

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.

Выполнил: Криличевский М. Е.

Номер группы: К3139

Проверил: Афанасьев А. В.

Санкт-Петербург

2024 г.

## Содержание

<i>Задание 1</i> .....	<b>3</b>
<i>Задание 2</i> .....	<b>8</b>
<i>Задание 3</i> .....	<b>11</b>
<i>Задание 6</i> .....	<b>14</b>
<i>Задание 8</i> .....	<b>17</b>
<i>Задание 9</i> .....	<b>20</b>
<i>Вывод</i> .....	<b>23</b>

# Задание 1

## 1 задача. 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 случайных 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 размера  $10^3, 10^4, 10^5$  чисел порядка  $10^9$ , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний - случайный. Сравните на данных наборах 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 наборах 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	

## Решение:

```
1. import random
2. import heapq
3. from lab_3.utils import input_operation,
   output_operation, first_check, string
4.
5.
6. def three_way_partition(arr, low, high):
7.     pivot = arr[low]
8.     i = low + 1
9.     j = low + 1
10.    k = high
11.
12.    while j <= k:
13.        if arr[j] < pivot:
14.            arr[i], arr[j] = arr[j], arr[i]
15.            i += 1
16.            j += 1
17.        elif arr[j] > pivot:
18.            arr[j], arr[k] = arr[k], arr[j]
19.            k -= 1
20.        else:
21.            j += 1
22.    arr[low], arr[i-1] = arr[i-1], arr[low]
23.    return i - 1, k
24.
25.    def partition(arr, low, high):
26.        pivot_index = random.randint(low, high)
27.        pivot = arr[pivot_index]
28.        arr[pivot_index], arr[high] = arr[high],
arr[pivot_index]
29.        i = low - 1
30.        for j in range(low, high):
31.            if arr[j] <= pivot:
32.                i += 1
33.                arr[i], arr[j] = arr[j], arr[i]
34.        arr[i + 1], arr[high] = arr[high], arr[i + 1]
35.        return i + 1
36.
37.    def quicksort_threeway(arr, low=0, high=None):
38.        if high is None:
39.            high = len(arr) - 1
40.        if low < high:
41.            mid1, mid2 = three_way_partition(arr, low,
high)
42.            quicksort_threeway(arr, low, mid1 - 1)
43.            quicksort_threeway(arr, mid2 + 1, high)
44.
45.    def quicksort_standard(arr, low=0, high=None,
max_depth=None):
46.        if high is None:
47.            high = len(arr) - 1
48.        if max_depth is None:
49.            max_depth = 2 * len(arr).bit_length()
50.
51.        if low < high:
```

```

52.         if max_depth <= 0:
53.             heapq.heapify(arr[low:high+1])
54.             for i in range(high, low - 1, -1):
55.                 arr[i], arr[low] = arr[low], arr[i]
56.                 heapq.heapify(arr[low:i])
57.         else:
58.             pivot_index = partition(arr, low, high)
59.             quicksort_standard(arr, low, pivot_index
- 1, max_depth - 1)
60.             quicksort_standard(arr, pivot_index + 1,
high, max_depth - 1)
61.
62.     if __name__ == "__main__":
63.         input_f = "../txtf/input"
64.         output_f = "../txtf/output"
65.         input_f2 = "../txtf/hard_input"
66.         output_f2 = "../txtf/hard_output"
67.         FIRST_CHECK_1 = first_check("../txtf/input")
68.         FIRST_CHECK_2 =
first_check("../txtf/hard_input")
69.         if FIRST_CHECK_1:
70.             result = input_operation(input_f)
71.             quicksort_threeway(result)
72.             output_operation(output_f, string(result))
73.             print(f"Входные данные в файле: {input_f}
корректны")
74.         else:
75.             output_operation(output_f, "Ошибка входных
данных")
76.             print(f"Ошибка входных данных в файле:
{input_f}")
77.
78.         if FIRST_CHECK_2:
79.             result2 = input_operation(input_f2)
80.             quicksort_threeway(result2)
81.             output_operation(output_f2, string(result2))
82.             print(f"Входные данные в файле: {input_f2}
корректны")
83.         else:
84.             output_operation(output_f2, "Ошибка входных
данных")
85.             print(f"Ошибка входных данных в файле:
{input_f2}")

