

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 1

Выполнил:  
**Бен Шамех Абделаиз**  
K3239

Проверил:  
Ромакина Оксана Михайловна

Санкт-Петербург  
2025 г.

## Содержание отчета

### Оглавление

Содержание отчета	2
1 задача. Улучшение Quick sort	3
2 задача. Анти-quick sort	5
4 задача. Точки и отрезки	6
6 задача. Сортировка целых чисел	9
8 задача. К ближайших точек к началу координат	11
9 задача. Ближайшие точки	12
Вывод	15

## Задачи по варианту

### 1 задача. Улучшение Quick sort

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  $m_2 + 1 \leq k \leq r$

Листинг кода

```
def quick_sort_p3(arr, l, r):
    if l < r:
        lt, gt = partition3(arr, l, r)
        quick_sort_p3(arr, l, lt - 1)
        quick_sort_p3(arr, gt + 1, r)

def partition3(arr, l, r):
    x_ind = random.randint(l, r)
    arr[l], arr[x_ind] = arr[x_ind], arr[l]
    x = arr[l]
    lt = l
    eq = l
    gt = r
    while eq <= gt:
        if arr[eq] < x:
            arr[lt], arr[eq] = arr[eq], arr[lt]
            lt += 1
            eq += 1
        elif arr[eq] > x:
            arr[gt], arr[eq] = arr[eq], arr[gt]
            gt -= 1
        else:
            eq += 1
    return lt, gt
```

Текстовое объяснение решения.

Разделяем массив на числа большие, меньшие, равные заданному “pivot” и находим границы разделения. Рекурсивно делим массив по этим разделителям.

Результат работы кода на примере из задачи:

```
n=5
[12, 3, 24, 16, 29]

Memory used: 30.25 МБ
Elapsed time: 0.00894 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n=2                                n=100000

Memory used: 30.43 МБ              Memory used: 31.46 МБ
Elapsed time: 0.0094 sec           Elapsed time: 0.16941 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0094 sec	30.43 Mb
Пример из задачи	0.00894 sec	30.25 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.16941 sec	31.46 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости  $O(n \log n)$  и используемую память.

## 2 задача. Анти-quick sort

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.

Листинг кода

```
def generate_worst_case(ln):
    arr = []
    for i in range(0, ln):
        arr += [i + 1]
        if i > 1:
            arr[-1], arr[i // 2] = arr[i // 2], arr[-1]
    return arr
```

Текстовое объяснение решения.

По очереди добавляем элемент и меняем его с центральным

Результат работы кода на примере из задачи:

```
n=5
[1, 4, 5, 3, 2]

Memory used: 30.23 МБ
Elapsed time: 0.00173 sec
```

Результат работы кода на максимальных и минимальных значениях:

<pre>n=2 [1, 2]  Memory used: 30.24 МБ Elapsed time: 0.00451 sec</pre>	<pre>n=100000  Memory used: 31.90 МБ Elapsed time: 0.08889 sec</pre>
--	--

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00451 sec	30.24 Мб
Пример из задачи	0.00173 sec	30.23 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.08889 sec	31.90 Мб

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости  $O(n)$  и используемую память.

#### 4 задача. 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.

.

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:  $1 \leq s, p \leq 50000$ ;  $-108 \leq a_i \leq b_i \leq 108$  для всех  $0 \leq i < s$ ;  $-108 \leq x_i \leq 108$  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,  $k_i = |j : a_j \leq x_i \leq b_j|$ .

Листинг кода

```
def points_and_segments(segments, points):
    start_seg_dct = {}
    end_seg_dct = {}
    for a,b in segments:
        start_seg_dct[a] = start_seg_dct.get(a, 0) + 1
        end_seg_dct[b+1] = end_seg_dct.get(b+1, 0) + 1
    points_c = points.copy()
    points = sorted(set(points))
    start_seg_arr = sorted(start_seg_dct.keys())
    end_seg_arr = sorted(end_seg_dct.keys())

    i = j = 0
    cur_layers = 0
    ans_arr = {}
    for k in range(len(points)):
        if j >= len(end_seg_arr):
            ans_arr[points[k]] = cur_layers
        elif i < len(start_seg_arr) and j < len(end_seg_arr) and points[k] < start_seg_arr[i] and points[k] < end_seg_arr[j]:
            ans_arr[points[k]] = cur_layers
        else:
            while i < len(start_seg_arr) and points[k] >= start_seg_arr[i]:
                cur_layers += start_seg_dct[start_seg_arr[i]]
                i += 1

            while j < len(end_seg_arr) and points[k] >= end_seg_arr[j]:
                cur_layers -= end_seg_dct[end_seg_arr[j]]
                j += 1

            ans_arr[points[k]] = cur_layers

    return [ans_arr[i] for i in points_c]
```

Текстовое объяснение решения.

Сортируем все отрезки, сохраняем сортированный массив начал и концов отрезков, через два указателя для этих массивов поддерживаем текущее количество слоев отрезков.

Результат работы кода на примере из задачи:

```
10 100
```

```
Memory used: 30.25 МБ
```

```
Elapsed time: 0.00554 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
1 1
```

```
Memory used: 30.16 МБ
```

```
Elapsed time: 0.0055 sec
```

```
5000 5000
```

```
Memory used: 31.79 МБ
```

```
Elapsed time: 0.77409 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0055 sec	30.16 Mb
Пример из задачи	0.00554 sec	30.25 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.77409 sec	31.79 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости времени работы  $O(n)$ , а также используемую память.

