

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, сортировка за линейное время  
Вариант 1

Выполнила:

Гайдук А. С.

К3241

Проверила:

Ромакина О. М.

Санкт-Петербург

2024 г.

## Содержание отчета

Содержание отчета.....	2
Задачи по варианту .....	3
Задача №1. Улучшение Quick sort.....	3
Задача №4. Точки и отрезки .....	7
Задача №7. Цифровая сортировка.....	10
Дополнительные задачи .....	14
Задача №3. Сортировка пугалом .....	14
Задача №5. Индекс Хирша.....	16
Вывод .....	20

## 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 её, создав несколько случайных массивов, подходящих под параметры.

#### Листинг кода

```
import tracemalloc
import time

tracemalloc.start()

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    element = arr[0]
    less = [x for x in arr[1:] if x <= element]
    greater = [x for x in arr[1:] if x > element]

    return quick_sort(less) + [element] + quick_sort(greater)

with open("input.txt", "r") as file:
    n = int(file.readline().strip())
    arr = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

sorted_arr = quick_sort(arr)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open("output.txt", "w") as file:
    file.write(" ".join(map(str, sorted_arr)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое  
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

#### Текстовое объяснение решения:

Для решения данной задачи я использую двухстороннее разделение. Я определяю опорный элемент (в моем случае элемент нулевого индекса) и сравниваю остальные элементы массива с ним, формируя два новых массива «less» и «greater». Далее я снова вызываю этот метод для сортировки новых массивов.

Результат работы кода на примерах из текста задачи:

task1.py input.txt x output.txt

```
1 5
2 2 3 9 2 2
```

task1.py input.txt output.txt x

```
1 2 2 2 3 9
```

Результат работы кода на максимальных и минимальных значениях:

task1.py input.txt x output.txt

```
1 1
2 0
```

task1.py input.txt output.txt x

```
1 0
```

task1.py input.txt x output.txt

```
1 10000
2 48354615 512775914 -642704208 258
   687965154 -222729385 32543112 70
```

task1.py input.txt output.txt x

```
1 -999777885 -999777848 -999679256 -999676
   -998481002 -998466562 -998417433 -99839
   -997559653 -997462630 -997454621 -99735
   -996696343 -996664307 -996594473 -99626
   -995647352 -995422044 -995053206 -99498
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из	0.000009 сек	0.037525 МВ

текста задачи		
Пример из задачи	0.000033 сек	0.037583 МВ
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ из текста задачи	0.059899 сек	0.983461 МВ

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 трёхстороннее (смотри в Лекции 3, слайд 17).

### Листинг кода

```
import tracemalloc
import time

tracemalloc.start()

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    element = arr[0]
    less = [x for x in arr[1:] if x < element]
    equal = [x for x in arr if x == element]
    greater = [x for x in arr[1:] if x > element]

    return quick_sort(less) + equal + quick_sort(greater)

with open("input.txt", "r") as file:
    n = int(file.readline().strip())
    arr = list(map(int, file.readline().strip().split()))

start_time = time.perf_counter()

sorted_arr = quick_sort(arr)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open("output.txt", "w") as file:
    file.write(" ".join(map(str, sorted_arr)))

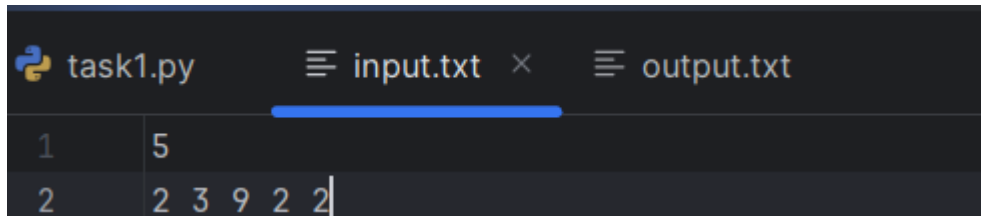
print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
```

```
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

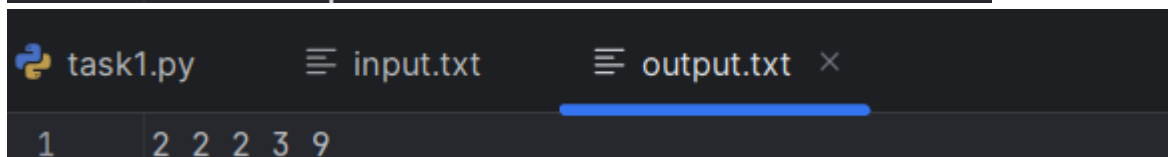
### Текстовое объяснение решения:

Для того чтобы перейти от двухстороннего деления на трехстороннее, я просто определила значение опорного элемента и сразу вынесла его из исходного массива.

### Результат работы кода на примерах из текста задачи:

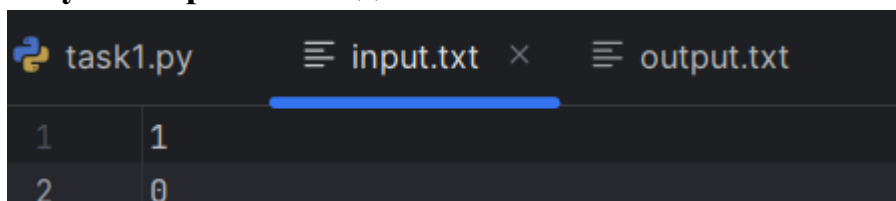