```

## Объяснение решения:

Этот код реализует два варианта алгоритма быстрой сортировки: стандартную и трёхпутевую сортировку для обработки элементов меньше, равных и больше опорного. Программа работает с входными данными из файлов, проверяет их с помощью функций `first_check()`, сортирует массив и записывает отсортированные данные в выходной файл.

Входной файл:

test_time_memory.py		hard_input	hard_output
1	100000		
2	83747 62546 46274 18818 32971 83767 95147 36105 7		
	↵ 13331 34395 2463 68855 93708 41210 94454 26731 1		
	↵ 40368 63609 68603 62447 19458 67869 35564 44953		
	↵ 45457 56356 37159 29599 26361 9530 41050 83765 9		
	↵ 59149 51796 49617 30120 49166 24006 47920 84064		
	↵ 71774 94429 75342 6843 20930 58757 1536 46893 7		
	↵ 39408 2741 30177 42292 92845 31777 70127 73119		
	↵ 16120 8695 19300 62393 67162 98039 69523 22754 9		
	↵ 27196 67153 15024 33754 20785 26898 35415 70058		
	↵ 87474 83022 89888 29912 36486 87440 56202 80764		
	↵ 64542 35526 55551 61424 48938 24147 59627 78825		

Выходной файл:

test_time_memory.py		hard_input	
1	1 1 3 4 4 4 7 7 9 9 9 10 11 14 16 17		
	↵ 65 67 67 68 69 70 72 73 74 77 78 79		
	↵ 115 117 119 119 122 122 123 125 125		
	↵ 150 151 153 153 154 155 156 157 158		
	↵ 186 188 188 190 191 192 195 197 198		
	↵ 228 229 229 231 231 232 232 233 237		
	↵ 264 265 265 268 270 270 271 271 271		
	↵ 308 310 310 312 313 314 314 317 318		

## Тестирование времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/PycharmI
```

```
ВРЕМЯ №1: 0.013111374995787628
```

```
Память №1: 21.5 МБ
```

```
Размер списка №1: 120 байт
```

```
ВРЕМЯ №2: 0.1548086249968037
```

```
Память №2: 24.3 МБ
```

```
Размер списка №2: 782 Мбайт
```

```
Process finished with exit code 0
```

## Задание 2

### 2 задача. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - QuickSort. Далее приведена программа на языке Pascal Python, которая сортирует массив *a*, используя этот алгоритм.

```
def qsort (left, right):
    key = a [(left + right) // 2]
    i = left
    j = right
    while i <= j:
        while a[i] < key: # first while
            i += 1
        while a[j] > key : # second while
            j -= 1
        if i <= j :
            a[i], a[j] = a[j], a[i]
            i += 1
            j -= 1
    if left < j:
        qsort(left, j)
    if i < right:
        qsort(i, right)
```

```
qsort(0, n - 1)
```

Хотя QuickSort является очень быстрой сортировкой в среднем, существуют тесты, на которых она работает очень долго. Оценивать время работы алгоритма будем числом сравнений с элементами массива (то есть, суммарным числом сравнений в первом и втором while). Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

[Задача на астр.](#)

- **Формат входного файла (input.txt).** В первой строке находится единственное число  $n$  ( $1 \leq n \leq 10^6$ ).
- **Формат выходного файла (output.txt).** Вывести перестановку чисел от 1 до  $n$ , на которой быстрая сортировка выполнит максимальное число сравнений. Если таких перестановок несколько, вывести любую из них.
- Ограничение по времени. 2 сек.
- Ограничение по памяти. 256 мб.
- Пример:

input.txt	output.txt
3	1 3 2



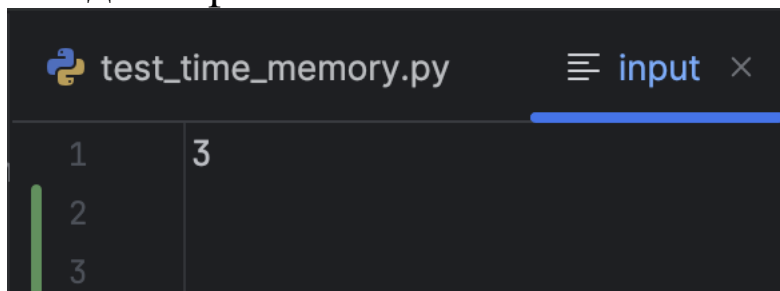
## Решение:

```
1. from lab_3.utils import open_f, string, output_operation,  
   second_check  
2.  
3. import random  
4.  
5. def generate_worst_case(n):  
6.     permutation = list(range(1, n + 1))  
7.     random.shuffle(permutation)  
8.     for i in range(n // 2):  
9.         if random.random() < 0.5:  
10.            permutation[i], permutation[n - 1 - i] =  
permutation[n - 1 - i], permutation[i]  
11.  
12.     return permutation  
13.  
14. if __name__ == "__main__":  
15.     FILE_INPUT = "../txtf/input"  
16.     FILE_OUTPUT = "../txtf/output"  
17.     t = open_f(FILE_INPUT)  
18.     if second_check(t):  
19.         worst_case_permutation = generate_worst_case(t)  
20.         res = string(worst_case_permutation)  
21.         output_operation(FILE_OUTPUT, res)  
22.         print("Входное значение корректно")  
23.     else:  
24.         print("Входное значение некорректно")
```

## Объяснение решения:

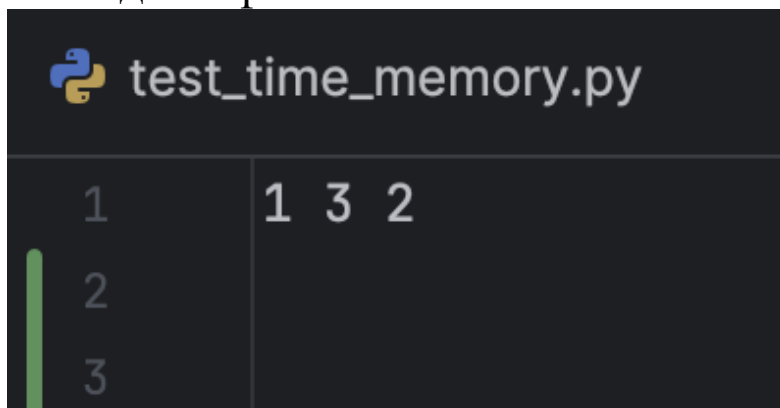
Этот код генерирует "худший случай" для какого-либо алгоритма, который сортирует массив перестановок (например, Quick Sort). Худший случай — это такая структура данных, при которой алгоритм работает наихудшим образом (максимум времени). Программа считывает значение из входного файла создаёт перестановку чисел от 1 до n, перемешивает их случайным образом, а затем выполняет случайные обмены элементов для усложнения упорядоченности. Полученный массив записывается в выходной файл, если входное значение корректно.

Входной файл:



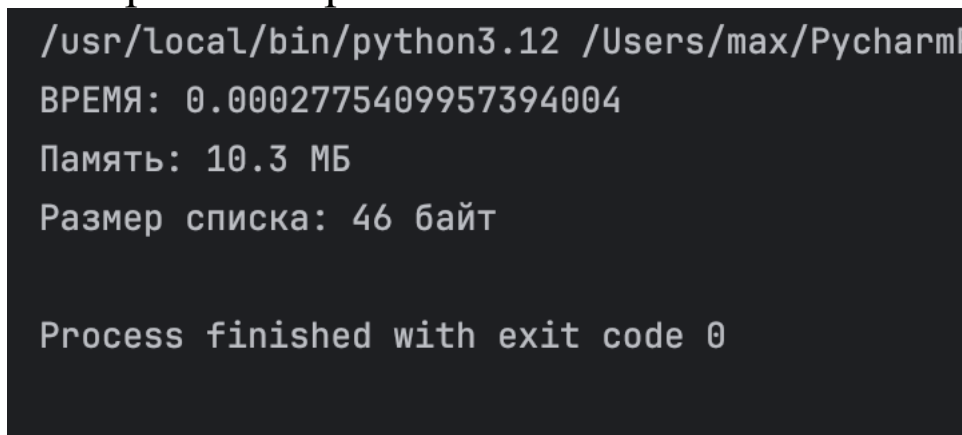
```
test_time_memory.py input ×  
1 3  
2  
3
```

Выходной файл:



```
test_time_memory.py  
1 1 3 2  
2  
3
```

Тестирование времени и памяти:



```
/usr/local/bin/python3.12 /Users/max/Pycharm  
ВРЕМЯ: 0.0002775409957394004  
Память: 10.3 МБ  
Размер списка: 46 байт  
  
Process finished with exit code 0
```

## Задание 3

### 3 задача. 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

## Решение:

```
1. from lab_3.utils import input_operation_two,
   input_operation_three, third_check, output_operation
2.
3. def matreshka(arr, k, n):
4.     groups = [[] for _ in range(k)]
5.     for i in range(n):
6.         groups[i % k].append(arr[i])
7.
8.     for group in groups:
9.         group.sort()
10.
11.         sorted_arr = []
12.         for i in range(n):
13.             sorted_arr.append(groups[i % k][i // k])
14.
15.         if sorted_arr == sorted(arr):
16.             return True
17.         else:
18.             return False
19.
20. if __name__=="__main__":
21.     FILE_INPUT = "../txtf/input"
22.     FILE_OUTPUT = "../txtf/output"
23.     l1 = input_operation_two(FILE_INPUT)
24.     l2 = input_operation_three(FILE_INPUT)
25.     n,k = int(l1[0]), int(l1[1])
26.     THIRD_CHECK = third_check(FILE_INPUT)
27.     if THIRD_CHECK:
28.         result = matreshka(l2, k, n)
29.         if result:
30.             file_o = output_operation(FILE_OUTPUT, "ДА")
31.         if not(result):
32.             file_o = output_operation(FILE_OUTPUT, "НЕТ")
33.         print("Входные данные корректны")
34.     else:
35.         output_operation(FILE_OUTPUT, "Входные данные
   некорректны")
36.         print("Входные данные некорректны")
```

## Объяснение решения:

Этот код проверяет, возможно ли разделить массив на групп так, чтобы после сортировки каждой группы и объединения элементов в исходной последовательности массив оказался в отсортированном порядке. Программа читает из файла размеры массива и количество групп, а затем сам массив. Если проверка успешна, в выходной файл записывается `ДА`, а если нет — `НЕТ`.

Входной файл:

test_time_memory.py	
1	5 3
2	1 5 3 4 1
3	

Выходной файл:

test_time_memory.py	
1	ДА
2	
3	

Тестирование времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/Pycharm
ВРЕМЯ: 0.0003642090014182031
Память: 9.7 МБ

Process finished with exit code 0
```

## Задание 6

### 6 задача. Сортировка целых чисел

В этой задаче нужно будет отсортировать много неотрицательных целых чисел.

Вам даны два массива,  $A$  и  $B$ , содержащие соответственно  $n$  и  $m$  элементов. Числа, которые нужно будет отсортировать, имеют вид  $A_i \cdot B_j$ , где  $1 \leq i \leq n$  и  $1 \leq j \leq m$ . Иными словами, каждый элемент первого массива нужно умножить на каждый элемент второго массива.

Пусть из этих чисел получится отсортированная последовательность  $C$  длиной  $n \cdot m$ . Выведите сумму каждого десятого элемента этой последовательности (то есть,  $C_1 + C_{11} + C_{21} + \dots$ ).

