

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3
по курсу «Алгоритмы и структуры данных»
Тема: Быстрая сортировка, сортировки за линейное время
Вариант 11

Выполнил:
Лютый Никита Артемович
К3140

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Улучшение Quick sort	3
Задача №7. Цифровая сортировка	8
Задача №8. К ближайших точек к началу координат	13
Дополнительные задачи	18
Задача №3. Сортировка пугалом	18
Задача №4. Точки и отрезки	23
Задача №5. Индекс Хирша	28
Вывод	33

Задачи по варианту

Задача №1. Улучшение Quick sort

1. Используя *псевдокод* процедуры Randomized - QuickSort, а также Partition из презентации к Лекции 3 (страницы 8 и 12), напишите программу быстрой сортировки на Python и проверьте ее, создав несколько рандомных массивов, подходящих под параметры:

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^4$) — число элементов в массиве. Во второй строке находятся n различных целых чисел, *по модулю* не превосходящих 10^9 .
- **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Для проверки можно выбрать наихудший случай, когда сортируется массив рамера 10^3 , 10^4 , 10^5 чисел порядка 10^9 , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний - случайный. Сравните на данных сетах Randomized-QuickSort и простой QuickSort. (А также есть Median-QuickSort, см. задание 10.2; и Tail-Recursive-QuickSort, см. [Кормен. 2013, стр. 217](#))

2. **Основное задание.** Цель задачи - переделать данную реализацию рандомизированного алгоритма быстрой сортировки, чтобы она работала быстро даже с последовательностями, содержащими много одинаковых элементов. Чтобы заставить алгоритм быстрой сортировки эффективно обрабатывать последовательности с несколькими уникальными элементами, нужно заменить двухстороннее разделение на трехстороннее (смотри в Лекции 3 слайд 17). То есть ваша новая процедура разделения должна разбить массив на три части:

- $A[k] < x$ для всех $\ell + 1 \leq k \leq m_1 - 1$
- $A[k] = x$ для всех $m_1 \leq k \leq m_2$
- $A[k] > x$ для всех $m_2 + 1 \leq k \leq r$
- Формат входного и выходного файла аналогичен п.1.
- Аналогично п.1 этого задания сравните Randomized-QuickSort + c Partition и ее с Partition3 на сетах случайных данных, в которых содержатся всего несколько уникальных элементов при $n = 10^3, 10^4, 10^5$. Что быстрее, Randomized-QuickSort + c Partition3 или Merge-Sort?
- Пример:

input.txt	output.txt
5	2 2 2 3 9
2 3 9 2 2	

Листинг кода

```
from random import randint

from lab3.utils import read_file, check_inp, write_file

def patrition(lst, l, r):
    x = lst[l]
    j=l
    h=l
    for i in range(l+1,r+1):
        if lst[i]<x:
            h+=1
            j+=1
            if h!=j:
                lst[h], lst[i] = lst[i], lst[h]
                lst[h], lst[j] = lst[j], lst[h]
        else:
            lst[i],lst[j] = lst[j],lst[i]
    elif lst[i]==x:
        h+=1
        lst[i], lst[h] = lst[h], lst[i]

    lst[l], lst[j] = lst[j], lst[l]
    return (j, h)

def randomized_quicksort(lst, l, r):
    if l<r:
        k = randint(l,r)
        lst[l], lst[k] = lst[k], lst[l]
        (m1,m2) = patrition(lst, l, r)
        randomized_quicksort(lst, l, m1-1)
        randomized_quicksort(lst, m2+1, r)

def main():
    read_inp = read_file("../txtf/input.txt", 1)
    n = read_inp[0][0]
    lst = read_inp[1]

    max_n = 10**4
    max_el = 10**9
    check_inp(max_n, max_el, n, lst, [], 1)

    randomized_quicksort(lst, 0, n-1)
    result = ' '.join(map(str,lst))
    write_file("../txtf/output.txt", result)

main()
```

Тесты:

1) Тест на время работы:

```
from lab3.task1.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab3.task1.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализовано 3 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и перезаписывает переменную lst, применяя к ней процедуру сортировки randomized_quicksort(). Формирует выходное сообщение, которое далее записывает в файл output.txt.

2) randomized_quicksort()

Выбирает случайный индекс, от которого начнется сортировка, меняет крайние элементы местами и вызывает функцию patrition(), из которой получает 2 индекса: m1 – индекс окончания части, в которой элементы меньше выбранного, m2 – равны выбранному. Вызывает себя рекурсивно, меняя индексы.

3) patrition()

Проходит по списку и меняет его, перемещая элементы, меньше выбранного в начало, а равные выбранному – в середину, после чего возвращает индексы окончаний этих промежутков.

Тесты:

- 1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

input.txt	output.txt
1	5
2	2 3 9 2 2

input.txt	output.txt
1	2 2 2 3 9

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

input.txt	output.txt
1	1
2	1

input.txt	output.txt
1	1

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_time_main.py
Время работы программы 0.0005183999892324209 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_memory_main.py
Максимально занимаемая память: 17.646484375 KB

2) Значения из примера:

input.txt	output.txt
1	5
2	2 3 9 2 2

input.txt	output.txt
1	2 2 2 3 9

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_time_main.py
Время работы программы 0.0005462000262923539 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_memory_main.py
Максимально занимаемая память: 17.677734375 KB

3) Максимальные значения:

```
⌵ input.txt × ⌵ output.txt 🐍 utils.py 🐍 main.py 🐍 test_time_main.py 🐍 test_memory_main.py :  
1 10000 ✓  
2 1000000000 999999999 999999998 999999997 999999996 999999995 999999994 999999993 999999992 999999991 999  
  
⌵ input.txt ⌵ output.txt × 🐍 utils.py 🐍 main.py 🐍 test_time_main.py 🐍 test_memory_main.py :  
1 999990001 999990002 999990003 999990004 999990005 999990006 999990007 999990008 999990009 999990010 9 ✓  
  
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_time_main.py  
Время работы программы 0.017550799995660782 секунд.  
  
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task1\tests\test_memory_main.py  
Максимально занимаемая память: 1104.1845703125 KB
```

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.00052	17.646
Пример из задачи	0.00055	17.678
Верхняя граница диапазона значений входных данных из текста задачи	0.01755	1104.185

Вывод по задаче:

Быстрая сортировка является более быстрой, относительно рассмотренных в прошлых лабораторных, но затратной по памяти.

Задача №7. Число инверсий

Дано n строк, выведите их порядок после k фаз цифровой сортировки.

- **Формат входного файла (input.txt).** В первой строке входного файла содержатся числа n - число строк, m - их длина и k - число фаз цифровой сортировки ($1 \leq n \leq 10^6$, $1 \leq k \leq m \leq 10^6$, $n \cdot m \leq 5 \cdot 10^7$). Далее находится описание строк, но **в нетривиальном формате**. Так, i -ая строка ($1 \leq i \leq n$) записана в i -ых символах второй, ..., $(m + 1)$ -ой строк входного файла. Иными словами, строки написаны по вертикали. **Это сделано специально, чтобы сортировка занимала меньше времени.**

Строки состоят из строчных латинских букв: от символа "a" до символа "z" включительно. В таблице символов ASCII все эти буквы располагаются подряд и в алфавитном порядке, код буквы "a" равен 97, код буквы "z" равен 122.

- **Формат выходного файла (output.txt).** Выведите номера строк в том порядке, в котором они будут после k фаз цифровой сортировки.
- Ограничение по времени. 3 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt
3 3 1 bab bba baa	2 3 1
3 3 2 bab bba baa	3 2 1
3 3 3 bab bba baa	2 3 1

- **Примечание.** Во всех примерах входных данных даны следующие строки:
 - «bbb», имеющая индекс 1;
 - «aba», имеющая индекс 2;
 - «baa», имеющая индекс 3.

Разберем первый пример. Первая фаза цифровой сортировки отсортирует строки по последнему символу, таким образом, первой строкой окажется «aba» (индекс 2), затем «baa» (индекс 3), затем «bbb» (индекс 1). Таким образом, ответ равен «2 3 1».

Листинг кода

```
from lab3.utils import read_file, check_inp, write_file

def cif_sort(lst, k, n):
    for i in range(n):
        lst[i] = (lst[i][::-1], i+1)

    for i in range(k):
        lst = sorted(lst, key=lambda x: x[0][i])

    answ = ""
    for i in range(n):
        answ += str(lst[i][1])+' '

    return answ

def main():
    read_inp = read_file("../txtf/input.txt", 7)
    n = read_inp[0]
    m = read_inp[1]
    k = read_inp[2]
    lst = read_inp[3]

    max_n = 10**6
    max_el = 5*10**7
    check_inp(max_n, max_el, n, lst, [m, k], 7)

    result = cif_sort(lst, k, n)
    # result = solve(m, n, k, lst)
    write_file("../txtf/output.txt", result)

main()
```

Тесты:

1) Тест на время работы

```
from lab3.task7.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

2) Тест на занимаемую память

```
from lab3.task7.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

Текстовое объяснение решения:

Реализовано 2 функции:

1) `main()`:

Открывает файл `input.txt` и считывает из него `n`, `m`, `k` (по условию задачи) и сами элементы (записывает в переменную `lst`). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Вызывает функцию `cif_sort()`, результат ее работы записывает. Формирует выходное сообщение и записывает в `output.txt`.

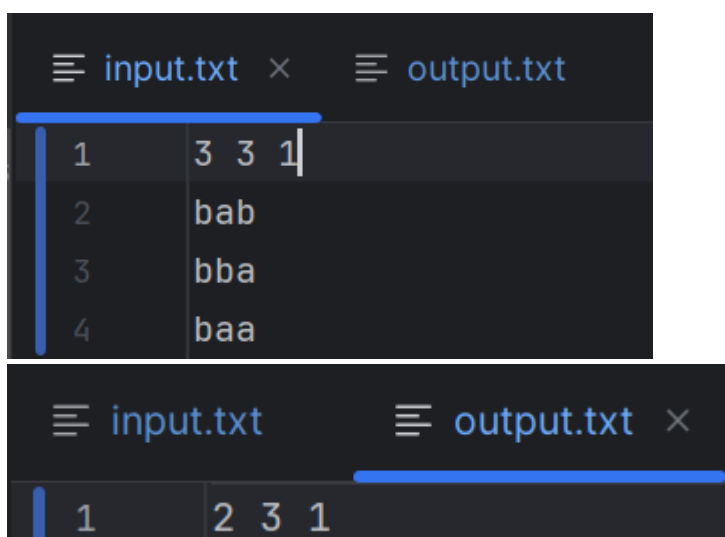
2) `cif_sort()`

Сначала она преобразует список элементов в новый, меняя местами строки и выдавая им индекс. Далее сортирует его по ключу, в зависимости от номера сортировки. Собирает ответ из индексов.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из задачи:



Результат работы кода на максимальных и минимальных значениях:

Значения из примера:

№1