```
task1.py  input.txt  output.txt
1      5
2      2 3 9 2 2
```

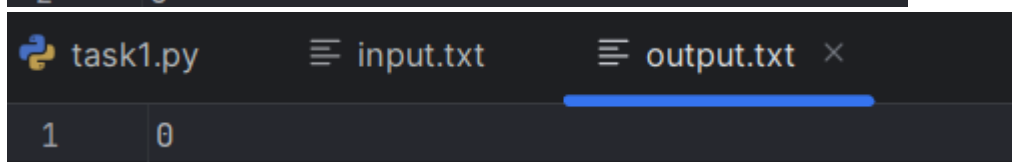


```
task1.py  input.txt  output.txt
1      2 2 2 3 9
```

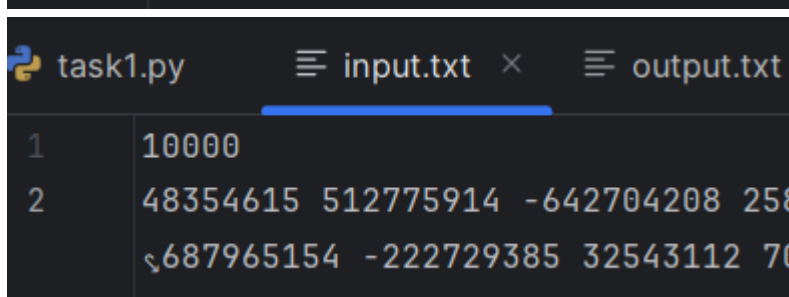
### Результат работы кода на максимальных и минимальных значениях:



```
task1.py  input.txt  output.txt
1      1
2      0
```



```
task1.py  input.txt  output.txt
1      0
```



```
task1.py  input.txt  output.txt
1      10000
2      48354615 512775914 -642704208 258
        687965154 -222729385 32543112 70
        500770671 066560017 570000100
```

```
task1.py  input.txt  output.txt ×
1  -999777885 -999777848 -999679256 -999676
   -998481002 -998466562 -998417433 -99839
   -997559653 -997462630 -997454621 -99735
   -996696343 -996664307 -996594473 -99626
   -995647352 -995422044 -995053206 -99498
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000008 сек	0.037525 MB
Пример из задачи	0.000027 сек	0.037583 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.062778 сек	0.983461 MB

### Вывод по задаче:

В ходе выполнения данной задачи я ознакомилась с реализацией алгоритма Quick sort при помощи двухстороннего и трехстороннего разделения массива.

### 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.

### Листинг кода

```
import tracemalloc
import time

tracemalloc.start()

def points_counter(p, sections, points):
    section_info = []
    counter = 0
    result = [0] * p
    for section in sections:
        section_info.append([section[0], "start"])
        section_info.append([section[1], "end"])
    for i, point in enumerate(points):
        section_info.append([point, "point", i])
    section_info.sort(key=lambda x: (x[0], x[1] == "end", x[1] ==
"point"))
    for element in section_info:
        if element[1] == 'start':
            counter += 1
        elif element[1] == 'end':
            counter -= 1
        else:
            result[element[2]] = counter
    return result

with open("input.txt", "r") as file:
    s, p = map(int, file.readline().split())
    sections = [list(map(int, file.readline().split())) for i in
range(s)]
    points = list(map(int, file.readline().split()))

start_time = time.perf_counter()

result = points_counter(p, sections, points)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open("output.txt", "w") as file:
    file.write(" ".join(map(str, result)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

### Текстовое объяснение решения



Я создаю массив `section_info`, куда добавляются концы отрезка и точка с её индексом. Далее события из этого массива сортируются, и я прохожу по ним циклом. Если я встречаю событие со флагом «старт», то к счетчику прибавляется единица, если же встречаю «конец», то от счетчика отнимается единица. Когда встречается точка, то в массив `result` сохраняется текущее значение счетчика.

**Результат работы кода на примерах из текста задачи:**

task4.py	input.txt	output.txt
1	2 3	
2	0 5	
3	7 10	
4	1 6 11	

task4.py	input.txt	output.txt
1	1 0 0	

**Результат работы кода на максимальных и минимальных значениях:**

task4.py	input.txt	output.txt
1	1 1	
2	0 1	
3	1	

task4.py	input.txt	output.txt
1	1	

1	50000 50000
2	-589071 8118713
3	-19369412 -18211820
4	42682890 71165124
5	-74895083 -9733034
6	-25633720 30195318

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000011 сек	0.037732 МВ
Пример из задачи	0.000038 сек	0.037910 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.569505 сек	37.383607 МВ

### Вывод по задаче:

В ходе решения этой задачи я научилась реализовывать алгоритм подсчета количества отрезков, содержащих каждую точку из заданного множества.

## Задача №7. Цифровая сортировка

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

### Листинг кода