- **Формат входного файла (input.txt).** В первой строке содержатся числа  $n$  и  $m$  ( $1 \leq n, m \leq 6000$ ) – размеры массивов. Во второй строке содержится

6

$n$  чисел – элементы массива  $A$ . Аналогично, в третьей строке содержится  $m$  чисел — элементы массива  $B$ . Элементы массива неотрицательны и не превосходят 40000.

- **Формат выходного файла (output.txt).** Выведите одно число — сумму каждого десятого элемента последовательности, полученной сортировкой попарных произведений элементов массивов  $A$  и  $B$ .
- Ограничение по времени. 2 сек.
- **Ограничение по времени распространяется на сортировку, без учета времени на перемножение. Подумайте, какая сортировка будет эффективнее, сравните на практике.**
- Однако бытует мнение [на OpenEdu, неделя 3, задача 2](#), что эту задачу можно решить на Python и уложиться в 2 секунды, включая в общее время перемножение двух массивов.
- Ограничение по памяти. 512 мб.
- Пример:

input.txt	output.txt
4 4	51
7 1 4 9	
2 7 8 11	

## Решение:

```
1. from lab_3.utils import input_operation_z,  
   input_operation_z_two, input_operation_z_three,  
   output_operation, sixth_check  
2.  
3. def sort_z(a, b):  
4.     products = []  
5.     for x in a:  
6.         for y in b:  
7.             products.append(x * y)  
8.  
9.     products.sort()  
10.    total_sum = sum(products[i] for i in range(0,  
len(products), 10))  
11.    return total_sum  
12.  
13. if __name__=="__main__":  
14.     FILE_INPUT = "../txtf/input"  
15.     FILE_OUTPUT = "../txtf/output"  
16.     list = input_operation_z(FILE_INPUT)  
17.     n, m = int(list[0]), int(list[1])  
18.     a = input_operation_z_two(FILE_INPUT)  
19.     b = input_operation_z_three(FILE_INPUT)  
20.     if sixth_check(n,m,a,b):  
21.         output_operation(FILE_OUTPUT, sort_z(a,b))  
22.         print("Входные данные корректны")  
23.     if not(sixth_check(n,m,a,b)):  
24.         output_operation(FILE_OUTPUT, "Ошибка  
входных данных")  
25.         print("Ошибка входных данных")
```

## Объяснение решения:

Этот код выполняет расчёт на основе произведений элементов двух массивов. Конкретно, он вычисляет все попарные произведения элементов массивов `a` и `b`, сортирует их, а затем находит сумму каждого десятого элемента в отсортированном списке произведений. Если входные данные из файла корректны, результат записывается в выходной файл, а если нет — выводится сообщение об ошибке.

Входной файл:

test_time_memory.py	
1	4 4
2	7 1 4 9
3	2 7 8 11

Выходной файл:

test_time_memory.py	
1	51
2	
3	

Тестирование времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/PycharmI  
ВРЕМЯ: 0.0003983750066254288  
Память: 9.8 МБ  
Размер списка: 28 байт  
  
Process finished with exit code 0
```



## Задание 8

### 8 задача. $K$ 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  $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ .

8

- Формат ввода или входного файла (input.txt).** Первая строка содержит  $n$  - общее количество точек на плоскости и через пробел  $K$  - количество ближайших 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 строчку 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]

## Решение:

```
1. from lab_3.utils import read_input, write_output,
   eighth_check, output_operation
2. import math
3.
4. def closest_points(points, k):
5.     distances = []
6.
7.     for point in points:
8.         if isinstance(point, tuple) and len(point) == 2:
9.             x, y = point
10.            distance = math.sqrt(x ** 2 + y ** 2)
11.            distances.append((distance, point))
12.        else:
13.            raise ValueError("Каждая точка должна
быть кортежем с двумя координатами.")
14.
15.        distances.sort(key=lambda x: x[0])
16.        return [point for _, point in distances[:k]]
17.
18. if __name__ == "__main__":
19.     FILE_INPUT = "../txtf/input"
20.     FILE_OUTPUT = "../txtf/output"
21.     points, k = read_input(FILE_INPUT)
22.     point_check = eighth_check(FILE_INPUT)
23.     if point_check:
24.         result = closest_points(points, k)
25.         write_output(FILE_OUTPUT, result)
26.         print("Входные данные корректны")
27.     else:
28.         output_operation(FILE_OUTPUT, "Ошибка
входных данных")
29.         print("Ошибка входных данных")
```

## Объяснение решения:

Этот код находит ближайших точек к началу координат (0, 0) из заданного списка точек на плоскости. Расстояние до начала координат рассчитывается для каждой точки с использованием формулы Эвклидовой метрики. Затем точки сортируются по возрастанию их расстояния, и выбираются ближайших точек. Если входные данные корректны, результат записывается в выходной файл. Если данные некорректны, программа выводит ошибку.

Входной файл:

test_time_memory.py	
1	3 2
2	3 3
3	5 -1
4	-2 4
5	

Выходной файл:

test_time_memory.py	
1	[[3, 3], [-2, 4]]
	💡

Тестирование времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/PycharmI  
ВРЕМЯ: 0.0002514160005375743  
Память: 10.2 МБ  
  
Process finished with exit code 0
```

## Задание 9

### 9 задача. Ближайшие точки

В этой задаче, ваша цель - найти пару ближайших точек среди данных  $n$  точек (между собой). Это базовая задача вычислительной геометрии, которая находит применение в компьютерном зрении, систем управления трафиком.

- Цель. Заданы  $n$  точек на поверхности, найти наименьшее расстояние между двумя (разными) точками. 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  $n$  - количество точек. 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 минимальное расстояние. Абсолютная погрешность между вашим ответом и оптимальным решением должна быть не более  $10^{-3}$ . Чтобы это обеспечить, выведите ответ с 4 знаками после запятой.

9

- Ограничение по времени. 10 сек.
- Ограничение по памяти. 256 мб.
- Пример 1.

input.txt	output.txt
2	5.0
0 0	
3 4	

Здесь всего 2 точки, расстояние между ними равно 5.

- Пример 2.

input.txt	output.txt
4	0.0
7 7	
1 100	
4 8	
7 7	

Здесь есть две точки, координаты которых совпадают, соответственно, расстояние между ними равно 0.

## Решение:

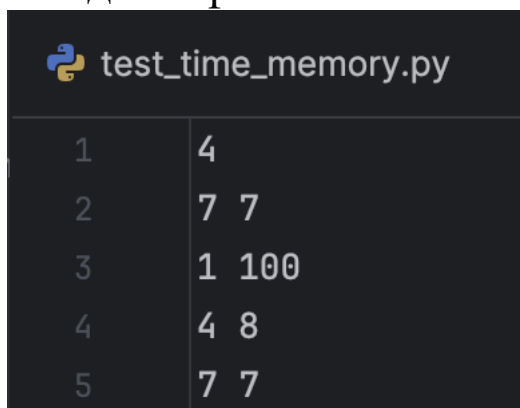
```
1. from lab_3.utils import nine_check, output_operation, read_input_n
2. import math
3.
4. def distance(p1, p2):
5.     return math.sqrt((p1[0] - p2[0]) ** 2 + (p1[1] - p2[1]) ** 2)
6.
7. def strip_closest(strip, d_min):
8.     min_d = d_min
9.     strip.sort(key=lambda point: point[1])
10.    for i in range(len(strip)):
11.        for j in range(i + 1, len(strip)):
12.            if strip[j][1] - strip[i][1] >= min_d:
13.                break
14.            min_d = min(min_d, distance(strip[i], strip[j]))
15.    return min_d
16.
17.    def closest_pair_recursive(points):
18.        if len(points) <= 3:
19.            min_d = float('inf')
20.            for i in range(len(points)):
21.                for j in range(i + 1, len(points)):
22.                    min_d = min(min_d, distance(points[i],
points[j]))
23.        return min_d
24.
25.        mid = len(points) // 2
26.        mid_x = points[mid][0]
27.
28.        left_half = points[:mid]
29.        right_half = points[mid:]
30.
31.        d_left = closest_pair_recursive(left_half)
32.        d_right = closest_pair_recursive(right_half)
33.
34.        d_min = min(d_left, d_right)
35.
36.        strip = [point for point in points if abs(point[0] - mid_x)
< d_min]
37.
38.        return min(d_min, strip_closest(strip, d_min))
39.
40.    def closest_pair(points):
41.        points.sort()
42.        return closest_pair_recursive(points)
43.
44.
45.    if __name__ == "__main__":
46.        FILE_INPUT = "../txtf/input"
47.        FILE_OUTPUT = "../txtf/output"
48.        points = read_input_n(FILE_INPUT)
49.        point_check = nine_check(FILE_INPUT)
50.        if point_check:
51.            result = closest_pair(points)
52.            output_operation(FILE_OUTPUT, result)
53.            print("Входные данные корректны")
54.        else:
55.            output_operation(FILE_OUTPUT, "Ошибка входных данных")
56.            print("Ошибка входных данных")
```

