САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

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](#_Toc179013569)

[Комментарий к общему коду 3](#_Toc179013570)

[Задача 1 4](#_Toc179013571)

[Текст задачи 4](#_Toc179013572)

[Код решения 5](#_Toc179013573)

[Результат выполнения 6](#_Toc179013574)

[Задача 2 7](#_Toc179013575)

[Текст задачи 7](#_Toc179013576)

[Код решения 9](#_Toc179013577)

[Результаты выполнения 10](#_Toc179013578)

[Задача 3 12](#_Toc179013579)

[Текст задачи 12](#_Toc179013580)

[Код решения 12](#_Toc179013581)

[Результаты выполнения 14](#_Toc179013582)

[Вывод 15](#_Toc179013583)

# 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 on it

Description automatically generated

## Код решения

from common import solve  
  
  
def get\_solution(input\_lines: list[str]) -> list[str]:  
 *"""  
 Сортирует заданный массив чисел с помощью сортировки слиянием без использования сигнальных значений.  
 """  
  
 # Преобразуем вторую строку в список целых чисел* arr = list(map(int, input\_lines[1].split()))  
  
 *# Выполняем сортировку слиянием* merge\_sort(arr)  
  
 *# Преобразуем отсортированный массив в строку с пробелами между числами* output\_line = " ".join(map(str, arr))  
  
 *# Возвращаем результат в виде списка строк* return [output\_line]  
  
  
def merge\_sort(arr: list[int]) -> None:  
 *"""Рекурсивно разделяет массив и сортирует его части с помощью функции merge."""  
  
 # Если массив состоит из одного элемента, то он уже отсортирован* if len(arr) > 1:  
 *# Находим середину массива* mid = len(arr) // 2  
  
 *# Делим массив на две половины* left\_half = arr[:mid]  
 right\_half = arr[mid:]  
  
 *# Рекурсивно сортируем каждую половину* merge\_sort(left\_half)  
 merge\_sort(right\_half)  
  
 *# Сливаем отсортированные половины* merge(arr, left\_half, right\_half)  
  
  
def merge(arr: list[int], left: list[int], right: list[int]) -> None:  
 *"""Сливает два отсортированных массива в один без использования сигнальных значений."""  
  
 # Инициализируем индексы для левого, правого и основного массива* i = j = k = 0  
  
 *# Пока в обоих массивах есть элементы* while i < len(left) and j < len(right):  
 *# Сравниваем элементы и добавляем меньший в основной массив* if left[i] <= right[j]:  
 arr[k] = left[i]  
 i += 1  
 else:  
 arr[k] = right[j]  
 j += 1  
 k += 1  
  
 *# Если в левом массиве остались элементы, добавляем их в основной массив* while i < len(left):  
 arr[k] = left[i]  
 i += 1  
 k += 1  
  
 *# Если в правом массиве остались элементы, добавляем их в основной массив* while j < len(right):  
 arr[k] = right[j]  
 j += 1  
 k += 1  
  
  
solve(get\_solution)

## Результат выполнения

input.txt

A close up of numbers

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 2

## Текст задачи

A screenshot of a paper

Description automatically generated

## Код решения

from typing import List  
  
from common import solve  
  
  
def get\_solution(input\_lines: List[str]) -> List[str]:  
 *"""  
 Сортирует массив чисел с помощью сортировки слиянием и выводит информацию о каждом слиянии и итоговый отсортированный массив.  
 """  
  
 # Считываем количество элементов из первой строки* n = int(input\_lines[0])  
  
 *# Преобразуем вторую строку в список целых чисел* arr = list(map(int, input\_lines[1].split()))  
  
 *# Инициализируем список для хранения строк вывода* output: List[str] = []  
  
 *# Запускаем сортировку слиянием* merge\_sort(arr, 0, n, output)  
  
 *# Добавляем отсортированный массив в список вывода* output.append(" ".join(map(str, arr)))  
  
 return output  
  
  
def merge\_sort(arr: List[int], l: int, r: int, output: List[str]) -> None:  
 *"""  
 Рекурсивная функция для сортировки массива с помощью сортировки слиянием.  
 """  
  
 # Проверяем, содержит ли подмассив более одного элемента* if r - l > 1:  
 *# Находим середину подмассива* m = (l + r) // 2  
  
 *# Рекурсивно сортируем левую половину* merge\_sort(arr, l, m, output)  
  
 *# Рекурсивно сортируем правую половину* merge\_sort(arr, m, r, output)  
  
