Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский институт)»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления
КАФЕДРА ИУ7

Отчёт

по РК №1

Дисциплина: Анализ алгоритмов

Студент гр. ИУ7-56Б _____ **Хеламбаге Г.С.**
(Подпись, дата) (И.О. Фамилия)

Преподаватель _____ **Волкова Л.Л.**
(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

Введение

1. **Определение алгоритма эратосфена**
2. **Постановка задачи**
3. **Решение**
4. **Заключение**

1. Определение алгоритма эратосфена

математике «Сито Эратосфена» представляет собой простой древний алгоритм поиска всех простых чисел вплоть до любого заданного предела.

Это делается путем итеративной маркировки как составных (то есть, не простых) кратных каждого простого числа, начиная с первого простого числа, 2. Кратные значения данного простого числа генерируются как последовательность чисел, начинающихся с этого простого числа, с постоянной разницей между ними, равными этому простому числу. Это ключевое отличие решета от использования пробного деления для последовательной проверки каждого числа кандидатов на делимость на каждое простое число.

Самое раннее известное упоминание о сите (древнегреческое: κόσκινον Ἐρατοσθένους, kóskinon Eratosthénous) содержится в «Никомахе» из «Введения в арифметику» Герасы, которое описывает его и приписывает его Эратосфену Киренскому, греческому математику**.**

Обзор

Простое число - это натуральное число, которое имеет ровно два различных делителя натуральных чисел: 1 и себя.

Чтобы найти все простые числа, меньшие или равные данному целому числу n по методу Эратосфена:

1.) Создайте список последовательных целых чисел от 2 до n: (2, 3, 4, ..., n).

2.) Вначале пусть p равно 2, наименьшему простому числу.

3.) Перечислите кратные p, посчитав приращения p от 2p до n, и отметьте их в списке (это будут 2p, 3p, 4p, ...; само p не должно быть отмечено).

4.) Найдите первое число больше p в списке, который не отмечен. Если такого номера не было, остановитесь. В противном случае, пусть теперь

p равно этому новому числу (которое является следующим простым числом), и повторите с шага 3.

5.) Когда алгоритм завершает работу, числа, оставшиеся не отмеченными в списке, представляют собой все простые числа ниже n.

Основная идея здесь заключается в том, что каждое значение, данное p, будет простым, потому что если бы оно было составным, оно было бы помечено как кратное некоторого другого, меньшего простого числа. Обратите внимание, что некоторые номера могут быть отмечены более одного раза (например, 15 будут отмечены как для 3 и 5).

В качестве уточнения достаточно отметить числа на шаге 3, начиная с p2, так как все меньшие кратные p будут уже отмечены в этой точке. Это означает, что алгоритму разрешено завершать на шаге 4, когда p2 больше, чем n.

Другое уточнение состоит в том, чтобы первоначально перечислять только нечетные числа (3, 5, ..., n) и считать с шагом 2p от p2 на шаге 3, таким образом отмечая только нечетные кратные числа p. Это фактически появляется в оригинальном алгоритме.Это можно обобщить с помощью факторизации колеса, формируя начальный список только из чисел, взаимно простых с первыми несколькими простыми числами, а не только из шансов (то есть чисел, взаимно совпадающих с 2), и считая в соответствующих скорректированных приращениях, так что только такие кратные p равны генерируются, которые взаимно просты с этими маленькими простыми числами, в первую очередь.

Пример понять

Чтобы найти все простые числа, меньшие или равные 30, выполните следующие действия.

Сначала создайте список целых чисел от 2 до 30:

```
2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30
```

Первое число в списке - 2; вычеркните каждое 2-е число в списке после 2, считая от 2 с шагом 2 (это будет все кратные 2 в списке):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ 9 ~~10~~ 11 ~~12~~ 13 ~~14~~ 15 ~~16~~ 17 ~~18~~ 19 ~~20~~ 21 ~~22~~ 23 ~~24~~ 25 ~~26~~ 27 ~~28~~ 29 ~~30~~

Следующее число в списке после 2 - 3; вычеркните каждое 3-е число в списке после 3, посчитав от 3 с шагом 3 (это будет все кратные 3 в списке):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ 25 ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~

The next number not yet crossed out in the list after 3 is 5; cross out every 5th number in the list after 5 by counting up from 5 in increments of 5 (i.e. all the multiples of 5):

2 3 ~~4~~ 5 ~~6~~ 7 ~~8~~ ~~9~~ ~~10~~ 11 ~~12~~ 13 ~~14~~ ~~15~~ ~~16~~ 17 ~~18~~ 19 ~~20~~ ~~21~~ ~~22~~ 23 ~~24~~ ~~25~~ ~~26~~ ~~27~~ ~~28~~ 29 ~~30~~

Следующее число, которое еще не вычеркнуто в списке после 5, равно 7; следующим шагом было бы вычеркнуть каждое 7-е число в списке после 7, но все они уже вычеркнуты в этой точке, так как эти числа (14, 21, 28) также кратны меньшим простым числам, потому что 7 × 7 больше чем 30. Числа, не вычеркнутые в этом пункте в списке, являются всеми простыми числами ниже 30:

2 3 5 7 11 13 17 19 23 29

2.Постановка задачи

Реализовать проверку натурального числа н на просьту.

