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 2

Выполнила:

Азизова Наида Элимизовна

К3140

Проверил(а):

Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание отчёта

[Задания 3](#_Toc185373434)

[Задача №1. Улучшение Quick Sort 3](#_Toc185373435)

[Задача №3. Сортировка пугалом 5](#_Toc185373436)

[Задача №4. Точки и отрезки 6](#_Toc185373437)

[Задача №5. Индекс Хирша 9](#_Toc185373438)

[Задача №6. Сортировка целых чисел 10](#_Toc185373439)

[Задача №8. К ближайших точек к началу координат 12](#_Toc185373440)

[Вывод 14](#_Toc185373441)

# Задания

## Задача №1. Улучшение Quick Sort

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Решение:

import random  
import utils  
import os  
def partition(A, l, r):  
 x = A[l]  
 j = l  
 for i in range(l+1, r+1):  
 if A[i] <= x:  
 j += 1  
 A[j], A[i] = A[i], A[j]  
 A[l], A[j] = A[j], A[l]  
 return j  
  
def randomized\_quicksort(A,l,r):  
 if l < r:  
 k = random.randint(l, r)  
 A[l], A[k] = A[k], A[l]  
 m = partition(A, l, r)  
 randomized\_quicksort(A, l, m-1)  
 randomized\_quicksort(A, m+1, r)  
 return A  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task1', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = randomized\_quicksort(data[1], 0, data[0] - 1)  
 utils.print\_task\_data(3, 1, data, res)

Текстовое объяснение решения:

 **Функция partition(A, l, r)**:

* Эта функция выполняет операцию разделения массива (partition) вокруг опорного элемента.
* Опорный элемент выбирается как первый элемент массива (x = A[l]).
* Перемещаем элементы, меньшие или равные опорному элементу, в левую часть массива.
* В конце меняем местами опорный элемент с элементом в позиции j, которая указывает на последний элемент, меньший или равный опорному.
* Возвращаем индекс j, который теперь указывает на место, где стоит опорный элемент, после завершения разделения.

 **Функция randomized\_quicksort(A, l, r)**:

* Это основная рекурсивная функция быстрой сортировки.
* Если левый индекс меньше правого (то есть есть, что сортировать), выбираем случайный индекс k между l и r, чтобы обменять его значение с первым элементом. Это шаг рандомизации.
* После этого вызываем partition, который разделяет массив, и затем рекурсивно сортируем две полученные части массива:
  + Массив слева от опорного элемента.
  + Массив справа от опорного элемента.
* Функция возвращает отсортированный массив.

Результат работы кода на примерах из текста задачи:

Ввод:

Изображение выглядит как часы, снимок экрана, Шрифт, число

Автоматически созданное описание

Вывод:

Изображение выглядит как текст, часы, Шрифт, число

Автоматически созданное описание

Вывод по задаче: Я научилась реализовывать рандомизированную версию алгоритма быстрой сортировки.

## Задача №3. Сортировка пугалом

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Решение:

import utils  
import os  
  
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\_\_":  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task3', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = pugalo\_sort(data[1], data[0][0], data[0][1])  
 utils.print\_task\_data(3, 3, data, res)

Текстовое объяснение решения:

**Функция pugalo\_sort(A, n, k)**:

* **Параметры**:
  + A — список чисел, который нужно отсортировать.
  + n — размер массива.
  + k — шаг, с которым сравниваются элементы в процессе сортировки.
* **Алгоритм**:
  + Основная идея алгоритма заключается в том, чтобы пройти по массиву несколько раз, сравнивая элементы, которые находятся на расстоянии k друг от друга.
  + В каждом проходе алгоритм проверяет элементы на позиции i и i + k, и если элемент на позиции i больше, чем на позиции i + k, они меняются местами.
  + После каждого прохода проверяется флаг sorted\_flag, который помогает понять, произошли ли изменения в массиве. Если изменений не было, сортировка завершена.
  + Алгоритм продолжает работать до тех пор, пока весь массив не будет отсортирован (или пока не установится флаг sorted\_flag).
* **Возвращаемое значение**:
  + После завершения работы алгоритма, если массив действительно отсортирован, функция возвращает "ДА", иначе — "НЕТ".

