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

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Алгоритмы и структуры данных»

Тема: Сортировки

Студент гр. 1304 Шаврин А.П.

Преподаватель Глазунов С.А

Санкт-Петербург

2022

Цель работы.

Изучить основные сортировки, их принцип работы и особенности с помощью языка программирования *Python*. Реализовать сортировки 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.

Пример

Вход:

```
3
2
1 2
1 31
3
1 1 1
1 11 1
1 1 -1
5
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
```

Выход:

```
2 1

2 1 0
```

2 1 0

Объяснение:

n = 3

m0 = 2
Первая матрица (порядковый номер 0):
1 2
1 31

m1 = 3
Вторая матрица (порядковый номер 1):
1 1 1
1 11 1
1 1 -1

m2 = 5
Третья матрица (порядковый номер 2):
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
1 2 0 1 -1
Сумма элементов диагонали матрицы с порядковым номером 0 = 32
Сумма элементов диагонали матрицы с порядковым номером 1 = 11
Сумма элементов диагонали матрицы с порядковым номером 2 = 3

Для упрощения, можем свести задачу сортировки массива матриц к задаче сортировки массива чисел, где каждое число определяет сумму элементов диагонали матрицы. В итоге мы имеем массив элементов с порядковыми номерами [0, 1, 2] и суммой элементов на главной диагонали 32, 11, 3 для нулевого, первого и второго по порядку элементов соответственно. На первой итерации сортировки исходный массив делится на два подмассива:

[0]

и

[1, 2]

Далее происходит деление второго массива на два массива по одному элементу:

[1]

[2]

После происходит слияние. Массивы [1] и [2] сливаются в один:

[2, 1]

Порядок элементов такой, поскольку сумма элементов диагонали матрицы с порядковым номером 2 меньше, чем сумма элементов диагонали матрицы с порядковым номером 1.

В этот момент ваша программа должна сделать первый вывод. Вывод содержит только порядковые номера матриц, разделенные пробелом. Далее массив [2, 1] сливается с массивом [0]:

[2, 1, 0]

И это является вторым выводом.
Массив отсортирован, теперь нужно вывести окончательный результат сортировки:

2 1 0.

Поэтому, правильный вывод задачи выглядит так:

2 1

2 1 0

2 1 0

При делении массива нечетной длины считаем, что первая часть после деления меньшая.

Примечание: вы можете использовать библиотеку numpy, но это не является обязательным.

Первой строкой добавьте #python или #c++, чтобы проверяющая система знала, каким языком вы пользуетесь.

Выполнение работы.

1. Изначально была реализована основная логика работы программы.

При запуске программы происходит считывание числа *n*, отвечабщего за количество матриц, подаваемых на вход программе.

Затем создан список *sums_diagonals,* который имитирует массив в данной задаче и будет хранить кортежи типа: (номер матрицы, сумма главной диагонали).

После реализован цикл *for*, считывающий *n* матриц.

Далее реализован цикл *for,* считывающий строки n-ой матрицы и прибавяющий к переменной *sum_diagonal* числа n-ой матрицы с главной диагонали. После завершения цикла в *sums_diagonals* добавляется кортеж (*matrix_number, sum_diagonal*).

После считывания всех матриц и заполнения массива *sums_diagonals* следует вызов функции *MergeSorting*, в которую передается наш массив *sums_diagonals.*

2. Затем реализована функция *MergeSorting*, принимающая на вход массив *array.*

Сначала происходит базовая проверка на пустой или массив из одного элемента.

После определяется середина массива, в случае нечетного количества элементов середина будет округлена в меньшую сторону (при помощи целочисленного деления).

Далее массив делится пополам и для каждой половины рекурсивно вызывается функция *MergeSorting.* Так происходит до тех пор, пока в полученных списках не будет по одному элементу.

После происходит рекурсивное слияние получившихся списков в нужном порядке в результирующий массив.

В конечном итоге мы получаем исходный отсортированный массив.

Разработанный программный код см. в приложении А.

Тестирование.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	1 2 1 2 2 3	0	Результат верный
2.	3 2 -62 -8 -1 97 3 -98 -84 28 32 -85 -33 96 -68 -99 2 15 81 67 68	1 2 1 0 2 1 0 2	Результат верный
3.	0		Результат верный

Выводы.

Были изучены основные виды сортировок, их принцип работы и особенности с помощью языка программирования *Python*. Реализована сортировка слиянием.

Проведена проверка кода с помощью системы *pytest*.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла *merge_sorting.py*

```
#python
def MergeSorting(array):
    if len(array) <= 1:
        return array

    middle = len(array) // 2
    left, right = array[:middle], array[middle:]

    MergeSorting(left)
    MergeSorting(right)

    result_array = [0] * (len(left) + len(right))
    index_left = index_right = index = 0

    while index_left < len(left) and index_right < len(right):
        if left[index_left][1] <= right[index_right][1]:
            result_array[index] = left[index_left]
            index_left += 1
        else:
            result_array[index] = right[index_right]
            index_right += 1
        index += 1

    while index_left < len(left):
        result_array[index] = left[index_left]
        index_left += 1
        index += 1
```

```
    while index_right < len(right):
        result_array[index] = right[index_right]
        index_right += 1
        index += 1

    for i in range(len(array)):
        array[i] = result_array[i]

    for i in range(len(array)):
       print(array[i][0], end = ' ')
    print()

    return array

if __name__ == "__main__":
    n = int(input())
    sums_diagonals = []
    for matrix_number in range(n):
        sum_diagonal = 0
        matrix_size = int(input())
        for i in range(matrix_size):
            line = list(map(int, input().split(' ')))
            sum_diagonal += line[i]
        sums_diagonals.append((matrix_number, sum_diagonal))

    MergeSorting(sums_diagonals)
    for i in range(n):
        print(sums_diagonals[i][0], end = ' ')
```

Название файла *pytests.py*

```
from merge_sorting import MergeSorting

def test_from_emoevm():
    sums = [(0,32), (1, 11), (2, 3)]
    assert MergeSorting(sums) == [(2, 3), (1, 11), (0, 32)]

def test_without_matrix():
    sums = []
    assert MergeSorting(sums) == []

def test_with_one_matrix():
    sums = [(0, 1)]
    assert MergeSorting(sums) == [(0, 1)]

def test_with_equal_sum_of_diagonal():
    sums = [(0, 6), (1, 10), (2, 10)]
    assert MergeSorting(sums) == [(0, 6), (1, 10), (2, 10)]

def test_with_equal_sums_of_diagonals():
    sums = [(0, 5), (1, 5), (2, 5)]
    assert MergeSorting(sums) == [(0, 5), (1, 5), (2, 5)]

def test_with_zero_summ_diagonal():
    sums = [(0, 23), (1, 0), (2, 4)]
    assert MergeSorting(sums) == [(1, 0), (2, 4), (0, 23)]

def test_with_the_negative_sum_of_the_diagonal():
    sums = [(0, 5), (1, 34), (2, -34)]
    assert MergeSorting(sums) == [(2, -34), (0, 5), (1, 34)]
```

```
def test_with_the_negative_sums_of_the_diagonals():
    sums = [(0, -1), (1, -3), (2, -123)]
    assert MergeSorting(sums) == [(2, -123), (1, -3), (0, -1)]
```