 *# Сливаем отсортированные половины* merge(arr, l, m, r, output)  
  
  
def merge(arr: List[int], l: int, m: int, r: int, output: List[str]) -> None:  
 *"""  
 Сливает два отсортированных подмассива в один.  
 """  
  
 # Создаем копии левого и правого подмассивов* left = arr[l:m]  
 right = arr[m:r]  
  
 *# Инициализируем индексы для итерации по подмассивам* i = j = 0  
  
 *# Индекс для вставки в основной массив* k = l  
  
 *# Пока есть элементы в обоих подмассивах* while i < len(left) and j < len(right):  
 *# Выбираем наименьший элемент и помещаем его в основной массив* if left[i] <= right[j]:  
 arr[k] = left[i]  
 i += 1  
 else:  
 arr[k] = right[j]  
 j += 1  
  
 k += 1  
  
 *# Копируем оставшиеся элементы из левого подмассива* while i < len(left):  
 arr[k] = left[i]  
 i += 1  
 k += 1  
  
 *# Копируем оставшиеся элементы из правого подмассива* while j < len(right):  
 arr[k] = right[j]  
 j += 1  
 k += 1  
  
 *# Подготавливаем данные для вывода информации о слиянии* If = l + 1 *# Начальный индекс области слияния (1-начало)* Il = r *# Конечный индекс области слияния* Vf = arr[l] *# Первый элемент после слияния* Vl = arr[r - 1] *# Последний элемент после слияния  
  
 # Добавляем информацию о слиянии в список вывода* output.append(f"{If} {Il} {Vf} {Vl}")  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A number on a white background

Description automatically generated

output.txt

A screenshot of a computer

Description automatically generated

# Задача 3

## 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]:  
 *"""  
 Подсчитывает количество инверсий в заданном массиве чисел.  
 """  
  
 # Парсим количество элементов в массиве* n = int(input\_lines[0])  
  
 *# Парсим сам массив чисел* array = list(map(int, input\_lines[1].split()))  
  
 *# Вызываем функцию для подсчета инверсий* inversions = count\_inversions(array)  
  
 *# Возвращаем результат в виде списка строк* return [str(inversions)]  
  
  
def count\_inversions(array: List[int]) -> int:  
 *"""  
 Использует сортировку слиянием для эффективного подсчета инверсий.  
 """  
  
 # Создаем вспомогательный массив для слияния* temp\_array = [0] \* len(array)  
  
 *# Запускаем рекурсивную функцию сортировки слиянием* return merge\_sort(array, temp\_array, 0, len(array) - 1)  
  
  
def merge\_sort(array: List[int], temp\_array: List[int], left: int, right: int) -> int:  
 *# Инициализируем счетчик инверсий* inv\_count = 0  
  
 *# Базовый случай рекурсии* if left < right:  
 *# Находим середину массива* mid = (left + right) // 2  
  
 *# Рекурсивно сортируем левую половину и подсчитываем инверсии* inv\_count += merge\_sort(array, temp\_array, left, mid)  
  
 *# Рекурсивно сортируем правую половину и подсчитываем инверсии* inv\_count += merge\_sort(array, temp\_array, mid + 1, right)  
  
 *# Сливаем две половины и подсчитываем инверсии при слиянии* inv\_count += merge(array, temp\_array, left, mid, right)  
  
 return inv\_count  
  
  
def merge(array: List[int], temp\_array: List[int], left: int, mid: int, right: int) -> int:  
 *# Инициализируем индексы для левой и правой половин* i = left *# Начало левой половины* j = mid + 1 *# Начало правой половины* k = left *# Начало вспомогательного массива* inv\_count = 0 *# Счетчик инверсий  
  
 # Сливаем левую и правую половины* while i <= mid and j <= right:  
 if array[i] <= array[j]:  
 temp\_array[k] = array[i]  
 i += 1  
 else:  
 temp\_array[k] = array[j]  
 *# Элементы оставшиеся в левой половине являются инверсиями* inv\_count += mid - i + 1  
 j += 1  
 k += 1  
  
 *# Копируем оставшиеся элементы левой половины* while i <= mid:  
 temp\_array[k] = array[i]  
 i += 1  
 k += 1  
  
 *# Копируем оставшиеся элементы правой половины* while j <= right:  
 temp\_array[k] = array[j]  
 j += 1  
 k += 1  
  
 *# Копируем отсортированный сегмент обратно в оригинальный массив* for idx in range(left, right + 1):  
 array[idx] = temp\_array[idx]  
  
 return inv\_count  
  
  
solve(get\_solution)

## Результаты выполнения

input.txt

A number on a white background

Description automatically generated

output.txt

A number on a white background

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.