected Output: TypeError Raised

Input given: 'six'

Expected Output: TypeError Raised

3. Too large input - time taken more than

This scenario tests whether the algorithm will time out if too large input is selected.

4. Below 2 input

This scenario tests various edge cases where there are no prime numbers produced by the algorithm.

Input given: 1

Expected Output: ValueError Raised

Input given: 0

Expected Output: ValueError Raised

Input given: -1

Expected Output: ValueError Raised

5. Correct output - small number

This scenario tests the correctness for valid small input

Input given: 3

Expected Output Datatype: List

Expected Output: [2,3]

6. Correct output - large number

This scenario tests the correctness for valid large input

Input given: 100

Expected Output Datatype: List

Expected Output: [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]

Integration and System testing

Due to the system consisting of only one module, there is no system integration. Thus, testing the system itself is deemed unnecessary.

Possible Improvements

Appendix A: Member responsibilities

	Person Responsible			
Task	**Kenan**	**Marc**	**Preshen**	**Shaylin**
Algorithm design				
Algorithm implementation				
Testing design and implementation				
Report structuring and creation				