```

≡ input.txt ×   ≡ output.txt
1      3 3 1
2      bab
3      bba
4      baa
  
```

```

≡ input.txt      ≡ output.txt ×
1      2 3 1
  
```

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_time_main.py
 Время работы программы 0.0004300999571569264 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_memory_main.py
 Максимально занимаемая память: 17.412109375 KB

№2

```

≡ input.txt ×
1      3 3 2
2      bab
3      bba
4      baa
  
```

```

≡ input.txt      ≡ output.txt ×
1      3 2 1
  
```

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_time_main.py
 Время работы программы 0.0004800999886356294 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_memory_main.py
 Максимально занимаемая память: 17.412109375 KB

№3

```

≡ input.txt ×   ≡ output.txt
1      3 3 3
2      bab
3      bba
4      baa
  
```

```

≡ input.txt      ≡ output.txt ×
1      2 3 1
  
```

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_time_main.py
 Время работы программы 0.0005326000391505659 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task7\tests\test_memory_main.py
 Максимально занимаемая память: 17.412109375 KB

	Время выполнения, с	Затраты памяти, KB
Пример из задачи №1	0.00043	17.4121

Пример из задачи №2	0.00048	17.4121
Пример из задачи №3	0.00053	17.4121

Вывод по задаче:

Цифровая сортировка является не очень быстрой и удобной.

Задача №8. К ближайших точек к началу координат.

В этой задаче, ваша цель - найти K ближайших точек к началу координат среди данных n точек.

- **Цель.** Заданы n точек на поверхности, найти K точек, которые находятся ближе к началу координат $(0, 0)$, т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками (x_1, y_1) и (x_2, y_2) равно $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
- **Формат ввода или входного файла (input.txt).** Первая строка содержит n - общее количество точек на плоскости и через пробел K - количество ближайший точек к началу координат, которые надо найти. Каждая следующая из n строк содержит 2 целых числа x_i, y_i , определяющие точку (x_i, y_i) . Ограничения: $1 \leq n \leq 10^5$; $-10^9 \leq x_i, y_i \leq 10^9$ - целые числа.
- **Формат выхода или выходного файла (output.txt).** Выведите K ближайших точек к началу координат в строку в квадратных скобках через запятую. Ответ вывести в порядке возрастания расстояния до начала координат. Если оно равно, порядок произвольный.
- Ограничение по времени. 10 сек.
- Ограничение по памяти. 256 мб.
- Пример 1.

input.txt	output.txt
2 1 1 3 -2 2	[-2,2]

- Пример 2.

input.txt	output.txt
3 2 3 3 5 -1 -2 4	[3,3],[-2,4]

Листинг кода:

```
from lab3.utils import read_file, check_inp, write_file
from random import randint

def patritition(lst, l, r):
    x = lst[l][2]
    j=l
    h=l
```

```

for i in range(l+1,r+1):
    if lst[i][2]<x:
        h+=1
        j+=1
    if h!=j:
        lst[h], lst[i] = lst[i], lst[h]
        lst[h], lst[j] = lst[j], lst[h]
    else:
        lst[i],lst[j] = lst[j],lst[i]
    elif lst[i][2]==x:
        h+=1
        lst[i], lst[h] = lst[h], lst[i]

lst[l], lst[j] = lst[j], lst[l]
return (j, h)

def randomized_quicksort(lst, l, r):
    if l<r:
        k = randint(l,r)
        lst[l], lst[k] = lst[k], lst[l]
        (m1,m2) = patrition(lst, l, r)
        randomized_quicksort(lst, l, m1-1)
        randomized_quicksort(lst, m2+1, r)

def get_answ(lst, k, n):
    randomized_quicksort(lst, 0, n-1)
    answ = ""
    for i in range(k):
        el = lst[i]
        answ += "["+str(el[0])+","+str(el[1])+"]"+","
    return answ[:-1]

def main():
    read_inp = read_file("../txtf/input.txt",8)
    n = read_inp[0]
    k = read_inp[1]
    lst = read_inp[2]

    max_n = 10**5
    max_el=10**9
    check_inp(max_n, max_el, n, [lst[i][0] for i in range(len(lst))], [k, [lst[i][1] for i in range(len(lst))]],8)

    write_file("../txtf/output.txt", get_answ(lst, k, n))

main()

```

Тесты:

1) Тест на время работы:

```

from lab3.task8.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab3.task8.src.main import main
import tracemalloc

```

```
tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализовано 4 функции:

1) main():

Открывает файл input.txt и считывает из него n, k и сами элементы (записывает в переменную lst) функцией read_file, которая к 2-м элементам списка добавляет 3-й – расстояние до начала координат. Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Формирует выходное сообщение, вызывая функцию get_answ(), которое далее записывает в файл output.txt.