```
import tracemalloc
import time

tracemalloc.start()

def radix_sort_vertical(n, m, k, strings):
    indices = list(range(n))
    for phase in range(k):
        current_position = m - 1 - phase
        indices.sort(key=lambda i: strings[current_position][i])
    return [i + 1 for i in indices]

with open("input.txt", "r") as file:
    n, m, k = map(int, file.readline().strip().split())
```

```

strings = [file.readline().strip() for _ in range(m)]

start_time = time.perf_counter()

result = radix_sort_vertical(n, m, k, strings)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open("output.txt", "w") as file:
    file.write(" ".join(map(str, result)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

### Текстовое объяснение решения:

Я начала с инициализации массива индексов строк. Затем, в  $k$  фазах, сортируются индексы строк по текущему символу. Позиция символа для сортировки начинается с последнего символа строки. В конце к каждому индексу я добавила единицу, чтобы индексирование началось с единицы.

### Результат работы кода на примерах из текста задачи:

input.txt	output.txt	task7.py
1	3 3 1	
2	bab	
3	bba	
4	baa	

input.txt	output.txt	task7.py
1	2 3 1	

input.txt	output.txt	task7.py
1	3 3 2	
2	bab	
3	bba	
4	baa	

```

input.txt  output.txt  task7.py
1 3 2 1|

```

```

input.txt  output.txt  task7.py
1 3 3 3
2 bab
3 bba
4 baa

```

```

input.txt  output.txt  task7.py
1 2 3 1

```

Результат работы кода на максимальных и минимальных значениях:

```

input.txt  output.txt  task7.py
1 1 1 1
2 b|

```

```

input.txt  output.txt  task7.py
1 1|

```

```

input.txt  output.txt  task7.py
1 10000 100000 100000
2 BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
  BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

```

```

input.txt  output.txt  task7.py
1 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 1
  39 40 41 42 43 44 45 46 47 48 49 50
  74 75 76 77 78 79 80 81 82 83 84 85
  107 108 109 110 111 112 113 114 115

```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000010 сек	0.037609 МВ
Пример из задачи	0.000016 сек	0.037785 МВ
Пример из задачи	0.000011 сек	0.037785 МВ
Пример из задачи	0.000017 сек	0.037785 МВ
Верхняя граница диапазона значений входных данных из текста задачи	1.594735 сек	16.886607 МВ

**Вывод по задаче:**

В данной задаче я реализовала цифровую сортировку строк в вертикальном формате после k фаз.

## Дополнительные задачи

### Задача №3. Сортировка пугалом

«Сортировка пугалом» — это старинная народная потеха. Участнику под верхнюю одежду продевают деревянную палку, так что у него оказываются раскинуты 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 с  $i$ -й и  $i+k$ -й позиции), развернуть их и поставить их обратно в ряд, таким образом поменяв их местами.

**Задача:** рассортировать матрёшки по неубывающему размеру. Может ли участник сделать это?

#### Листинг кода

```
import tracemalloc
import time

tracemalloc.start()

def scarecrow_sort(n, k, sizes):
    groups = [[] for _ in range(k)]
    for i in range(n):
        groups[i % k].append(sizes[i])
    for group in groups:
        group.sort()
    sorted_sizes = []
    for i in range(n):
        sorted_sizes.append(groups[i % k][i // k])
    return "ДА" if sorted_sizes == sorted(sizes) else "НЕТ"

with open("input.txt", "r", encoding="utf-8") as file:
    n, k = map(int, file.readline().split())
    sizes = list(map(int, file.readline().split()))

start_time = time.perf_counter()

result = scarecrow_sort(n, k, sizes)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open("output.txt", "w", encoding="utf-8") as file:
    file.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

#### Текстовое объяснение решения

Для решения этого задания я решила создать k групп, в которую матрешки добавляются на основании их индекса, далее каждая группа сортируется. Я создаю массив отсортированных групп, куда поочередно добавляются элементы каждой группы. Если массив равен массиву с отсортированными размерами, то ответ «ДА».

**Результат работы кода на примерах из текста задачи:**

