САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

по курсу «Алгоритмы и структуры данных»

Вариант 1

Выполнил:

Азаренков Георгий Денисович

К34421

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

[Содержание отчета 2](#_Toc179014282)

[Комментарий к общему коду 3](#_Toc179014283)

[Задача 1 5](#_Toc179014284)

[Текст задачи 5](#_Toc179014285)

[Код решения 6](#_Toc179014286)

[Результаты выполнения 6](#_Toc179014287)

[Задача 4 7](#_Toc179014288)

[Текст задачи 7](#_Toc179014289)

[Код решения 7](#_Toc179014290)

[Результаты выполнения 7](#_Toc179014291)

[Задача 7 9](#_Toc179014292)

[Текст задачи 9](#_Toc179014293)

[Код решения 9](#_Toc179014294)

[Результаты выполнения 9](#_Toc179014295)

[Вывод 11](#_Toc179014296)

# Комментарий к общему коду

При использовании всех задач я использую код из файла “common.py”, который находится в корне репозитория.

from typing import List, Callable  
  
  
def solve(solver: Callable[[List[str]], List[str]]) -> None:  
 with open("../../input.txt", "r") as file:  
 lines = file.readlines()  
 input\_lines = list(map(lambda x: x.strip(), lines))  
  
 output\_lines = solver(input\_lines)  
  
 with open("../../output.txt", "w") as file:  
 for line in output\_lines:  
 file.write(f"{line}\n")

Использование функции “solve” позволило мне избавиться от необходимости раз за разом имплементировать логику чтения и сохранения данных.

# Задача 1

## Текст задачи

A paper with text and numbers

Description automatically generated

## Код решения

import random  
from typing import List  
  
from common import solve  
  
  
def partition(arr: List[int], low: int, high: int) -> int:  
 *"""  
 Функция разделяет массив и возвращает индекс опорного элемента.  
 """* pivot = arr[high]  
 i = low - 1  
  
 *# Проходим по элементам и переставляем их относительно опорного* for j in range(low, high):  
 if arr[j] < pivot:  
 i += 1  
 arr[i], arr[j] = arr[j], arr[i]  
  
 *# Перемещаем опорный элемент на правильную позицию* arr[i + 1], arr[high] = arr[high], arr[i + 1]  
  
 return i + 1  
  
  
def randomized\_quicksort(arr: List[int], low: int, high: int) -> None:  
 *"""  
 Рекурсивная реализация рандомизированной быстрой сортировки.  
 """* if low < high:  
 *# Выбираем случайный опорный элемент* pivot\_index = random.randint(low, high)  
 arr[pivot\_index], arr[high] = arr[high], arr[pivot\_index]  
  
 *# Получаем позицию опорного элемента после разделения* pi = partition(arr, low, high)  
  
 *# Рекурсивно сортируем левую часть* randomized\_quicksort(arr, low, pi - 1)  
  
 *# Рекурсивно сортируем правую часть* randomized\_quicksort(arr, pi + 1, high)  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Считывает массив чисел из входных строк, сортирует его с помощью  
 рандомизированной быстрой сортировки и возвращает отсортированный массив.  
 """* n = int(input\_lines[0])  
  
 *# Преобразуем строку чисел в список целых чисел* arr = list(map(int, input\_lines[1].split()))  
  
 *# Вызываем функцию быстрой сортировки* randomized\_quicksort(arr, 0, n - 1)  
  
 *# Преобразуем отсортированный список обратно в строку* sorted\_arr\_str = " ".join(map(str, arr))  
  
 return [sorted\_arr\_str]  
  
  
solve(get\_solution)

addition

import random  
from typing import List  
  
from sem1\_lab3\_task1 import randomized\_quicksort  
  
  
def generate\_random\_array(min\_size: int = 10, max\_size: int = 1000) -> List[int]:  
 size = random.randint(min\_size, max\_size)  
 arr = random.sample(range(-(10\*\*9), 10\*\*9), size)  
 return arr  
  
  
def test\_randomized\_quicksort(iterations: int = 100) -> None:  
 for i in range(1, iterations + 1):  
 arr = generate\_random\_array()  
  
 arr\_copy = arr.copy()  
  
 randomized\_quicksort(arr, 0, len(arr) - 1)  
  
 expected = sorted(arr\_copy)  
  
 if arr != expected:  
 print(f"Тест {i} не прошел.")  
 print(f"Ожидалось: {expected}")  
 print(f"Получено: {arr}")  
 return  
  
 print("Все тесты прошли успешно!")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 test\_randomized\_quicksort()