2) get_answ():

Применяет к списку процедуру randomized_sort() и формирует строку – выходное сообщение.

3) randomized_sort():

Описано в задании №1, но сортировка идет по 3-му элементу i-го элемента списка.

4) patrition():

Описано в задании №1

Тесты:

- 1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

```
≡ input.txt × ≡ output.txt
1 2 1
2 1 3
3 -2 2
```

```
≡ input.txt ≡ output.txt ×
1 [-2, 2]
```

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

```
≡ input.txt × ≡ output.txt
1 1 1
2 0 0
```

```
≡ input.txt × ≡ output.txt ×
1 [0, 0]
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_time_main.py
Время работы программы 0.00045190000673756003 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_memory_main.py
Максимально занимаемая память: 17.6044921875 KB
```

2) Значения из примера:

```
≡ input.txt × ≡ output.txt
1 2 1
2 1 3
3 -2 2
```

```
≡ input.txt ≡ output.txt ×
1 [-2, 2]
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_time_main.py
Время работы программы 0.0004727000487037003 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_memory_main.py
Максимально занимаемая память: 17.759765625 KB
```


3) Максимальные значения:

The screenshot shows a code editor with three tabs: input.txt, output.txt, and main.py. The input.txt file contains a list of numbers from 1 to 28. The output.txt file shows a large list of memory addresses, likely generated by the algorithm. The main.py file is a Python script that tests the memory management algorithm. The script is located at C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_memory_main.py. The script is executed from the command line, and the output shows the execution time and memory usage.

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_time_main.py
Время работы программы 0.394362699971457 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task8\tests\test_memory_main.py
Максимально занимаемая память: 19838.73828125 KB
```

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.00045	17.604
Пример из задачи	0.00047	17.759
Верхняя граница диапазона значений входных данных из текста задачи	0.39436	19038.738

Вывод по задаче: реализованный алгоритм является довольно быстрым, но затратным по памяти

Задача №3. Сортировка пугалом

«Сортировка пугалом» — это давно забытая народная потешка. Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются растопырены руки, как у огородного пугала. Перед ним ставятся n матрёшек в ряд. Из-за палки единственное, что он может сделать — это взять в руки две матрёшки на расстоянии k друг от друга (то есть i -ую и $i + k$ -ую), развернуться и поставить их обратно в ряд, таким образом поменяв их местами.

Задача участника — расположить матрёшки по неубыванию размера. Может ли он это сделать?

- **Формат входного файла (input.txt).** В первой строчке содержатся числа n и k ($1 \leq n, k \leq 10^5$) – число матрёшек и размах рук. Во второй строчке содержится n целых чисел, которые по модулю не превосходят 10^9 – размеры матрёшек.
- **Формат выходного файла (output.txt).** Выведите «ДА», если возможно отсортировать матрёшки по неубыванию размера, и «НЕТ» в противном случае.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Примеры:

input.txt	output.txt
3 2 2 1 3	НЕТ
5 3 1 5 3 4 1	ДА

Листинг кода

```
from lab3.utils import read_file, check_inp, write_file

def sort_scarecrow(lst, n, k):
    lst_arrs = [[] for _ in range(k)]
    for i in range(n):
        ost = i % k
        lst_arrs[ost].append(lst[i])

    lst_arrs = list(map(sorted, lst_arrs))
    for i in range(n // k + 1):
        for j in range(k - 1):
            if len(lst_arrs[j]) > i and len(lst_arrs[j + 1]) > i:
                if lst_arrs[j][i] > lst_arrs[j + 1][i]:
                    return False

    return True
```

```
def main():
    read_inp = read_file("../txtf/input.txt", 2)
    n = read_inp[0][0]
    k = read_inp[0][1]
    lst = read_inp[1]

    max_n = 10**5
    max_el = 10**9
    check_inp(max_n, max_el, n, lst, [k], 2)

    if k != 1:
        l = sort_scarecrow(lst, n, k)
        if l:
            result = "YES"
        else:
            result = "NO"
    else:
        result = "YES"
    write_file("../txtf/output.txt", result)

main()
```

Тесты:

1) Тест на время работы:

```
from lab3.task3.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab3.task3.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: " + str(tracemalloc.get_traced_memory()[1]/1024) + " KB")
tracemalloc.stop()
```

Текстовое объяснение решения.

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n, k и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Проверяет частный случай при k=1. Если это не он, то

проводит сортировку и формирует выходное сообщение, которое далее записывает в файл output.txt.

2) `sort_scarecrow()`:

Делит элементы на n списков в зависимости от их остатка от деления. Затем сортирует получившийся список и проверяет сортировку. Отсортировано – возвращает True, нет – False.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

input.txt	output.txt
1	3 2
2	2 1 3

input.txt	output.txt
1	NO

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

input.txt	output.txt
1	1 1
2	1

input.txt	output.txt
1	YES

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_time_main.py
Время работы программы 0.00036800000816583633 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_memory_main.py
Максимально занимаемая память: 17.677734375 KB
```

