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: Сортировка слиянием.

Студент гр. 1303 _____ Кузнецов Н.А.

Преподаватель _____ Иванов Д.В.

Санкт-Петербург
2022

Цель работы.

Научиться работать с алгоритмом 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.

Выполнение работы

Class Matrix:

Содержит поля, хранящие в себе сумму элементов главной диагонали - *data* и индекс (порядковый номер ввода) - *ind*.

Scan_matrix():

Функция считывающая матрицы и потока ввода. Создает массив элементов типа экземпляр класса Matrix и возвращает его.

Merge(arr):

Функция принимает в качестве аргумента список элементов типа экземпляр класса *Matrix* - *arr*. Функция рекурсивная. Базовый случай: если

длина списка равна 1, то функция завершается. Далее список делится на две части, и для каждой из них рекурсивно вызывается функция *merge(arr)*. Затем создается список *result*, в котором будет находится результат сортировки. Производится обход по каждой части, с добавлением элементов в *result* в порядке возрастания и, если одна часть (левая или правая) оказалась больше, чем другая, то в *result* помещается остаток от большей части. *Result* перемещается в arr для дальнейшего взаимодействия. На каждой итерации в поток вывода выводятся индексы участвующих в текущей сортировке элементов.

Main():

Вызывает функции *scan_matrix()* и *merge(arr)*. Результат сортировки записывается в пустой массив, далее этот массив конвертируется в строку и выводится.

Разработанный программный код находится в приложении А.

Тестирование.

Для проверки работы программы был разработан код тестовой программы.

Код файла с тестами находится в приложении Б.

Результаты тестирования представлены в табл. 1.

Таблица 1 – Результаты тестирования

№ п/п	Входные данные	Выходные данные	Комментарии
1.	3 2 1 2 1 31 3 1 1 1 1 11 1 1 1 -1 5 1 2 0 1 -1	2 1 2 1 0 2 1 0	Верно.

	1 2 0 1 -1 1 2 0 1 -1 1 2 0 1 -1 1 2 0 1 -1		

Выводы.

Изучен алгоритм сортировки слиянием.

Был разработан программный код, позволяющий отсортировать массив матриц по сумме элементов на главной диагонали.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Название файла: matrix.py

```
class Matrix:
    def __init__(self, ind, data):
        self.ind = ind
        self.data = data
```

Название файла: scan_matrix.py

```
from matrix import Matrix


def scan_matrix():
    matrix_kol = int(input())
    arr = []
    for i in range(matrix_kol):
        cur_kol = int(input())
        cur_sum = 0
        for j in range(cur_kol):
            cur_str = list(map(int, input().split()))
            cur_sum += cur_str[j]
        arr.append(Matrix(i, cur_sum))
    return arr
```

Название файла: merge.py

```
from matrix import Matrix


def merge(arr):
    if len(arr) == 1:
        return
    middle = len(arr) // 2
    left, right = arr[:middle], arr[middle:]
    merge(left)
    merge(right)
    index_left = index_right = index = 0
    result = [Matrix(0, 0)] * (len(left) + len(right))
    while index_left < len(left) and index_right < len(right):
```

```
    if left[index_left].data <= right[index_right].data:
        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

output = []
for i in range(len(arr)):
    arr[i] = result[i]
    output.append(str(result[i].ind))
print(' '.join(output))
return arr
```

Название файла: main.py

```
from modules.scan_matrix import scan_matrix
from modules.merge import merge


def main():
    arr = scan_matrix()
    m_arr = merge(arr)
    final_output = []
    for i in m_arr:
        final_output.append(str(i.ind))
    print(' '.join(final_output))


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

ПРИЛОЖЕНИЕ Б
ФАЙЛ ТЕСТИРОВАНИЯ ПРОГРАММЫ

Название файла: test.py

```
from modules.merge import merge
from modules.matrix import Matrix


def test1():
    matrix0 = Matrix(-245, 0)
    matrix1 = Matrix(228, 1)
    matrix2 = Matrix(147, 2)
    matrix3 = Matrix(-99, 3)
    matrix4 = Matrix(-113, 4)
    matrix_list = [matrix0, matrix1, matrix2, matrix3, matrix4]
    matrix_result = [matrix0, matrix4, matrix3, matrix2, matrix1]
    assert merge(matrix_list) == matrix_result


def test2():
    matrix0 = Matrix(69, 0)
    matrix_list = [matrix0]
    matrix_result = [matrix0]
    assert merge(matrix_list) == matrix_result


def test3():
    matrix0 = Matrix(-87, 0)
    matrix1 = Matrix(93, 1)
    matrix2 = Matrix(111, 2)
    matrix_list = [matrix0, matrix1, matrix2]
    matrix_result = [matrix0, matrix1, matrix2]
    assert merge(matrix_list) == matrix_result
```