

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 14

Выполнила:  
Рудникова Виктория Олеговна  
К3125

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2022 г.

## Содержание отчета

<b>Содержание отчета</b>	<b>2</b>
<b>Задачи по варианту</b>	<b>3</b>
Задача №1. Улучшение Quick sort	3
Задача №2. Анти-quick sort	7
Задача №3. Сортировка пугалом	10
<b>Дополнительные задачи</b>	<b>13</b>
Задача №4. Точки и отрезки	13
Задача №8. К ближайших точек к началу координат	18
<b>Вывод:</b>	<b>22</b>

## Задачи по варианту

### Задача №1. Улучшение Quick sort

1. Используя псевдокод процедуры Randomized-QuickSort, а также Partition, 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.

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. То есть ваша новая процедура разделения должна разбить массив на три части:

- $A[k] < x$  для всех  $l + 1 \leq k \leq m_1 - 1$
- $A[k] = x$  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.

```
from random import randint
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def partition_3(arr, first, last):
    cur = arr[first]
    left = []
    mid = []
    right = []
    j = 0
```

```

count = 0
for i in range(first, last + 1):
    if arr[i] < cur:
        left.append(arr[i])
        j += 1
    elif arr[i] == cur:
        mid.append(arr[i])
        count += 1
    else:
        right.append(arr[i])
arr[first: last + 1] = left + mid + right
return first + j, first + j + count - 1

```

```

def randomized_quick_sort(arr, first, last):
    if first < last:
        k = randint(first, last)
        arr[first], arr[k] = arr[k], arr[first]
        m1, m2 = partition_3(arr, first, last)
        randomized_quick_sort(arr, first, m1 - 1)
        randomized_quick_sort(arr, m2 + 1, last)

```

```

if __name__ == "__main__":
    start()
    with open('input.txt', 'w') as f:
        n = randint(0, 10 ** 3)
        f.write(str(n) + '\n')
        array = [randint(-10 ** 6, 10 ** 6) for i in range(n)]
        f.write(' '.join(str(i) for i in array))
    randomized_quick_sort(array, 0, n - 1)
    with open('output.txt', 'w+') as g:
        g.write(' '.join(str(i) for i in array))
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

```

Реализуется метод `partition_3`, который оптимизирует quicksort для массивов с большим количеством одинаковых элементов, разделяя список не на две, а на три части. Этот метод используется в алгоритме быстрой сортировки.

Результат работы кода на примерах из текста задачи:

```

Задачи по варианту\input.txt
1 5
2 2 3 9 2 2

Задачи по варианту\output.txt
1 2 2 2 3 9
  
```

Результат работы кода на максимальных и минимальных значениях:

```

Задачи по варианту\input.txt
1 100000
2 99999 99998 99997 99996 99995 99994 99993

Задачи по варианту\output.txt
1 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1
  
```

```

Задачи по варианту\input.txt
1 3
2 1 3 2

Задачи по варианту\output.txt
1 1 2 3
  
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.244140625 KB
Пример из задачи	0.109375 sec	17.244140625 KB
Верхняя граница	0.109375 sec	17.244140625 KB

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче: в этой задаче я улучшила алгоритм быстрой сортировки путём деления списка на три части вместо двух.

## Задача №2. Анти-quick sort

Для сортировки последовательности чисел широко используется быстрая сортировка - QuickSort. 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. Требуется написать программу, генерирующую тест, на котором быстрая сортировка сделает наибольшее число таких сравнений.

- 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.

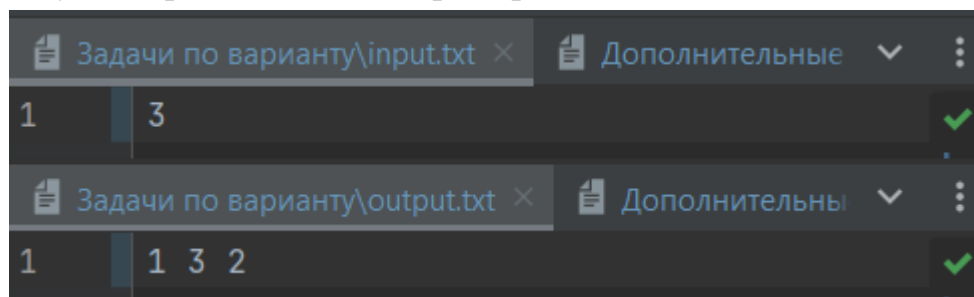
```
from time import process_time
from tracemalloc import start, get_traced_memory