2) Значения из примеров:

input.txt		output.txt	
1	3 2		
2	2 1 3		

input.txt		output.txt	
1	NO		

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_time_main.py
Время работы программы 0.00043370004277676344 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_memory_main.py
Максимально занимаемая память: 17.708984375 KB
```

3) Максимальные значения:

input.txt

output.txt

main.py

test_memory_main.py

test_time_main.py

This document contains very long lines. Soft wraps were enabled to improve editor performance.

Hide notification

Don't show again

1

100000 2

2

1000000000 99999999 99999998 99999997 99999996 99999995 99999994 99999993 99999992 99999991
99999990 99999989 99999988 99999987 99999986 99999985 99999984 99999983 99999982 99999981
99999980 99999979 99999978 99999977 99999976 99999975 99999974 99999973 99999972 99999971
99999970 99999969 99999968 99999967 99999966 99999965 99999964 99999963 99999962 99999961
99999960 99999959 99999958 99999957 99999956 99999955 99999954 99999953 99999952 99999951
99999950 99999949 99999948 99999947 99999946 99999945 99999944 99999943 99999942 99999941
99999940 99999939 99999938 99999937 99999936 99999935 99999934 99999933 99999932 99999931
99999930 99999929 99999928 99999927 99999926 99999925 99999924 99999923 99999922 99999921
99999920 99999919 99999918 99999917 99999916 99999915 99999914 99999913 99999912 99999911
99999910 99999909 99999908 99999907 99999906 99999905 99999904 99999903 99999902 99999901
99999900 99999899 99999898 99999897 99999896 99999895 99999894 99999893 99999892 99999891
99999890 99999889 99999888 99999887 99999886 99999885 99999884 99999883 99999882 99999881
99999880 99999879 99999878 99999877 99999876 99999875 99999874 99999873 99999872 99999871
99999870 99999869 99999868 99999867 99999866 99999865 99999864 99999863 99999862 99999861
99999860 99999859 99999858 99999857 99999856 99999855 99999854 99999853 99999852 99999851
99999850 99999849 99999848 99999847 99999846 99999845 99999844 99999843 99999842 99999841
99999840 99999839 99999838 99999837 99999836 99999835 99999834 99999833 99999832 99999831
99999830 99999829 99999828 99999827 99999826 99999825 99999824 99999823 99999822 99999821
99999820 99999819 99999818 99999817 99999816 99999815 99999814 99999813 99999812 99999811
99999810 99999809 99999808 99999807 99999806 99999805 99999804 99999803 99999802 99999801
99999800 99999799 99999798 99999797 99999796 99999795 99999794 99999793 99999792 99999791

input.txt		output.txt	
1	NO		

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_time_main.py
Время работы программы 0.02078540000366047 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\algorithms-and-data-structures\lab3\task3\tests\test_memory_main.py
Максимально занимаемая память: 9973.533203125 KB
```

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.00037	17.678
Пример из задачи	0.00043	17.709
Верхняя граница диапазона значений входных данных из текста задачи	0.02079	9973.533

Вывод по задаче: Данная сортировка довольно быстрая и не такая затратная по памяти, но не обрабатывает все случаи.

Задача №4. Точки и отрезки

Допустим, вы организовываете онлайн-лотерею. Для участия нужно сделать ставку на одно целое число. При этом у вас есть несколько интервалов последовательных целых чисел. В этом случае выигрыш участника пропорционален количеству интервалов, содержащих номер участника, минус количество интервалов, которые его не содержат. (В нашем случае для начала - подсчет только количества интервалов, содержащих номер участника). Вам нужен эффективный алгоритм для расчета выигрышей для всех участников. Наивный способ сделать это - просто просканировать для всех участников список всех интервалов. Однако ваша лотерея очень популярна: у вас тысячи участников и тысячи интервалов. По этой причине вы не можете позволить себе медленный наивный алгоритм.

- **Цель.** Вам дается набор точек и набор отрезков. Цель состоит в том, чтобы вычислить для каждой точки количество отрезков, содержащих эту точку.
- **Формат входного файла (input.txt).** Первая строка содержит два неотрицательных целых числа s и p . s - количество отрезков, p - количество точек. Следующие s строк содержат 2 целых числа a_i, b_i , которые определяют i -ый отрезок $[a_i, b_i]$. Последняя строка определяет p целых чисел - точек x_1, x_2, \dots, x_p . Ограничения: $1 \leq s, p \leq 50000$; $-10^8 \leq a_i \leq b_i \leq 10^8$ для всех $0 \leq i < s$; $-10^8 \leq x_i \leq 10^8$ для всех $0 \leq j < p$.
- **Формат выходного файла (output.txt).** Выведите p неотрицательных целых чисел k_0, k_1, \dots, k_{p-1} , где k_i - это число отрезков, которые содержат x_i . То есть,

$$k_i = |\{j : a_j \leq x_i \leq b_j\}|.$$

- Пример 1.

input.txt	output.txt
2 3	1 0 0
0 5	
7 10	
1 6 11	

Здесь, у нас есть 2 отрезка и 2 точки. Первая точка принадлежит интервалу $[0, 5]$, остальные точки не принадлежат ни одному из данных интервалов.

