ФГБОУ ВО «Сочинский государственный университет»

Факультет инновационных, инженерных и цифровых технологий

Кафедра информационных технологий и математики

**ОТЧЕТ**

о выполнении лабораторной работы №7

*«Python. Сортировка и поиск»*

*Вариант 4*

по дисциплине **«Алгоритмизация и программирование»**

***Выполнил:***

студент 1 курса гр. 24-ПИЦ

Нерсесян С. В.

«\_*15*\_»\_\_\_\_*Февраля*\_\_\_\_2025г

***Проверил:***

проф. каф. инф. тех., д.т.н.

Попов Дмитрий Иванович

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_202-г

Оценка\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Отзыв, замечания\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**ОБЩИЕ ЗАДАНИЯ ДЛЯ ВСЕХ ВАРИАНТОВ:**

№1.

Исследование сложности (скорости выполнения) алгоритмов сортировки.

1.1. Отладьте три функции сортировки массивов: сортировка обменом, сортировка выбором, сортировка вставками.

1.2. Проведите замеры скорости выполнения каждого из алгоритмов с различными значениями N – размером массива и зафиксируйте время выполнения в таблице.

1.3. По таблице постройте три графика зависимостей и сделайте выводы о сложности различных алгоритмов сортировки.

Сортировка обменом:



from datetime import \*

from random import \*

def BubbleSort(n):

    a=n

    t=int(datetime.now().timestamp() \* 1000)

    n=len(a)

    for i in range(n - 1):

        for j in range(n - 1 - i):

            if a[j] > a[j+ 1]:

                a[j], a[j + 1] = a[j + 1], a[j]

    t=int(datetime.now().timestamp() \* 1000)-t

    print("Сортировка обменом:")

    print("Время (мс):", t)

n=int(input("Введите N: "))

r = [randint(1, 1000) for \_ in range(n)]

BubbleSort(r)







Сортировка выбором:



from datetime import \*

from random import \*

def SelectionSort(n):

    a=n

    t=int(datetime.now().timestamp() \* 1000)

    for i in range(0, len(a) - 1):

        smallest = i

        for j in range(i + 1, len(a)):

            if a[j] < a[smallest]:

                smallest = j

        a[i], a[smallest] = a[smallest], a[i]

    t=int(datetime.now().timestamp() \* 1000)-t

    #print (a)

    print("Сортировка выбором:")

    print("Время (мс):", t)

n=int(input("Введите N: "))

r = [randint(1, 1000) for \_ in range(n)]

SelectionSort(r)







Сортировка вставками:



from datetime import \*

from random import \*

def InsertionSort(n):

    a=n

    t=int(datetime.now().timestamp() \* 1000)

    for i in range(1, len(a)):

        temp = a[i]

        j = i - 1

        while (j >= 0 and temp < a[j]):

            a[j + 1] = a[j]

            j = j - 1

        a[j + 1] = temp

    t=int(datetime.now().timestamp() \* 1000)-t

    #print(a)

    print("Сортировка вставками:")

    print("Время (мс):", t)

n=int(input("Введите N: "))

r = [randint(1, 1000) for \_ in range(n)]

InsertionSort(r)







Таблица скорости выполнения:

|  |  |  |  |
| --- | --- | --- | --- |
| N | Время выполнения, сортировка обменом, мс | Время выполнения, сортировка выбором, мс | Время выполнения, сортировка вставками, мс |
| 100 | 2 | 0 | 0 |
| 200 | 2 | 1 | 0 |
| 400 | 4 | 8 | 4 |
| 500 | 19 | 8 | 5 |
| 1000 | 68 | 31 | 34 |
| 1500 | 167 | 68 | 77 |
| 2000 | 291 | 123 | 137 |
| 3000 | 666 | 272 | 311 |
| 5000 | 1917 | 761 | 865 |
| 10000 | 7928 | 3051 | 3594 |
| 15000 | 21995 | 6939 | 8096 |
| 20000 | 34545 | 12311 | 14206 |
| 30000 | 69528 | 29287 | 36837 |
| 50000 | - | 78746 | 104786 |
| 70000 | - | - | - |
| 90000 | - | - | - |
| 100000 | - | - | - |

Графики зависимости:

Вывод:

Как можно заметить, самым медленным способом сортировки оказалась сортировка обменом, а самым быстрым – сортировка выбором. Средние результаты показала сортировка вставками.

№2.

Разработайте и отладьте функции для алгоритмов поиска методом грубой силы, методом Бойера-Мура, бинарного поиска.

Метод грубой силы:



def BFSearch(str, pattern):

    for i in range(0,len(str)-len(pattern)+1):

        for j in range(0,len(pattern)):

            if (pattern[j] != str[i + j]):

                break

            else:

                if (j == len(pattern)-1):

                    return True

    return False

s=str(input("Введите строку: "))

p=str(input("Введите паттерн: "))

print("Паттерн был найден!" if BFSearch(s, p) else "Паттерн не был найден!")





Метод Бойера-Мура:



def forming\_d(pattern):

    d = [len(pattern) for i in range(1105)]

    new\_p = pattern[::-1]

    for i in range(len(new\_p)):

        if d[ord(new\_p[i])] != len(new\_p):

            continue

        else:

            d[ord(new\_p[i])] = i

    return d

def BMSearch(string, pattern):

    d = forming\_d(pattern)

    len\_p = x = j = k = len(pattern)

    counter = 0

    while x <= len(string) and j > 0:

        if pattern[j - 1] == string[k - 1]:

            j -= 1

            k -= 1

        else:

            x += d[ord(string[k - 1])]

            k = x

            j = len\_p

            counter += 1

    if j <= 0:

        return True

    else:

        return False

s=str(input("Введите строку: "))

p=str(input("Введите паттерн: "))

print("Паттерн был найден!" if BMSearch(s, p) else "Паттерн не был найден!")





Метод бинарного поиска:



def BinarySearch(pArr, lb, ub, key):

    while (lb <= ub):

        m = int( (lb + ub) / 2)

        if (key < pArr[m]):

            ub = m - 1

        else:

            if (key > pArr[m]):

                lb = m + 1

            else:

                return True

    return False

m=list(map(str, str(input("Введите массив: ")).split()))

k=str(input("Введите ключ: "))

print("Ключ был найден!" if BinarySearch(m, 0, len(m)-1, k) else "Ключ не был найден!")