Результат работы кода на примерах задачи:

Ввод:

Изображение выглядит как часы, снимок экрана, текст, Шрифт

Автоматически созданное описание

Вывод:

Изображение выглядит как снимок экрана, Шрифт, часы, дизайн

Автоматически созданное описание

Вывод по задаче: Я научилась реализовывать сортировку пугалом

## Задача №4. Точки и отрезки

Изображение выглядит как текст, Шрифт, снимок экрана, документ

Автоматически созданное описание

Решение:

import utils  
import os  
  
def interval\_count(data):  
 s, p = data[0][0], data[0][1]  
 intervals = [x for x in data[1:1 + s]]  
 points = data[-1]  
 result = {}  
 coordinates = []  
  
 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\_\_':  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task4', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = interval\_count(data)  
 utils.print\_task\_data(3, 4, data, res)

Текстовое объяснение решения:

**Функция interval\_count(data)**:

* **Параметры**:
  + data — входные данные, где первая строка содержит количество интервалов и точек, а вторая часть — сами интервалы и точки.
* **Процесс**:
  + Извлекаем количество интервалов (s) и точек (p).
  + Извлекаем интервалы и точки.
  + Создаём список координат (точек начала интервалов, концов интервалов и самих точек).
  + Для каждого интервала добавляем две координаты:
    - Начало интервала (помечаем как "L").
    - Конец интервала (помечаем как "R").
  + Для каждой точки добавляем её в список координат и создаём для неё запись в словаре result с начальным значением 0.
  + **Сортировка координат**:
    - Все координаты сортируются сначала по значению (по числовому порядку), а если значения равны — то по типу координаты ("L" -> "P" -> "R"). Это важно для правильной обработки, когда несколько объектов (начало интервала, точка или конец интервала) могут иметь одинаковое значение.
  + **Процесс подсчёта активных интервалов**:
    - Итерируем по отсортированным координатам:
      * Когда встречаем начало интервала ("L"), увеличиваем счётчик активных интервалов.
      * Когда встречаем конец интервала ("R"), уменьшаем счётчик.
      * Когда встречаем точку ("P"), записываем количество активных интервалов в результат для этой точки.
  + **Результат**:
    - Для каждой точки изначального списка точек в том же порядке возвращается количество интервалов, которые её содержат.

Результат работы кода на примерах задачи:

Ввод:

Изображение выглядит как снимок экрана, текст, число, часы

Автоматически созданное описание

Вывод:

Изображение выглядит как часы, снимок экрана, Шрифт, число

Автоматически созданное описание

Вывод по задаче: Научилась вычислять для каждой точки количество отрезков, содержащих эту точку.

## Задача №5. Индекс Хирша

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

Решение:  
import utils  
import os  
  
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\_\_':  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task5', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = hirsh\_index(data[0])  
 utils.print\_task\_data(3, 5, data, res)

Текстовое объяснение решения:

**Функция hirsh\_index(arr)**:

* **Входные данные**: массив чисел arr, где каждое число представляет собой количество цитирований для соответствующей публикации.
* **Алгоритм**:
  1. Сортируем массив arr в убывающем порядке. Это необходимо, чтобы сначала рассмотреть публикации с наибольшим числом цитирований.
  2. После сортировки массива:
     + Мы начинаем с h=0h = 0h=0 и проверяем, для какого максимального индекса iii выполняется условие, что число цитирований arr[i]arr[i]arr[i] хотя бы равно i+1i + 1i+1. Это гарантирует, что данная публикация имеет хотя бы i+1i + 1i+1 цитирований.
     + Мы увеличиваем hhh, если публикация удовлетворяет этому условию.
  3. Если публикация не удовлетворяет этому условию, то мы прерываем цикл, так как дальнейшие публикации (из-за сортировки) будут иметь меньшее число цитирований.
* **Возвращаемое значение**:
  1. Функция возвращает значение hhh, которое является индексом Хирша для переданных данных.