## Результаты выполнения

input.txt

A number on a white background

Description automatically generated

output.txt

A number on a white background

Description automatically generated

# Задача 4

## Текст задачи

A screenshot of a computer

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Рассчитывает количество отрезков, содержащих каждую из заданных точек.  
 """  
  
 # Чтение первой строки и разбор количества отрезков и точек* s, p = map(int, input\_lines[0].split())  
  
 *# Инициализация списков для начала и конца отрезков* starts = []  
 ends = []  
  
 *# Чтение отрезков* for i in range(1, s + 1):  
 a, b = map(int, input\_lines[i].split())  
 starts.append(a)  
 ends.append(b)  
  
 *# Чтение точек* points = list(map(int, input\_lines[s + 1].split()))  
  
 *# Сортировка начала и конца отрезков* starts.sort()  
 ends.sort()  
  
 *# Инициализация списка для результатов* result = []  
  
 *# Функция для подсчета количества элементов <= x с использованием бинарного поиска* def count\_leq(arr: List[int], x: int) -> int:  
 left, right = 0, len(arr)  
  
 while left < right:  
 mid = (left + right) // 2  
 if arr[mid] <= x:  
 left = mid + 1  
 else:  
 right = mid  
 return left  
  
 *# Подсчет для каждой точки количества содержащих отрезков* for x in points:  
 *# Количество отрезков с началом <= x* cnt\_start = count\_leq(starts, x)  
  
 *# Количество отрезков с концом < x* cnt\_end = count\_leq(ends, x - 1)  
  
 *# Количество отрезков, содержащих точку x* count = cnt\_start - cnt\_end  
  
 *# Добавление результата в список* result.append(str(count))  
  
 *# Возвращение результата в виде списка строк* return [" ".join(result)]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A close up of numbers

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

Пример 2

input.txt

A number with numbers on it

Description automatically generated with medium confidence

output.txt

A screenshot of a computer

Description automatically generated

Пример 3

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a calendar

Description automatically generated

# Задача 7

## Текст задачи

A screenshot of a computer

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Определяет порядок строк после k фаз цифровой сортировки.  
 """  
  
 # Чтение первой строки для получения n, m и k* first\_line = input\_lines[0].strip().split()  
 n = int(first\_line[0]) *# Количество строк* m = int(first\_line[1]) *# Длина каждой строки* k = int(first\_line[2]) *# Количество фаз сортировки  
  
 # Инициализация списка строк* strings = ["" for \_ in range(n)]  
  
 *# Формирование строк из вертикального ввода* for i in range(m):  
 current\_line = input\_lines[i + 1].strip()  
  
 for j in range(n):  
 strings[j] += current\_line[j]  
  
 *# Индексы строк для сортировки* indices = list(range(1, n + 1))  
  
 *# Выполнение k фаз цифровой сортировки* for phase in range(1, k + 1):  
 *# Определение позиции символа для текущей фазы (с конца)* pos = m - phase  
  
 *# Сортировка индексов на основе символа на позиции pos* indices.sort(key=lambda x: strings[x - 1][pos])  
  
 *# Преобразование списка индексов в строку с пробелами* result: str = " ".join(map(str, indices))  
  
 *# Возврат результата в виде списка строк* return [result]  
  
  
solve(get\_solution)

## Результаты выполнения

Пример 1

input.txt

A screenshot of a computer

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

Пример 2

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Пример 3

A screenshot of a computer

Description automatically generated

A close up of a number

Description automatically generated

# Вывод

В ходе данной лабораторной работы я научился решать задачи по теме. Написанные программы были протестированы с измерением затрат времени и оперативной памяти. Все программы работаю корректно и укладываются в установленные ограничения.