**Lab Assignment (2)**

**1. Problem Description (prime number):**

A prime number (or a prime) is a natural number greater than 1 that is not a product of two smaller natural numbers. We want to know the number of prime numbers less than 1,000,000.

In this problem, you are required to implement five methods to solve the problem according to the following instructions. You need to return the number of prime numbers and executing time of each method. You can refer to this template:

import time

def PN\_X(n):

    time\_start = time.time()

number\_pn = 0

#Your code

    time\_end = time.time()

    return time\_end - time\_start, number\_pn

**2. Brute Force**

In the first method, we judge whether a number is prime according to the definition. In other words, whether a number can be divided by some number greater than 1 with no remainder. You need to check each number less than 1,000,000. (If your program runs for more than 5 minutes, you can directly stop it.)

**3. Optimize Brute Force**

You must notice that brute force method is quite time-consuming. So, we need to optimize the method.

If a number is a product of two smaller natural numbers , it is obvious that and . Use this idea to optimize brute force method and record its execution time.

**4. Optimize Factor**

Notice that .

So, a prime should be in the form of or . Use this conclusion to optimize the range of factor.

**5. Sieve of Eratosthenes**

In the last method, we use the fact that the multiples of 2 and 3 cannot be prime. The basic idea of **Sieve of Eratosthenes** is to extend this fact to more numbers. Starting with the first prime number, 2, iteratively remove the multiples of each prime. The remaining numbers must be prime. Implement **Sieve of Eratosthenes** by yourself.

**6. Miller-Rabin**

The **Miller-Rabin primality test** is a primality test: an algorithm which determines whether a given number is likely to be prime. When the prime number is quite large, **Miller-Rabin primality test** is very efficient.

It is based on **Fermat's little theorem (FLT)**, that for a prime number :

where and are coprime. (**Necessary condition 1)**

For any prime number  , could be written as

where is an integer and is odd. Thus,

**Notice that:**

So to compute , we could start with , and then compute Finally, (All the products here are based on mod n)

**Theorem.** When p is prime, if , then .

Thus, if there exists an integer , , such that (mod n) holds, then (mod) should also hold. (**Necessary condition 2**)

This condition does not hold for co-prime number: let satisfies (mod 8). Yet, and (mod 8). So, you should always check Condition 2.

When we talked about mod, is identical to =. However, take mod could save lot of time as the arithmetic operations are conducted on numbers less than

The converse is not true. There exists some which is not prime, but still holds for some a. So, you may need to choose several to rule out these possibilities.

Implement **Miller-Rabin primality test** by yourself.

Hints: for , choose all the primes [2, 3, 5, 7, ….] in 0~100 is sufficient. If your implementation is correct, then [2, 3] should return the correct answer.

**Extra**

You can try judging whether a number larger than is prime and compare the execution time of all the methods. For example: 1000000000000037 is a prime.

**Submit your code including the implementation of the five methods above on Canvas before the deadline.**

**DDL 2022-11-29 23:59**

**No Latter is allowed!!!!**