

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

Отчет по лабораторной работе №2
по курсу «Алгоритмы и структуры данных»
Тема: Быстрая сортировка, сортировки за линейное время
Вариант 12

Выполнил:
Колпаков А.С.
К3139

Проверил:
Афанасьев А.В

Санкт-Петербург
2024 г.

Содержание отчета

Содержание отчета	2
Задачи по варианту	3
Задача №1. Улучшение Quick sort	3
Задача №3. Сортировка пугалом	6
Задача №8. К ближайших точек к началу координат	9
Дополнительные задачи	12
Задача №4. Точки и отрезки	12
Задача №5. Индекс Хирша	15
Задача №6. Сортировка целых чисел	17
Вывод	20

Задачи по варианту

Задача №1. Улучшение Quick sort

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 сетях 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?

Листинг кода:

```
import random

def partition3(A, l, r):
    x = A[l]
    m1 = l
    m2 = r
    i = l

    while i <= m2:
        if A[i] < x:
            A[m1], A[i] = A[i], A[m1]
            m1 += 1
            i += 1
```

```

        elif A[i] > x:
            A[i], A[m2] = A[m2], A[i]
            m2 -= 1
        else:
            i += 1

    return m1, m2

def randomized_quicksort(A, l, r):
    if l < r:
        k = random.randint(l, r)
        A[l], A[k] = A[k], A[l]
        m1, m2 = partition3(A, l, r)
        randomized_quicksort(A, l, m1 - 1)
        randomized_quicksort(A, m2 + 1, r)
    return A

if __name__ == '__main__':
    randomized_quicksort()

```

Текстовое объяснение решения:


1. Задаем функцию partition3, которая разбивает массив на три части по отношению к опорному элементу и возвращает индексы m1, m2, обозначающие диапазон элементов равных x.
2. Далее в функции randomized_quicksort случайным образом выбирается опорный элемент и рекурсивно, пока в массиве не окажется меньше двух элементов, сортирует две части массива от l до m1-1 и от m2+1 до r.

Результат работы кода:

 *input.txt*

itmo > algo > lab3 > task1 > textf > input.txt

```
1  12
2 -1 -5 3 43 -12 5 1 8 10 1 -1 -3
```

 *output.txt*

itmo > algo > lab3 > task1 > textf > output.txt

```
1 -12 -5 -3 -1 -1 1 1 3 5 8 10 43
```

Тест примера

Время работы: 0.0008746250023250468 секунд

Память: 0.013306617736816406 МБ

Худший случай

Время работы: 0.09210199999506585 секунд

Память: 0.002410888671875 МБ

Средний случай

Время работы: 0.10059174999332754 секунд

Память: 0.00266265869140625 МБ

Лучший случай

Время работы: 0.08807579100539442 секунд

Память: 0.00260162353515625 МБ

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.08807579100539442 секунд	0.00260162353515625 МБ
Пример из задачи	0.0008746250023250468 секунд	0.013306617736816406 МБ
Медиана диапазона значений входных данных из текста задачи	0.10059174999332754 секунд	0.00266265869140625 МБ
Верхняя граница диапазона значений входных данных из текста задачи	0.09210199999506585 секунд	0.002410888671875 МБ

Вывод по задаче:

Программа успешно реализует алгоритм быстрой сортировки. Тестирование показало правильную работу алгоритма.

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?

Листинг кода:

```
def pugalo_sort(A, n, k):
    sorted_flag = False
    while not sorted_flag:
        sorted_flag = True
```

```

        for i in range(0, n - k):
            if A[i] > A[i + k]:
                A[i], A[i + k] = A[i + k], A[i]
                sorted_flag = False
        return "ДА" if A == sorted(A) else "НЕТ"

if __name__ == '__main__':
    pugalo_sort()

```

Текстовое объяснение решения:

1. В функции pugalo_sort задаем флаг равный False.
2. Далее в цикле while задаем цикл for, который проходит по всем значениям от 0 до n-k элемента. Если элемент i больше i+k, то заменяем его и флаг задаем равным false, что означает, что элементы в исходном массиве еще можно заменить.
3. Если полученный массив после действия алгоритма равен отсортированному массиву, то возвращаем ДА, в ином случае НЕТ.

Результат работы кода на примерах из текста задачи:

```
input.txt
itmo > algo > lab3 > task3 > te
1 5 3
2 1 5 3 4 1
```

```
output.txt
itmo > algo > lab3 > task3
1 Д А
```

Тест примера
Время работы: 0.0005821660015499219 секунд
Память: 0.013286590576171875 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.0005821660015499219 секунд	0.013286590576171875 МБ

Вывод по задаче:

Программа успешно реализует проверку действия алгоритма “Сортировки пугалом”.

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

Листинг кода:

```
from task1.src.task2 import randomized_quicksort

def k_closest_points(arr, k):
    points = []
    for x,y in arr:
        dist = int((x**2 + y**2)**0.5)
        points.append([dist, [x,y]])
    points_sort = randomized_quicksort(points, 0,
len(points) - 1)
    return [x[1] for x in points_sort[0:k]]

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

Текстовое объяснение решения:

1. В функции `k_closest_points` заполняем массив `points` расстоянием каждой точки до начала координат, также записываем и сами координаты.
2. Далее, используя уже написанный нами алгоритм быстрой сортировки, сортируем массив дистанций точек по возрастанию.
3. В итоге выводим k ближайших к началу координат точек.

Результат работы кода на примерах из текста задачи:

```
input.txt
itmo > algo > lab3 > task8
1 3 2
2 3 3
3 5 -1
4 -2 4
```

```
output.txt
itmo > algo > lab3 > task8 > textf > output.txt
1 [[-2, 4], [3, 3]]
```

Тест примера
Время работы: 0.0007237500030896626 секунд
Память: 0.013368606567382812 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.0007237500030896626 6 секунд	0.013368606567382812 МБ

Вывод по задаче:

Программа эффективно, пользуясь формулой для нахождения расстояния между двумя точками, находит заданное количество наиболее близких к началу координат точек.

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.

Листинг кода:

```
def count_interval(data):
    s, p = data[0][0], data[0][1]
    intervals = [x for x in data[1:1+s]]
    points = data[-1]

    coordinates = []
    result = {}

    for st, end in intervals:
        coordinates.append([st, "L"])
        coordinates.append([end, "R"])
    for point in points:
        coordinates.append([point, "P"])
        result[point] = 0

    coordinates.sort()

    active = 0
```

```
for pos, coord_type in coordinates:
    if coord_type == "L":
        active += 1
    elif coord_type == "R":
        active -= 1
    elif coord_type == "P":
        result[pos] = active

return [result[point] for point in points]

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

Текстовое объяснение решения:

1. В функции `count_interval` циклом `for` записываем начало и конец интервалов в общий массив координат, где начало обозначается "L", а конец "R".
2. Аналогично записываем координаты точек, обозначая их буквой "P".
3. После чего сортируем массив координат.
4. После чего циклом `for` проходим по всем элементам массива координат. Если координата - это начало интервала, то прибавляем переменной активных интервалов +1, в ином случае убавляем.
5. Если координата - это точка, то записываем в переменную `result` значение текущих активных интервалов.
6. В итоге возвращаем количество активных интервалов у каждой точки.

Результат работы кода на примерах из текста задачи:

```
task.py M input.txt
itmo > algo > lab3 > task4 > textf > input.txt
1 3 2
2 0 5
3 -3 2
4 7 10
5 1 6
```

```
task.py M × output.txt
itmo > algo > lab3 > task4 > textf > o
1 2 0
```

Тест примера
Время работы: 0.00026091600011568516 секунд
Память: 0.013411521911621094 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.00026091600011568516 секунд	0.013411521911621094 МБ

Вывод по задаче:

Алгоритм поиска количества интервалов каждой точки, эффективно, пользуясь отсортированным массивом границ, выполняет подсчет количества отрезков, содержащих определенную точку.

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).

Листинг кода:

```
def hirsh_index(arr):  
    arr.sort(reverse=True)  
    h = 0  
    for i, n in enumerate(arr):  
        if n >= i + 1:  
            h = i + 1  
        else:  
            break  
    return h  
  
if __name__ == '__main__':  
    hirsh_index()
```

Текстовое объяснение решения:

1. Задаем функцию `hirsh_index`, в которой сортируем заданный массив цитирований по убыванию.
2. После чего циклом `for` проходим по элементам массива, используя функцию `enumerate`, получая сам элемент массива и его индекс.
3. Если текущий элемент больше чем его индекс + 1, тогда записываем его индекс + 1, как потенциального кандидата на индекс хирша.
4. В ином случае выходим из цикла и возвращаем сам искомое значение.

Результат работы кода на примерах из текста задачи:

```
input.txt
itmo > algo > lab3 > task5 >
1 3 0 6 1 5
```

```
output.txt M
itmo > algo > lab3 > task5 > te
1 | 3
```

```
Тест примера
Время работы: 0.0023491249958169647 секунд
Память: 0.013243675231933594 МБ
```

	Время выполнения	Затраты памяти
--	------------------	----------------

Пример из задачи	0.002349124995816964 7 секунд	0.013243675231933594 МБ
------------------	----------------------------------	----------------------------

Вывод по задаче:

Алгоритм поиска индекса Хирша эффективно находит искомое значение, что подтверждают результаты тестов.

Задача №6. 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

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.

Листинг кода:

```
from task1.src.task2 import randomized_quicksort

def sum_of_tenth(A, B):
    C = []
```

```

for b in B:
    for a in A:
        C.append(a * b)
    randomized_quicksort(C, 0, len(C) - 1)
    sum_of_tenth = sum(C[i] for i in range(0,
len(C), 10))

    return sum_of_tenth

if __name__ == '__main__':
    sum_of_tenth()

```

Текстовое объяснение решения:

1. В функции `sum_of_tenth` задаем пустой массив `C` и заполняем его произведениями всех элементов массива `A` на `B`.
2. После чего сортируем уже созданным алгоритмом сортировки `randomized_quicksort` массив `C`.
3. В конце суммируем каждый 10 элемент в массиве `C` и возвращаем результат.

Результат работы кода на примерах из текста задачи:

```
input.txt
itmo > algo > lab3 > task6 > textf >
1    4  4
2    7  1  4  9
3    2  7  8  11
```

```
output.txt
itmo > algo > lab3 > task6 > textf
1    51
```

Тест примера
Время работы: 0.0007642919954378158 секунд
Память: 0.013332366943359375 МБ

	Время выполнения	Затраты памяти
Пример из задачи	0.0007642919954378158 секунд	0.013332366943359375 МБ

Вывод по задаче:

Данный алгоритм эффективно находит сумму каждого 10 элемента массива произведений элементов двух заданных списков.

Вывод

В ходе лабораторной работы были изучены алгоритмы, использующие метод "Разделяй и властвуй", а также разные методы сортировки: быстрая сортировка и сортировки с линейной сложностью. Научились применять подходящий метод в зависимости от условий задачи. Работа позволила лучше понять, как работают различные алгоритмы сортировки и в каких случаях они наиболее эффективны.