def generate_array(num):
    arr = [i + 1 for i in range(num)]
    for i in range(2, num):
        arr[i], arr[i // 2] = arr[i // 2], arr[i]
    return arr

if name == 'main':
    start()
    with open('input.txt', 'r') as f:
        n = int(f.readline())
    with open('output.txt', 'w+') as g:
        g.write(' '.join([str(i) for i in generate_array(n)]))
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')
```

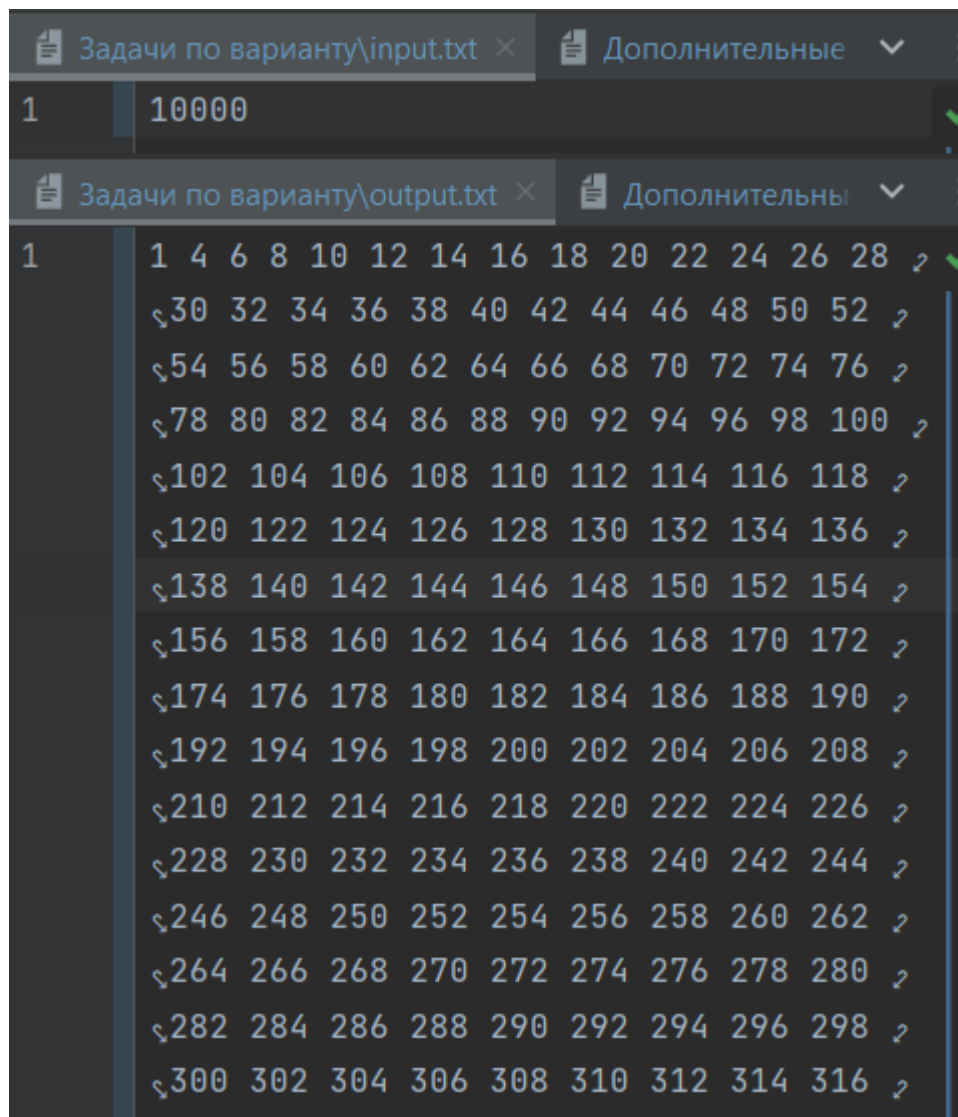
В задаче реализуется алгоритм генерации такого списка, на котором быстрая сортировка сделает наибольшее количество сравнений, а именно: каждый элемент меняется местами с элементом, индекс которого в два раза меньше.

Результат работы кода на примерах из текста задачи:



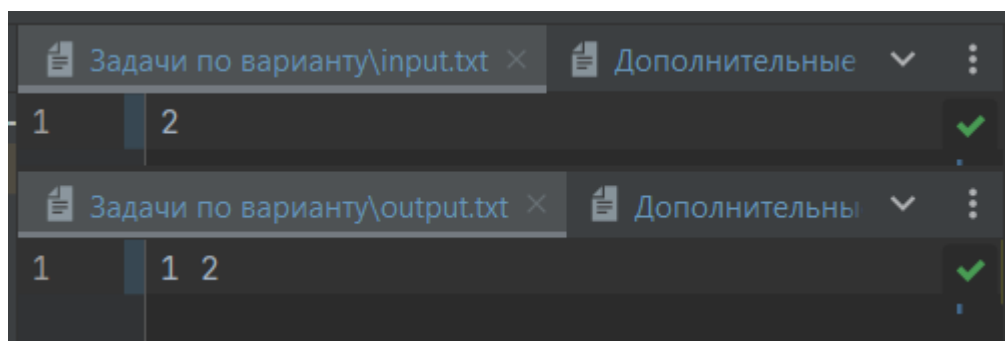
```
Задачи по варианту\input.txt x  Дополнительные v ...
1 3 ✓
Задачи по варианту\output.txt x  Дополни... v ...
1 1 3 2 ✓
```

Результат работы кода на максимальных и минимальных значениях:



```
Задачи по варианту\input.txt x  Дополнительные v ...
1 10000 ✓
Задачи по варианту\output.txt x  Дополни... v ...
1 1 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50 52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100 102 104 106 108 110 112 114 116 118 120 122 124 126 128 130 132 134 136 138 140 142 144 146 148 150 152 154 156 158 160 162 164 166 168 170 172 174 176 178 180 182 184 186 188 190 192 194 196 198 200 202 204 206 208 210 212 214 216 218 220 222 224 226 228 230 232 234 236 238 240 242 244 246 248 250 252 254 256 258 260 262 264 266 268 270 272 274 276 278 280 282 284 286 288 290 292 294 296 298 300 302 304 306 308 310 312 314 316 ✓
```





	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.1875 sec	17.5947265625 KB
Пример из задачи	0.125 sec	17.7724609375 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.15625 sec	156.7783203125 KB

Вывод по задаче: в этой задаче я подумала, когда быстрая сортировка сделает наибольшее количество сравнений.

### 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 строке 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.

```
from copy import copy
from task1 import randomized_quick_sort
from time import process time
from tracemalloc import start, get_traced_memory
```

```
def scarecrow(arr, num, key):
    sorted_array = copy(arr)
    for i in range(key):
        arr1 = []
        idx = i
        size = 0
        while idx <= num - 1:
            arr1.append(sorted_array[idx])
            idx += key
            size += 1
        randomized_quick_sort(arr1, 0, size - 1)
        for j in range(size):
            sorted_array[i + j * key] = arr1[j]
        randomized_quick_sort(arr, 0, num - 1)
    if sorted_array == arr:
        return True
    return False
```