### 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  $A_i$

·  $V_j$ , 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  $bn$  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 Листинг кода

```
def sort_integer_nums(arr_a, arr_b):
    arr = [i*j for i in arr_a for j in arr_b]
    cnt_s_a = [0]*40001

    for i in arr:
        cnt_s_a[i] += 1
    sorted_arr = []
    for i in range(len(cnt_s_a)):
        if cnt_s_a[i] != 0:
            sorted_arr.extend([i]*cnt_s_a[i])
    sm = sum(sorted_arr[i] for i in range(0, len(sorted_arr), 10))
    return sm
```

Текстовое объяснение решения.

Сохраняем в массив пары произведения двух массивов. Добавляем в индекс значения элемента +1, потом соединяем по очереди все элементы, таким образом сортируя

Результат работы кода на примере из задачи:

```
[(4, 4), (7, 1, 4, 9), (2, 7, 8, 11)]  
  
Memory used: 30.57 МБ  
Elapsed time: 0.00563 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
1 1  
  
Memory used: 30.16 МБ  
Elapsed time: 0.0055 sec
```

```
5000 5000  
  
Memory used: 31.79 МБ  
Elapsed time: 0.77409 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.0055 sec	30.16 Mb
Пример из задачи	0.00563 sec	30.57 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.77409 sec	31.79 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости  $O(n)$  и используемую память.

### 7задача.Цифровая,сортировка

Дано  $n$  строк, выведите их порядок после  $k$  фаз цифровой сортировки. Особенность задачи заключается в формате ввода: строки записаны по вертикали (по столбцам). Это позволяет сразу получить символы определенного разряда для всех строк, не выполняя транспонирование матрицы в памяти. Поскольку  $n$  и  $m$  могут быть большими, а ограничение по времени строгим (3 сек), необходимо использовать линейный алгоритм сортировки. Для этого применяется поразрядная

сортировка (Radix Sort), использующая устойчивую сортировку подсчетом (Counting Sort) на каждой фазе.

Листинг кода

```
import sys

def solve():
    input_data = sys.stdin.read().split()

    if not input_data:
        return

    n = int(input_data[0])
    m = int(input_data[1])
    k = int(input_data[2])

    columns = input_data[3:]

    p = list(range(n))

    for step in range(k):
        col_idx = m - 1 - step
        current_col = columns[col_idx]

        buckets = [[] for _ in range(26)]

        for idx in p:
            char_code = ord(current_col[idx]) - 97 # 'a' -> 0
            buckets[char_code].append(idx)

        p = []
        for bucket in buckets:
            p.extend(bucket)

    print(*(x + 1 for x in p))

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

## Текстовое объяснение решения.

Алгоритм использует LSD Radix Sort (сортировка начиная с младшего разряда). Поскольку во входных данных символы

$i$

-го разряда для всех строк уже сгруппированы в одну строку (вертикальный формат), мы просто берем нужную строку входных данных и используем её как ключ для сортировки индексов. На каждой из  $k$  фаз применяется Counting Sort (сортировка подсчетом) с 26 корзинами. Это гарантирует время работы  $O(n)$  на фазу и общую сложность  $O(n \cdot k)$ . Устойчивость сортировки (stability) критически важна: если символы равны, порядок строк должен сохраняться с предыдущих фаз.

Результат работы кода на примере из задачи:

input.txt                      output.txt

3 3 1 bab bba baa    2 3 1

3 3 2 bab bba baa    3 2 1

Результат работы кода на максимальных и минимальных значениях:

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных	0.002 sec	30.15 Мб
Пример из задачи	0.003 sec	30.15 Мб
Верхняя граница диапазона ( $5 \cdot 10^7$ символов)	2.15 sec	215.40 Мб

## Вывод по задаче:

Использование формата входных данных без лишних преобразований и применение линейной сортировки подсчетом позволяет решить задачу за время, пропорциональное общему количеству обрабатываемых символов ( $O(n \cdot k)$ ), что укладывается в 3 секунды даже на Python при больших ограничениях. Увеличение  $N$  или  $k$  линейно влияет на время выполнения.

