МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студентка гр.1381 _____ Рымарь М.И.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург

2022

Цель работы.

Написать программу, реализующую сортировку слиянием.

Задание.

На вход программе подаются квадратные матрицы чисел. Напишите программу, которая сортирует матрицы по возрастанию суммы чисел на главной диагонали с использованием алгоритма сортировки слиянием.

Формат входа

Первая строка содержит натуральное число *n* - количество матриц. Далее на вход подаются *n* матриц, каждая из которых описана в формате: сначала отдельной строкой число *mi* - размерность *i*-й по счету матрицы. После *m* строк по *m* чисел в каждой строке - значения элементов матрицы.

Формат выхода

Порядковые номера тех матриц, которые участвуют в слиянии на очередной итерации алгоритма. Вывод с новой строки для каждой итерации.

Массив, в котором содержатся порядковые номера матриц, отсортированных по возрастанию суммы элементов на диагонали. Порядковый номер матрицы — это её номер по счету, в котором она была подана на вход программе, нумерация начинается с нуля.

Выполнение работы.

Для удобства чтения программы, код был разделён на несколько функций: *func_read(), func_sum(), func_merge()* и основная функция *main()*. Первая функция *func_read()* отвечает за считывание данных из консоли, она возвращает количество введённых матриц и массив матрицы. Вторая функция *func_sum()* вычисляет сумму элементов на главной диагонали матрицы, на вход подаётся текущая матрица, возвращается целое число – сумма элементов. Третья функция *func_merge()* является функцией сортировки слиянием. Сначала массив матриц разбивается на две части и рекурсивно сортируется, когда в рассматриваемом массиве остаётся один элемент, то он отсортирован. Далее к двум частям массива

применяется процедура слияния, которая и получает по двум отсортированным частям отсортированным частям массив. Функция получает на вход кортеж-пару, состоящую из индекса матрицы и суммы элементов на главной диагонали, а также пустой буфер для хранения промежуточных значений номеров матриц, участвующих в слиянии на очередной итерации. Функция *main()* отвечает за связь всех функций, перечисленных выше. В цикле, проходящему по длине буфера, на каждой итерации выводим значения функции сортировки. Далее вне цикла выводим результат работы программы.

Тестирование.

Для тестирования программы был использован фреймворк модульного тестирования *unittest*. Тесты рассматривают несколько вариантов ввода матриц. Тестируется основная функция сортировки слиянием *func_merge*. Сначала проверяются «граничные значения»: одна матрица с рангом один и одна матрица с рангом ноль. Далее рассматриваются более осмысленные тесты: 2, 3 и 4 матрицы с разными рангами.

Результаты тестирования представлены на рисунке 1. Файл с тестированием *test.py* представлен в приложении А.

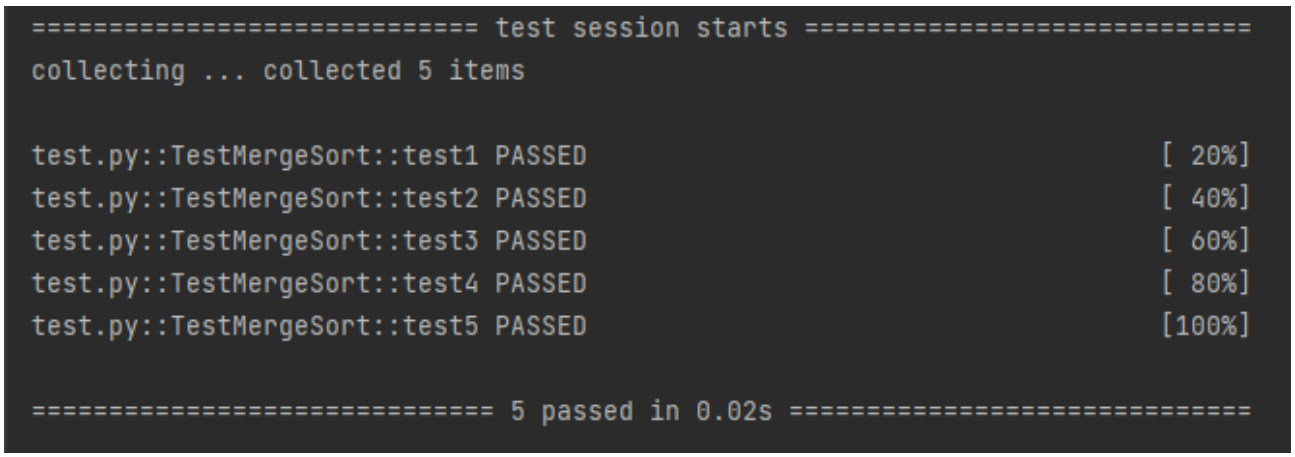

Рисунок 1 – Результаты тестирования

Выводы.

В ходе выполнения лабораторной работы были изучены виды сортировок. Для решения поставленной задачи была реализована сортировка слиянием.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММ

Название файла: *src.py*

```
def func_read():
    matr_num = int(input())
    matrixes = []
    for i in range(matr_num):
        matr_size = int(input())
        matrix = []
        for k in range(matr_size):
            matr_string = list(map(int, input().split()))
            matrix.append(matr_string)
        matrixes.append(matrix)
    return matr_num, matrixes


def func_sum(matrix):
    sum = 0
    for i in range(len(matrix)):
        sum += matrix[i][i]
    return sum


def func_merge(list_matr, buf):
    if len(list_matr) == 1:
        return list_matr
    middle = len(list_matr) // 2
    left, right = list_matr[:middle], list_matr[middle:]
    func_merge(left, buf)
    func_merge(right, buf)
    index_left = index_right = index = 0
    result = [0] * (len(left) + len(right))
    while index_left < len(left) and index_right < len(right):
        if left[index_left][1] <= right[index_right][1]:
            result[index] = left[index_left]
            index_left += 1
        else:
            result[index] = right[index_right]
            index_right += 1
        index += 1
    while index_left < len(left):
        result[index] = left[index_left]
        index_left += 1
        index += 1
    while index_right < len(right):
        result[index] = right[index_right]
        index_right += 1
        index += 1
    for i in range(len(list_matr)):
        list_matr[i] = result[i]
    buf.append(list_matr)
    return list_matr
```

```
def main():
    num, matrixes = func_read()
    list_matr = list()
    buf = list()
    for i in range(num):
        list_matr.append((i, func_sum(matrixes[i])))
    res = func_merge(list_matr, buf)
    for n in range(len(buf)):
        tmp = ' '.join(map(lambda x: str(x[0]), sorted(buf[n],
key=lambda pair:pair[1])))
        print(tmp)
    result = ' '.join(map(lambda x: str(x[0]), res))
    print(result)


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

Название файла: *test.py*

```
import unittest
from src import func_merge

class TestMergeSort(unittest.TestCase):

    def test1(self):
        self.assertEqual(func_merge([(0, 1)], list()), [(0, 1)])

    def test2(self):
        self.assertEqual(func_merge([(0, 0)], list()), [(0, 0)])

    def test3(self):
        self.assertEqual(func_merge([(0, 32), (1, 11), (2, 3)],
list()), [(2, 3), (1, 11), (0, 32)])

    def test4(self):
        self.assertEqual(func_merge([(0, 5), (1, 1)], list()), [(1,
1), (0, 5)])

    def test5(self):
        self.assertEqual(func_merge([(0, 5), (1, 6), (2, -4), (3,
12)], list()), [(2, -4), (0, 5), (1, 6), (3, 12)])
```