1) Решить эратофена
2) Любыми другая проверка на простьту
    a) Оценить память(аналитически либо замерить)
    b) Замерить время для N=10,...,100

3. Решение

```
from time import perf_counter
# measuring time
def getProcessTime(N):
    t1_start = perf_counter()
    sieve_of_eratosthenes(N)
    t1_stop = perf_counter()
    print("Elapsed time:",t1_stop - t1_start)

    t2_start = perf_counter()
    normal_method(N)
    t2_stop = perf_counter()
    print("Elapsed time:",t2_stop - t2_start)

#using normal method to find prime numbers

def normal_method(N):
    print('using method 2->')
    for k in range(0,N):
        flag = 0
        if k > 1:
            for i in range(2,k):
                if(k % i) == 0:
                    flag = 1
                    break
            if(flag == 0):
              print(k)

#eratosthenes algorithm to find prime numbers

def sieve_of_eratosthenes(n):
    print('using method 1->')
```

```
    is_prime = [True]*(n-1) #create array
    p = 2 # 2 as first prime number

    while True:
        multiplier = 2
        multiple = p * multiplier # (2p,3p,4p,,,)

        while multiple <= n:
            is_prime[multiple- 2] = False
            multiplier += 1
            multiple = p * multiplier

        for i,prime in enumerate(is_prime):
            if prime and i+2>p:
                p = i + 2
                break

        else:
            break

    for i,prime in enumerate(is_prime):
        if prime:
            print (i+2)

    #print('Total time : %0.5f' %(end_time-start_time))

def getoutput(N):
    getProcessTime(N)

def main():
    N = int(input('Enter the quantity of natural numbers:'))
    getoutput(N)
```

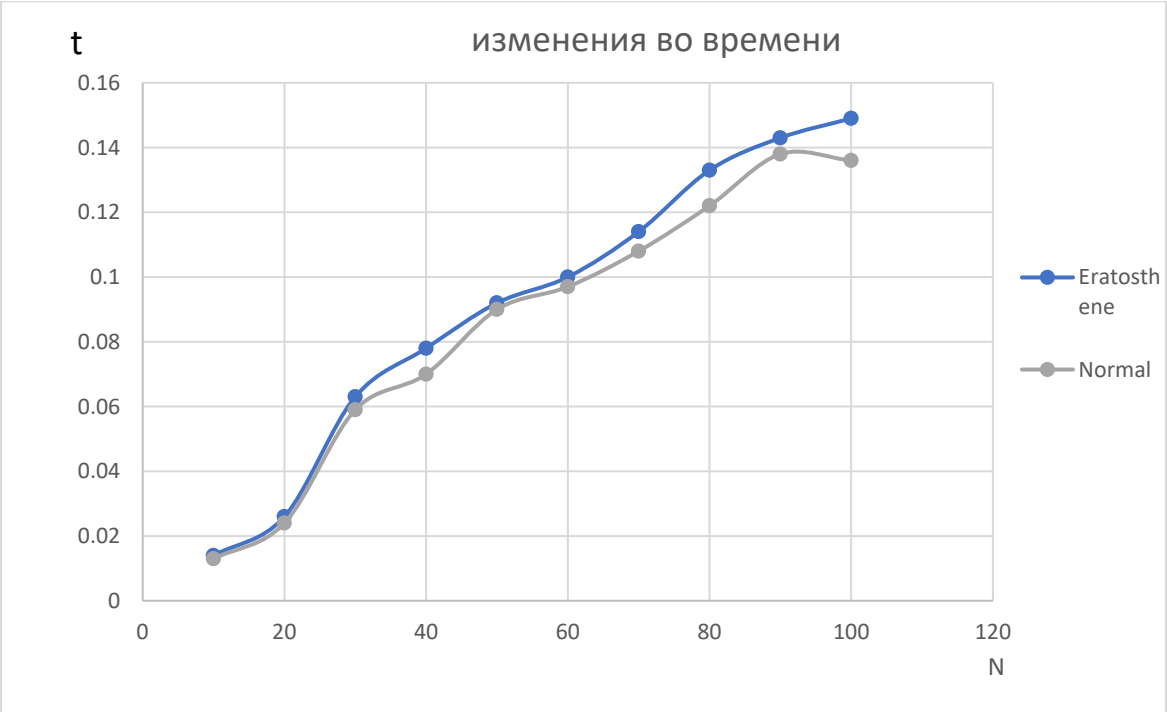
```
if __name__ == '__main__':
    main()
```

пример до 10 натуральных чисел

```
Enter the quantity of natural numbers:10
using Eratosthene Algorithm->
2
3
5
7
Elapsed time to eratosthene: 0.014227799999999999
using Normal Algorithm->
2
3
5
7
Elapsed time normal : 0.012828199999999998
>>>
```

измерение времени

количество натуральных чисел(N)	алгоритм эратостене	алгоритм нормальный
10	0.014	0.013
20	0.026	0.024
30	0.063	0.059
40	0.078	0.070
50	0.092	0.090
60	0.100	0.097
70	0.114	0.108
80	0.133	0.122
90	0.143	0.138
100	0.149	0.136


t
изменения во времени
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
0
20
40
60
80
100
120
N
Eratosthene
Normal

4.Заключение

Цель этого задания была достигнута. В ходе работы были изучены алгоритмы нахождения простых чисел с использованием алгоритма Эратостена и любого другого алгоритма. Эратосфен и нормальные алгоритмы сравниваются. Изучены зависимости времени выполнения алгоритмов от количества натуральных чисел. Также были реализованы 2 описанных алгоритма для нахождения простых чисел.