Результат работы кода на примерах задачи:

Ввод:

Изображение выглядит как текст, часы, снимок экрана, Шрифт

Автоматически созданное описание

Вывод:

Изображение выглядит как снимок экрана, черный, белый, дизайн

Автоматически созданное описание

Вывод по задаче: Научилась находить индекс Хирша

## Задача №6. Сортировка целых чисел

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Решение:

import utils  
import os  
from lab3.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\_\_':  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task6', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = sum\_of\_tenth(data[1], data[2])  
 utils.print\_task\_data(3, 6, data, res)

Текстовое объяснение решения:

**Функция sum\_of\_tenth(A, B)**:

* **Входные данные**:
  + A — первый массив.
  + B — второй массив.
* **Алгоритм**:
  + Сначала создаётся новый список CCC, в котором будут храниться произведения всех пар элементов aaa из массива AAA и bbb из массива BBB.
  + Для каждого bbb из массива BBB и каждого aaa из массива AAA вычисляется произведение a×ba \times ba×b, которое добавляется в список CCC.
  + После того как все произведения вычислены, массив CCC сортируется с использованием алгоритма randomized\_quicksort.
  + Затем, с помощью среза, суммируются элементы массива CCC через 10 индексов, начиная с индекса 0 (то есть элементы с индексами 0,10,20,…0, 10, 20, \dots0,10,20,…).
  + Результат возвращается.

Результат работы кода на примерах задачи:

Ввод:

Изображение выглядит как Шрифт, снимок экрана, текст, число

Автоматически созданное описание

Вывод:

Изображение выглядит как снимок экрана, часы, текст, Шрифт

Автоматически созданное описание

Вывод по задаче: Я научилась осуществлять сортировку целых чисел

## Задача №8. К ближайших точек к началу координат

Изображение выглядит как текст, Шрифт, снимок экрана, алгебра

Автоматически созданное описание

Решение:

import utils  
import os  
from lab3.task1.src.task2 import randomized\_quicksort  
  
def 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\_\_':  
 project\_root = os.path.abspath(os.path.join(os.path.dirname(\_\_file\_\_), '../../..'))  
 input\_file\_path = os.path.join(project\_root, 'lab3', 'task8', 'txtf', 'input.txt')  
 data = utils.read\_input(input\_file\_path)  
 res = closest\_points(data[1:], data[0][1])  
 utils.print\_task\_data(3, 8, data, res)

Текстовое объяснение решения:

**Функция closest\_points(arr, k)**:

* **Входные данные**:
  + arr — массив точек, где каждая точка представлена списком координат [x,y][x, y][x,y].
  + k — количество ближайших точек, которые нужно найти.
* **Алгоритм**:
  + Для каждой точки (x,y)(x, y)(x,y) из массива arr вычисляется её расстояние от начала координат с использованием формулы Пифагора: distance=x2+y2\text{distance} = \sqrt{x^2 + y^2}distance=x2+y2​
  + Все точки сохраняются в массив points, где каждый элемент представляет собой пару:
    - расстояние (вместо самого числа используется округленное значение),
    - сами координаты точки.
  + Массив points сортируется с использованием алгоритма randomized\_quicksort (по расстоянию).
  + Возвращаются первые kkk точек из отсортированного массива, что и будет результатом.

Результат работы кода на примерах задачи:

Ввод:

Изображение выглядит как часы, снимок экрана, число, дизайн

Автоматически созданное описание

Вывод:

Изображение выглядит как часы, текст, Шрифт, число

Автоматически созданное описание

Вывод по задаче: Я научилась находить К ближайших точек к началу координат среди данных n точек.

# Вывод

В ходе лабораторной работы я изучила алгоритмы, в которых используется метод разделяй и властвуй, а также разные методы сортировки: быстрая сортировка и сортировки с линейной сложностью.