- Пример 2.

input.txt	output.txt
1 3	0 0 1
-10 10	
-100 100 0	

- Пример 3.

input.txt	output.txt
3 2	2 0
0 5	
-3 2	
7 10	
1 6	

Листинг кода

```
from lab3.utils import read_file, check_inp, write_file

def find_kol(s,k,lst_int, lst_us):
    lst = []
    for i in range(s):
        lst.append((lst_int[i][0], -1))
        lst.append((lst_int[i][1], -2))
```



```

for i in range(k):
    lst.append((lst_us[i], i))

lst = sorted(lst)
length = len(lst)
cnt = 0

for i in range(length):
    if lst[i][1] == -1:
        cnt += 1
    elif lst[i][1] == -2:
        cnt -= 1
    else:
        lst_us[lst[i][1]] = cnt
return lst_us

def main():
    s, k, lst_int, lst_us = read_file('input.txt', 4)

    max_n = 50000
    max_el = 10 ** 8
    check_inp(max_n, max_el, s, lst_int, [k, lst_us], 4)

    lst_answ = find_kol(s, k, lst_int, lst_us)

    result = ' '.join(map(str, lst_answ))

    write_file("../txtf/output.txt", result)

main()

```

Тесты:

1) Тест на время работы:

```

from lab3.task4.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab3.task4.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: " + str(tracemalloc.get_traced_memory()[1]/1024) + " KB")
tracemalloc.stop()

```

Текстовое объяснение решения.

Реализовано 2 функции:

1) `main()`:

Открывает файл `input.txt` и считывает из него `s` (количество отрезков), `k` (количество точек), `k` (длина списка элементов для поиска), `lst_int` (список отрезков), `lst_us` (список точек). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). В переменную `lst_answ` записывает результат работы функции `find_kol()`. Затем формирует выходное сообщение, которое далее записывает в файл `output.txt`.

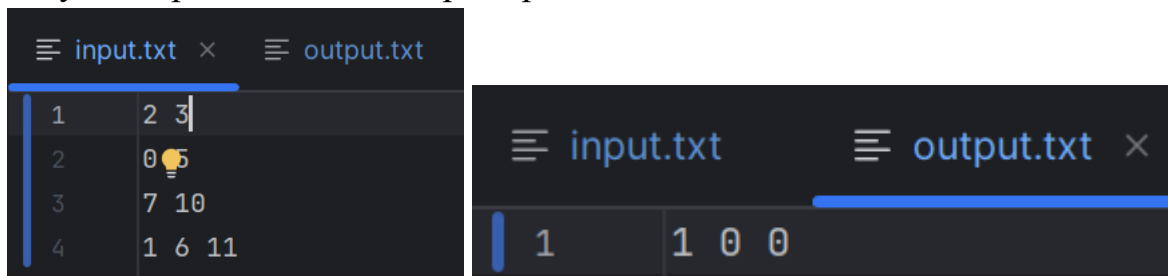
2) `find_kol()`:

Каждый элемент `lst_int` преобразует в кортеж с кодом: -1 – Начало отрезка, -2 – конец отрезка и записывает в `lst`. Далее в `lst` записывает точки, с их индексом. Сортирует список, а затем проходит по нему и считает количество открытых, но не закрытых отрезков.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:



```
input.txt x  output.txt
1 2 3
2 0 5
3 7 10
4 1 6 11

output.txt
1 1 0 0
```

Результат работы кода на максимальных и минимальных значениях:

- 1) Минимальные значения:

input.txt
output.txt

1	1 1
2	0 2
3	1

input.txt
output.txt

1	1
---	---

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_time_main.py
Время работы программы 0.0009091999381780624 секунд.

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_memory_main.py
Максимально занимаемая память: 17.634765625 KB

2) Значения из примера:

input.txt
output.txt

1	2 3
2	0 5
3	7 10
4	1 6 11

input.txt
output.txt

1	1 0 0
---	-------

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_time_main.py
Время работы программы 0.0009225999237969518 секунд.

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_memory_main.py
Максимально занимаемая память: 17.75390625 KB

3) Максимальные значения:

input.txt
main.py
test_memory_main.py
test_time_main.py

input.txt
output.txt
main.py
test_memory_main.py
test_time_main.py

1	50000 50000
2	100000000 100000000
3	99999999 99999999
4	99999999 99999999
5	99999999 99999999
6	99999999 99999999
7	99999999 99999999
8	99999999 99999999
9	99999999 99999999
10	99999999 99999999
11	99999999 99999999
12	99999999 99999999
13	99999999 99999999
14	99999999 99999999
15	99999999 99999999
16	99999999 99999999
17	99999999 99999999
18	99999999 99999999
19	99999999 99999999
20	99999999 99999999
21	99999999 99999999
22	99999999 99999999
23	99999999 99999999
24	99999999 99999999

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_time_main.py
Время работы программы 0.15355899999849498 секунд.

"С:\Program Files\Python311\python.exe" C:\Users\smafi\OneDrive\Desktop\ITMO\algorithms-and-data-structures\lab3\task4\tests\test_memory_main.py
Максимально занимаемая память: 20102.6748046875 KB

	Время выполнения, с	Затраты памяти, КВ
Нижняя граница диапазона значений входных данных из текста задачи	0.00091	17.635
Пример из задачи	0.00092	17.754
Верхняя граница диапазона значений входных данных из текста задачи	0.15356	20102.675

Вывод по задаче: Данный алгоритм является довольно быстрым, но затратным по времени.