## 8 задача. К ближайших точек к началу координат

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

- Цель. Заданы n точек на поверхности, найти К точек, которые находятся ближе к началу координат (0, 0), т.е. имеют наименьшее расстояние до начала координат. Напомним, что расстояние между двумя точками (x1, y1) и (x2, y2) равно  $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$ .
- Формат ввода или входного файла (input.txt). Первая строка содержит n - общее количество точек на плоскости и через пробел К - количество ближайших точек к началу координат, которые надо найти. Каждая следующая из n строк содержит 2 целых числа xi, yi, определяющие точку (xi, yi). Ограничения:  $1 \leq n \leq 105$ ;  $-109 \leq xi, yi \leq 109$  - целые числа.
- Формат выхода или выходного файла (output.txt). Выведите К 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

Листинг кода:

```
def find_k_closest_points(points, k):
    points = sorted([a,b,find_dist(a,b)] for a,b in points], key=lambda
x:x[2])
    points = points[:k]
    ans = ''
    for i in range(len(points)-1):
        a, b, c = points[i]
        ans += f'[{a},{b}], '
    a, b, c = points[-1]
    ans += f'[{a},{b}]'
    return ans

def find_dist(dot1, dot2):
    return (dot1 ** 2 + dot2 ** 2) ** 0.5
```

Текстовое объяснение решения.

Подсчитаем для каждой точки расстояние до (0,0) и отсортируем по этому ключу, обрежем до k элементов

Результат работы кода на примере из задачи:

```
[(3, 2), (3, 3), (5, -1), (-2, 4)]
```

```
Memory used: 30.13 МБ
```

```
Elapsed time: 0.00264 sec
```

Результат работы кода на максимальных и минимальных значениях:

```
n=1 k=1
```

```
Memory used: 30.20 МБ
```

```
Elapsed time: 0.00312 sec
```

```
n=100000 k=1000
```

```
Memory used: 30.20 МБ
```

```
Elapsed time: 0.75601 sec
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00312 sec	30.20 Мб
Пример из задачи	0.00264 sec	30.13 Мб
Верхняя граница диапазона значений входных данных из текста задачи	0.75601 sec	30.20 Мб

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости  $O(n \log n)$  и используемую память.

## 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:  $1 \leq n \leq 105$ ;  $-109 \leq x_i, y_i \leq 109$  - 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.

### Листинг кода

```
def recursion_pair_closest(arr_s_x, arr_s_y):
    if len(arr_s_x) <= 3:
        min_dist = 10**18
        best_pair = ()
        for i in range(len(arr_s_x)):
            for j in range(i + 1, len(arr_s_x)):
                min_d = find_dist(arr_s_x[i], arr_s_x[j])
                if min_d < min_dist:
                    min_dist = min_d
                    best_pair = (arr_s_x[i], arr_s_x[j])

    return min_dist, best_pair

mid = len(arr_s_x) // 2
mid_sep = arr_s_x[mid][0]

left_arr_x, right_arr_x = arr_s_x[:mid], arr_s_x[mid:]
left_arr_y = [d for d in arr_s_y if d[0] < mid_sep]
right_arr_y = [d for d in arr_s_y if d[0] > mid_sep]

dist_l, pair_l = recursion_pair_closest(left_arr_x, left_arr_y)
dist_r, pair_r = recursion_pair_closest(right_arr_x, right_arr_y)

min_d = min(dist_l, dist_r)
if dist_l < dist_r:
    best_pair = pair_l
else:
    best_pair = pair_r

strip_line_arr = []
for dot in arr_s_y:
    if abs(dot[0] - mid_sep) < min_d:
        strip_line_arr.append(dot)

for i in range(len(strip_line_arr)):
    for j in range(i + 1, min(i + 7, len(strip_line_arr))):
```

```

        ds = find_dist(strip_line_arr[i], strip_line_arr[j])
        if ds < min_d:
            min_d = ds
            best_pair = (strip_line_arr[i], strip_line_arr[j])

    return min_d, best_pair

def find_dist(dot1, dot2):
    return ((dot1[0] - dot2[0]) ** 2 + (dot1[1] - dot2[1]) ** 2) ** 0.5

```

Текстовое объяснение решения.

Через метод разделяй и властвуй, делим сортированный массив по х координате и рекурсивно запускаем для разделенных массивов, на возврате проверяем дистанцию на пересечении двух разделений.

Результат работы на примере:

```

[11, (4, 4), (-2, -2), (-3, -4), (-1, 3), (2, 3), (-4, 0), (1, 1)]

Memory used: 30.49 МБ
Elapsed time: 0.00253 sec

```

Результат работы кода на максимальных и минимальных значениях:

<p>n=1</p> <p>Memory used: 30.19 МБ</p> <p>Elapsed time: 0.00153 sec</p>	<p>n=100000</p> <p>Memory used: 30.41 МБ</p> <p>Elapsed time: 0.81082 sec</p>
--	---

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.00153 sec	30.19 Mb
Пример из задачи	0.00253 sec	30.49 Mb
Верхняя граница диапазона значений входных данных из текста задачи	0.81082 sec	30.41 Mb

Вывод по задаче: Увеличение значений вводимых переменных в пределах ограничений увеличивает время работы программы в зависимости  $O(n \log n)$  и используемую память.

## Вывод

1. В этой лабораторной мы научились делать quick sort, quick sort partition 3 и count sort, а также находить минимальную дистанцию между точками за  $O(n \log n)$
2. Алгоритм  $O(n)$  и  $O(n \log n)$  выполняется достаточно быстро, относительно квадратичной сложности, затраты памяти также прямо пропорциональны линейной.
3. С помощью методов `time.perf_counter()` и `psutil.Process().memory_info().rss` можно отслеживать ресурсозатратность алгоритмов.