```
task3.py  input.txt  output.txt
1         3 2
2         2 1 3
```

```
task3.py  input.txt  output.txt
1         НЕТ
```

```
task3.py  input.txt  output.txt
1         5 3
2         1 5 3 4 1
```

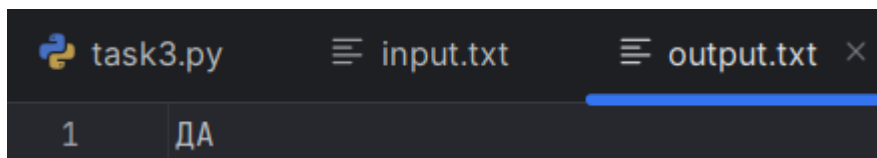
```
task3.py  input.txt  output.txt
1         ДА
```

**Результат работы кода на максимальных и минимальных значениях:**

```
task3.py  input.txt  output.txt
1         1 1
2         -10000000000
```

```
task3.py  input.txt  output.txt
1         ДА
```

```
task3.py  input.txt  output.txt
1         100000 100000
2         10000000000 10000000000 10000000000 10000000000
        10000000000 10000000000 10000000000
```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000013 сек	0.018138 МВ
Пример из задачи	0.000015 сек	0.018126 МВ
Пример из задачи	0.000021 сек	0.018134 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.264935 сек	15.208153 МВ

### Вывод по задаче:

В данной задаче я ознакомилась с алгоритмом сортировки «пугалом», реализовав его через группировку матрешек по индексам.

### 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 из *h* 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.

### Листинг кода

```
import tracemalloc
import time
```



```

tracemalloc.start()

def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    element = arr[0]
    less = [x for x in arr[1:] if x < element]
    equal = [x for x in arr if x == element]
    greater = [x for x in arr[1:] if x > element]

    return quick_sort(less) + equal + quick_sort(greater)

def h_index(citations):
    n = len(citations)
    sorted_citations = quick_sort(citations)
    index_h = 0
    for i in range (n-1, -1, -1):
        if sorted_citations[i] >= index_h + 1:
            index_h += 1
            continue
        break
    return index_h

with open ('input.txt', 'r') as file:
    citations = list(map(int, file.read().strip().replace(',', ' '
    ').split()))

start_time = time.perf_counter()

result = h_index(citations)

current, peak = tracemalloc.get_traced_memory()
end_time = time.perf_counter()

with open ('output.txt', 'w') as file:
    file.write(str(result))

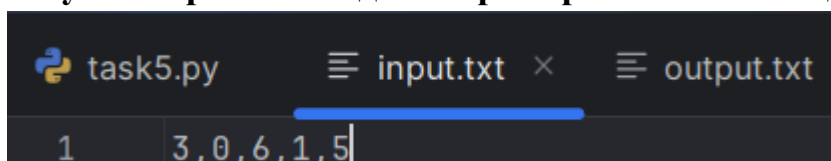
print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

## Текстовое объяснение решения

Для вычисления индекса Хирша я создала функцию `h_index`, где список цитирований сортируется через Quick Sort, а затем цитирования перебираются в обратном порядке. Если текущее цитирование больше или равно текущему индексу Хирша, то значение `h` увеличивается на единицу.

## Результат работы кода на примерах из текста задачи:



```
task5.py input.txt output.txt
1 3
```

```
task5.py input.txt output.txt
1 1, 3, 1
```

```
task5.py input.txt output.txt
1 1
```

**Результат работы кода на максимальных и минимальных значениях:**

```
task5.py input.txt output.txt
1 0
```

```
task5.py input.txt output.txt
1 0
```

```
task5.py input.txt output.txt
1 1000 1000 1000 1000 1000 1000 1000
```

```
task5.py input.txt output.txt
1 1000
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000009 сек	0.030395 МВ
Пример из задачи	0.000030 сек	0.030395 МВ
Пример из задачи	0.000026 сек	0.030395 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.001212 сек	0.477995 МВ

**Вывод по задаче:**

В ходе выполнения этой задачи я написала алгоритм для вычисления индекса Хирша через Quick Sort.

## **Вывод**

В ходе выполнения данной лабораторной работы я изучила что такое Quick Sort и смогла применить ее на практике.