Задача №5. Индекс Хирша

Для заданного массива целых чисел `citations`, где каждое из этих чисел - число цитирований i -ой статьи ученого-исследователя, посчитайте индекс Хирша этого ученого.

По определению Индекса Хирша на Википедии: Учёный имеет индекс h , если h из его/её N_p статей цитируются как минимум h раз каждая, в то время как оставшиеся $(N_p - h)$ статей цитируются не более чем h раз каждая. Иными словами, учёный с индексом h опубликовал как минимум h статей, на каждую из которых сослались как минимум h раз.

Если существует несколько возможных значений h , в качестве h -индекса принимается максимальное из них.

- **Формат ввода или входного файла (`input.txt`).** Одна строка `citations`, содержащая n целых чисел, по количеству статей ученого (длина `citations`), разделенных пробелом или запятой.
- **Формат выхода или выходного файла (`output.txt`).** Одно число - индекс Хирша (h -индекс).
- Ограничения: $1 \leq n \leq 5000$, $0 \leq citations[i] \leq 1000$.
- Пример.

input.txt	output.txt
3,0,6,1,5	3

Пояснение. `citations = [3,0,6,1,5]` означает, что ученый опубликовал 5 статей в целом, и каждая из них оказалась процитирована 3, 0, 6, 1, 5 раз соответственно. Поскольку у ученого есть 3 статьи с минимум тремя цитированиями, а у оставшихся двух - не более 3 цитирований, его индекс Хирша равен 3.

- Пример.

input.txt	output.txt
1,3,1	1

- Ограничений по времени (и памяти) не предусмотрено, проверьте максимальный случай при заданных ограничениях на данные, и оцените асимптотическое время.
- Подумайте, если бы массив `citations` был бы изначально отсортирован по возрастанию, можно было бы еще ускорить алгоритм?

Листинг кода

```
from lab3.utils import read_file, check_inp, write_file
from random import randint
```

```

def find_index_hirsh(lst):
    n = len(lst)
    answ = -1
    for i in range(max(lst)):
        cnt_pov = 0
        cnt_new = 0
        for j in range(n):
            if lst[j] == i:
                cnt_pov += 1
            elif lst[j] > i:
                cnt_new += 1

        if cnt_new == i or cnt_new < i and cnt_pov + cnt_new >= i:
            answ = i

    return answ

```

```

def patrition(lst, l, r):
    x = lst[l]
    j = l
    h = l
    for i in range(l+1, r+1):
        if lst[i] < x:
            h += 1
            j += 1
            if h != j:
                lst[h], lst[i] = lst[i], lst[h]
                lst[h], lst[j] = lst[j], lst[h]
        else:
            lst[i], lst[j] = lst[j], lst[i]
    elif lst[i] == x:
        h += 1
        lst[i], lst[h] = lst[h], lst[i]

    lst[l], lst[j] = lst[j], lst[l]
    return (j, h)

```

```

def randomized_quicksort(lst, l, r):
    if l < r:
        k = randint(l, r)
        lst[l], lst[k] = lst[k], lst[l]
        (m1, m2) = patrition(lst, l, r)
        randomized_quicksort(lst, l, m1-1)
        randomized_quicksort(lst, m2+1, r)

```

```

def main():
    read_inp = read_file("../txtf/input.txt", 5)
    lst = read_inp

    max_n = 10**5
    max_el = 10**9
    check_inp(max_n, max_el, len(lst), lst, [], 5)

    randomized_quicksort(lst, 0, len(lst)-1)
    ind = str(find_index_hirsh(lst))

    write_file("../txtf/output.txt", str(ind))

```

```

main()

```

Тесты:

1) Тест на время работы:

```
from lab3.task5.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))
```