## Объяснение решения:

Этот код находит пару точек в двумерной плоскости, которые находятся на минимальном расстоянии друг от друга. Для этого используется эффективный алгоритм "разделяй и властвуй" с временной сложностью

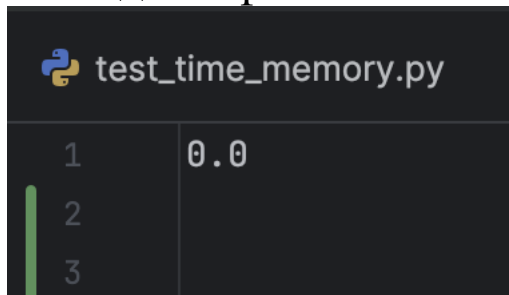
. Он рекурсивно делит точки на две части, вычисляет минимальное расстояние для каждой половины, а затем проверяет точки на стыке между левым и правым регионами для минимального расстояния. Если входные данные корректны, результат записывается в выходной файл, в противном случае выводится сообщение об ошибке.

## Входной файл:



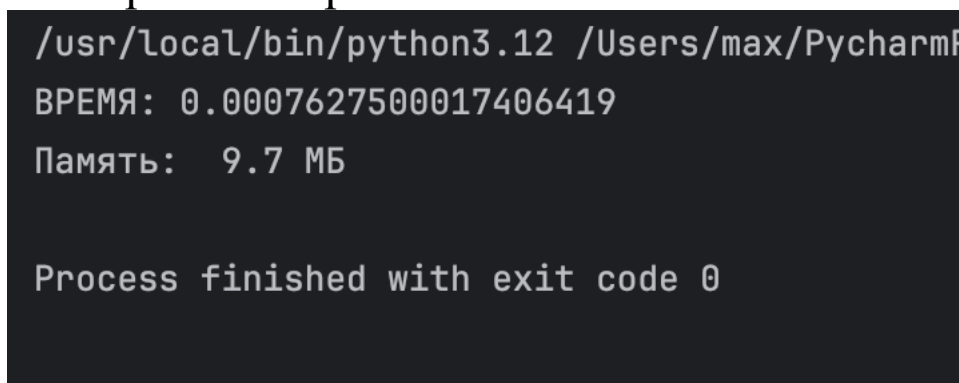
1	4
2	7 7
3	1 100
4	4 8
5	7 7

## Выходной файл:



1	0.0
2	
3	

## Тестирование времени и памяти:



```
/usr/local/bin/python3.12 /Users/max/PycharmF
ВРЕМЯ: 0.0007627500017406419
Память: 9.7 МБ

Process finished with exit code 0
```

## Вывод

В данной лабораторной работе закрепил работу с алгоритмами быстрой сортировки и сортировки за линейное время, а также местами сравнить время их работы.