```
if __name__ == "__main__":
```

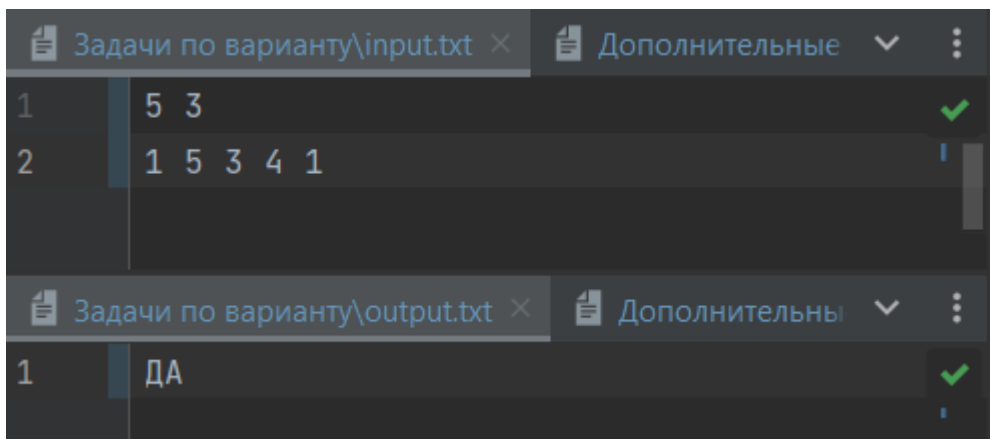
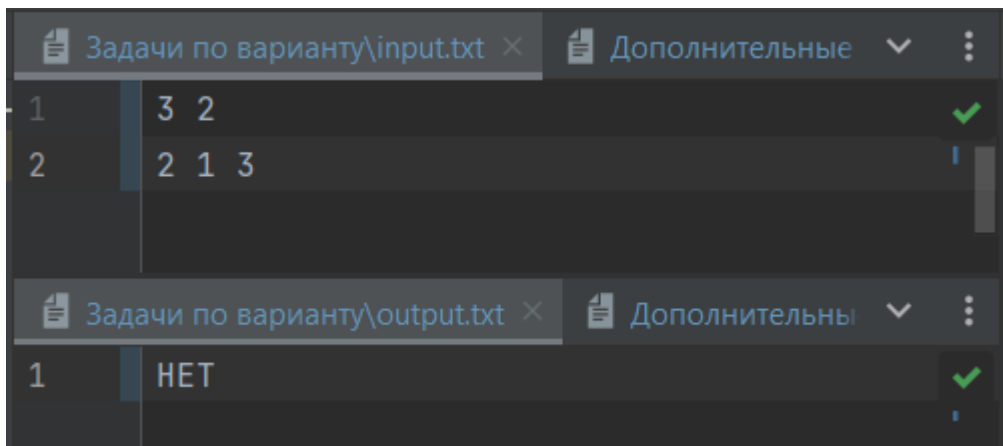
```

start()
with open('input.txt', 'r') as f:
    n, k = map(int, f.readline().split())
    array = [int(i) for i in f.readline().split()]
with open('output.txt', 'w+') as g:
    if scarecrow(array, n, k):
        g.write('ДА')
    else:
        g.write('НЕТ')
print('Time:', str(process_time()), 'sec')
print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

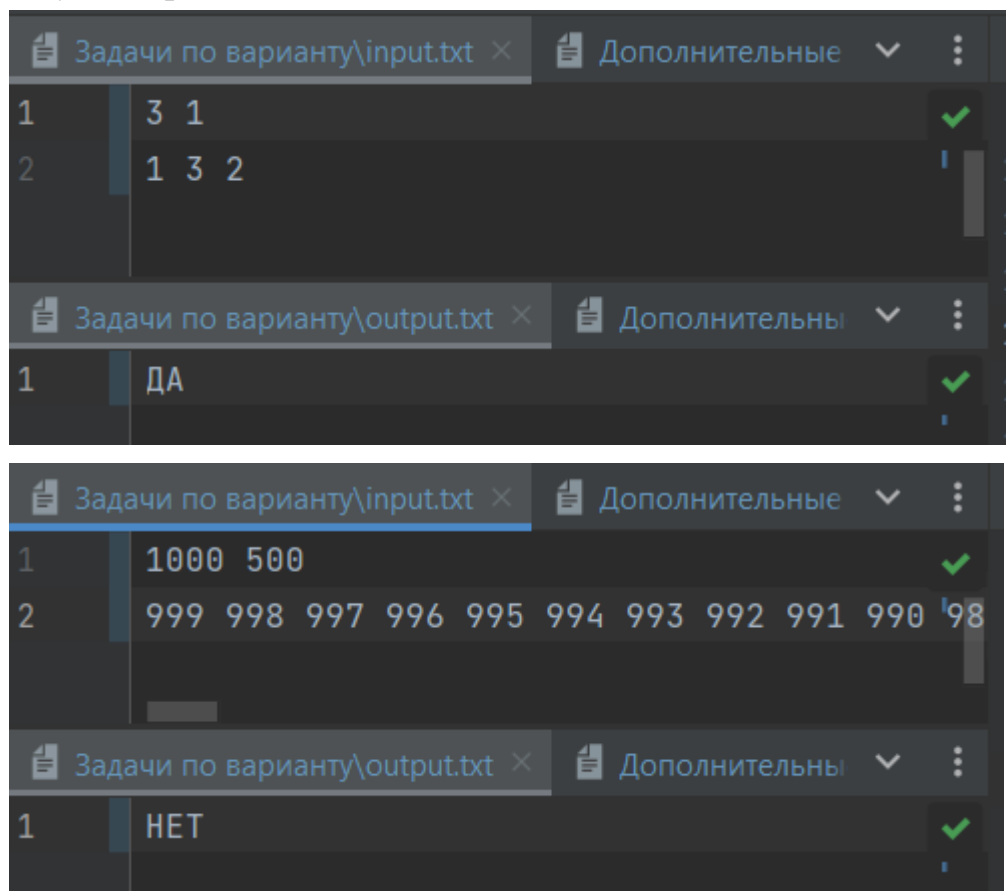
```

В методе `scarecrow` реализуется алгоритм сортировки пугалом, и его результат сравнивается со списком, отсортированным быстрой сортировкой.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.203125 sec	17.5087890625 KB
Пример из задачи	0.234375 sec	17.5166015625 KB
Пример из задачи	0.21875 sec	17.5126953125 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.265625 sec	59.958984375 KB

Вывод по задаче: в этой задаче я реализовала алгоритм сортировки пугалом.

## Дополнительные задачи

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