2) Тест на занимаемую память:

```
from lab3.task5.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализовано 4 функции:

1) main():

Открывает файл input.txt и считывает из него список. Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Применяет к списку сортировку randomized_sort() и с помощью функции find_index_hirsh() получает индекс, который далее записывает в файл output.txt.

2) randomized_sort():

Описано в задании №1

3) patrition():

Описано в задании №1

4) find_index_hirsh():

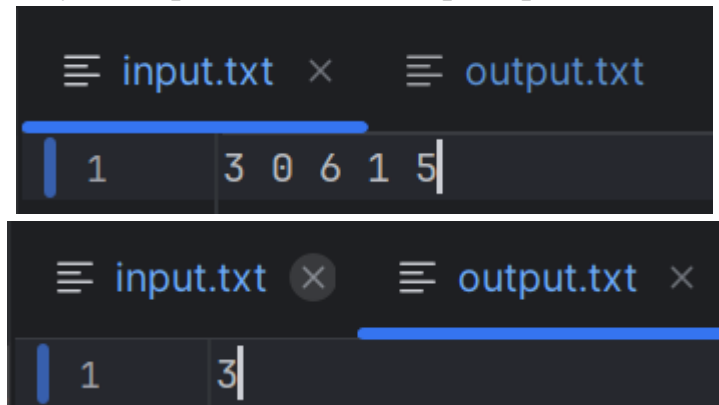
Запускает цикл от 0 до максимального элемента списка. Далее проверяет, может ли i быть индексом хирша: проходит по списку и считает число статей, процитированных больше, чем i раз, и равное i раз. Затем сверяет с условием задачи и записывает в ответ, если подходит.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.

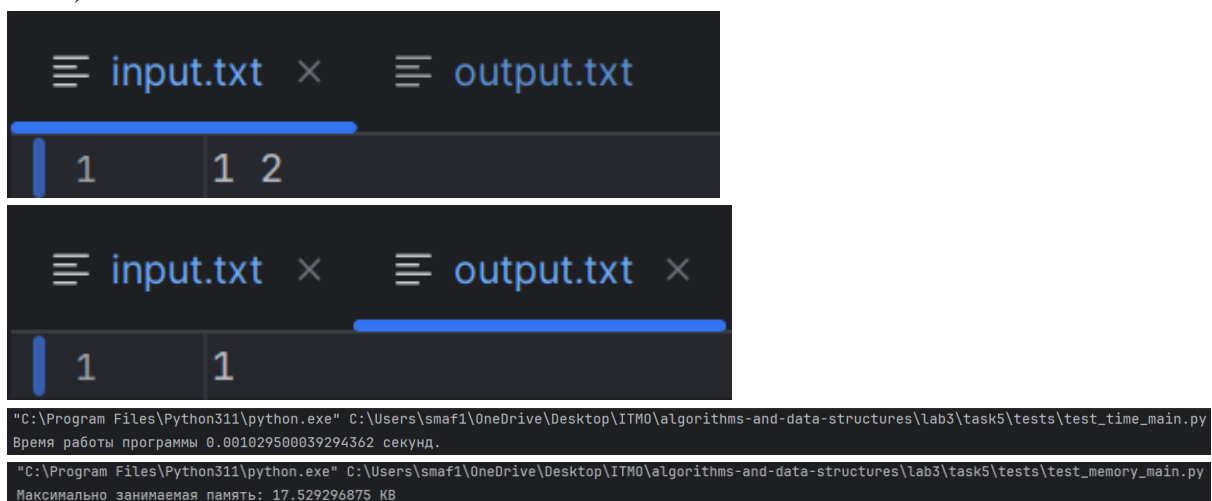
- 2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

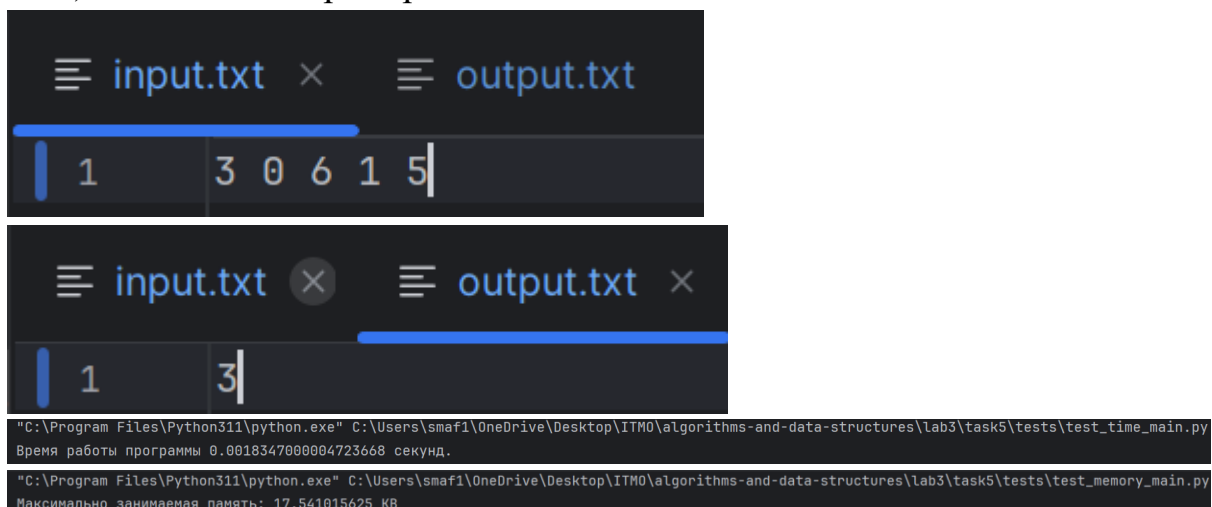


Результат работы кода на максимальных и минимальных значениях:

- 1) Минимальные значения:



- 2) Значения из примера:



- 3) Максимальные значения:

The screenshot shows a Python IDE with a toolbar containing tabs for `input.txt`, `output.txt`, `main.py`, `test_memory_main.py`, and `test_time_main.py`. The main window displays a list of numbers: 1, 1000, 1000, 1000, 1000, 1000, 1000, 999, 999, 999, 999, 999, 998, 998, 998, 998, 998, 997, 997, 997, 997, 997, 996, 996, 996, 996. A zoomed-in view of the first two elements shows the value 1 and 834.

	Время выполнения, с	Затраты памяти, КВ
Минимальные значения	0.00103	17.529
Пример из задачи	0.00183	17.541
Максимальные значения	0.42715	443.910

Вывод по задаче: Был реализован алгоритм поиска индекса Хирша с использованием randomized quicksort.

Вывод

Была изучена быстрая сортировка и закрепились знания программирования на языке Python на практике.