```
from random import randint
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def partition_3(arr, first, last):
    cur = arr[first][0]
    left = [[], []]
```

```

mid = [[], []]
right = [[], []]
j = 0
cnt = 0
for i in range(first, last + 1):
    if arr[i][0] < cur:
        left[0].append(arr[i][0])
        left[1].append(arr[i][1])
        j += 1
    elif arr[i][0] == cur:
        mid[0].append(arr[i][0])
        mid[1].append(arr[i][1])
        cnt += 1
    else:
        right[0].append(arr[i][0])
        right[1].append(arr[i][1])
total0 = left[0] + mid[0] + right[0]
total1 = left[1] + mid[1] + right[1]
for i in range(len(total0)):
    arr[first + i][0] = total0[i]
    arr[first + i][1] = total1[i]
return first + j, first + j + cnt - 1

```

```

def randomized_quick_sort(arr, first, last):
    if first < last:
        k = randint(first, last)
        arr[first], arr[k] = arr[k], arr[first]
        m1, m2 = partition_3(arr, first, last)
        randomized_quick_sort(arr, first, m1 - 1)
        randomized_quick_sort_under(arr, m1, m2)
        randomized_quick_sort(arr, m2 + 1, last)

```

```

def partition_3_under(arr, first, last):
    cur = arr[first][1]
    left = []
    mid = []
    right = []
    j = 0
    cnt = 0
    for i in range(first, last + 1):
        if arr[i][1] < cur:
            left.append(arr[i][1])
            j += 1
        elif arr[i][1] == cur:
            mid.append(arr[i][1])
            cnt += 1
        else:
            right.append(arr[i][1])
    total = left + mid + right
    for i in range(len(total)):
        arr[first + i][1] = total[i]

```

```

    return first + j, first + j + cnt - 1

def randomized_quick_sort_under(arr, first, last):
    if first < last:
        k = randint(first, last)
        arr[first][1], arr[k][1] = arr[k][1], arr[first][1]
        m1, m2 = partition_3_under(arr, first, last)
        randomized_quick_sort_under(arr, first, m1 - 1)
        randomized_quick_sort_under(arr, m2 + 1, last)

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as f:
        s, p = map(int, f.readline().split())
        sgms = [[int(i) for i in f.readline().split()] for i in range(s)]
        dots = list(map(int, f.readline().split()))
        randomized_quick_sort(sgms, 0, s - 1)
        with open('output.txt', 'w+') as g:
            for i in range(p):
                count = 0
                idx = 0
                while idx < s and dots[i] >= sgms[idx][0]:
                    if dots[i] <= sgms[idx][1]:
                        count += 1
                    idx += 1
                g.write(str(count) + ' ')
            print('Time:', str(process_time()), 'sec')
            print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

```

Для каждого отрезка создаётся двумерный список, в котором хранятся его начало и конец. С помощью быстрой сортировки вычисляются точки, которые принадлежат данным отрезкам.

Результат работы кода на примерах из текста задачи:

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

input.txt	output.txt
1 1 3	0 0 1
2 -10 10	
3 -100 100 0	

input.txt	output.txt
1 3 2	2 0
2 0 5	
3 -3 2	
4 7 10	
5 1 6	



Результат работы кода на максимальных и минимальных значениях:

The image shows two screenshots of a code editor interface. The top screenshot displays the 'Дополнительные задачи\input.txt' file with the following content:

```

1 3 2
2 -1000000000 1000000000
3 -1000 1000
4 -10000 10000
5 4 7

```

The corresponding 'Дополнительные задачи\output.txt' file shows:

```

1 3 3

```

The bottom screenshot displays the same 'input.txt' file with different content:

```

1 2 1
2 0 4
3 9 10
4 3

```

The corresponding 'output.txt' file shows:

```

1 1

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.109375 sec	17.537042375 KB
Пример из задачи	0.140625 sec	17.537109375 KB
Пример из задачи	0.140625 sec	17.537109375 KB
Пример из задачи	0.140625 sec	17.537109375 KB

Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	96.921875 KB
---	--------------	--------------

Вывод по задаче: в задаче я применила алгоритм быстрой сортировки для поиска принадлежащих отрезку точек.

### 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 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.

```
from random import randint
from time import process_time
from tracemalloc import start, get_traced_memory
```

```
def partition_3(arr, first, last):
    cur = arr[first][0] ** 2 + arr[first][1] ** 2
    left = []
    mid = []
    right = []
    j = 0
    cnt = 0
    for i in range(first, last + 1):
```

```

        r = arr[i][0] ** 2 + arr[i][1] ** 2
        if r < cur:
            left.append(arr[i])
            j += 1
        elif r == cur:
            mid.append(arr[i])
            cnt += 1
        else:
            right.append(arr[i])
    arr[first:last + 1] = left + mid + right
    return first + j, first + j + cnt - 1

```

```

def randomized_quick_sort(arr, first, last):
    if first < last:
        k = randint(first, last)
        arr[first], arr[k] = arr[k], arr[first]
        m1, m2 = partition_3(arr, first, last)
        randomized_quick_sort(arr, first, m1 - 1)
        randomized_quick_sort(arr, m2 + 1, last)

```

```

if __name__ == '__main__':
    start()
    with open('input.txt', 'r') as g:
        n, k = map(int, g.readline().split())
        dots = [[int(i) for i in g.readline().split()] for i in range(n)]
        randomized_quick_sort(dots, 0, n - 1)
    with open('output.txt', 'w') as g:
        for i in range(k - 1):
            g.write(str(dots[i]) + ',')
        g.write(str(dots[k - 1]))
    print('Time:', str(process_time()), 'sec')
    print('Memory usage:', str(get_traced_memory()[1] / 1024), 'KB')

```

Координаты каждой точки хранятся в двумерном списке, для них считается расстояние и сортируется быстрой сортировкой, К первых элементов списка выводятся.

Результат работы кода на примерах из текста задачи:

```
Дополнительные задачи\input.txt
1 2 1
2 1 3
3 -2 2

Дополнительные задачи\output.txt
1 [-2, 2]
```

```
Дополнительные задачи\input.txt
1 3 2
2 3 3
3 5 -1
4 -2 4

Дополнительные задачи\output.txt
1 [3, 3], [-2, 4]
```

Результат работы кода на максимальных и минимальных значениях:

```

анту\input.txt x  Дополнительные задачи\input.txt x
1 10 4 ✓
2 -2 234
3 38 44
4 -10000000000 10000000000
5 9999999 2345623
6 -222 23
7 381 441111
8 -100345 979643322
9 9444699 234666623
10 -200 24534
11 2138 49

ту\output.txt x  Дополнительные задачи\output.txt x
1 [38, 44],[-222, 23],[-2, 234],[2138, 49] ✓

```

```

анту\input.txt x  Дополнительные задачи\input.txt x
1 2 1 ✓
2 2 4
3 9 0
4
ту\output.txt x  Дополнительные задачи\output.txt x
1 [2, 4] ✓

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.140625 sec	17.560546875 KB
Пример из задачи	0.125 sec	17.244140625 KB

Пример из задачи	0.125 sec	17.244140625 KB
Верхняя граница диапазона значений входных данных из текста задачи	0.109375 sec	92.58984375 KB

Вывод по задаче: в задаче я использовала алгоритм быстрой сортировки для поиска ближайших к началу координат точек.

### **Вывод:**

В работе я вспомнила алгоритм быстрой сортировки, научилась делать его более эффективным, а также применять для решения различных практических задач.