# 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: Сортировка слиянием

|  |  |  |
| --- | --- | --- |
| Студент гр. 1384 |  | Соломин Д.А. |
| Преподаватель |  | Иванов Д.В. |

Санкт-Петербург 2022

# Цель работы.

Разработка алгоритма сортировки слиянием, тестирование и поверка алгоритма пошагово

# Ход выполнения.

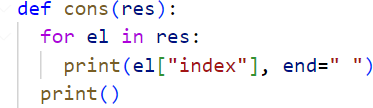
1. Для выполнения работы будем использовать язык *python,* для получения данных из ввода воспользуемся несколькими циклом, инкапсулированными в отдельную функцию.

Рисунок 1. (Вывод в консоль результата)

1. Для получения из консоли создадим 2 цикла, которые будут брать высоту

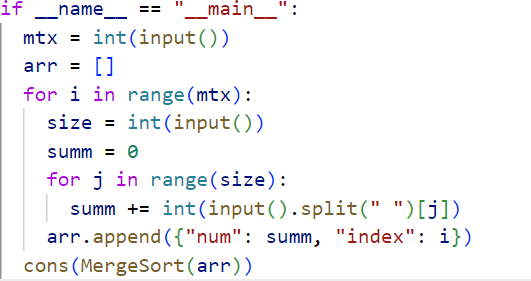
матрицы и вычислять сумму ее диагональных элементов (саму матрицу сохранять не обязательно). Результат будем хранить в массиве объектов, со свойствами num и index [1]. Для последующей сортировки и идентификации. После сбора информации запустим алгоритм сортировки:

Рисунок 2. (реализация обработки ввода пользователя)

[1] Пример обработки: *[{'num': 35, 'index': 0}, {'num': -282, 'index': 1}, {'num': 83, 'index': 2}]*

1. Запустим функцию MergeSort, которая рекурсивно будет разбивать наш массив до тех пор пока мы не получим массив из 2 или 1 элемента. Затем начинается этап

сворачивания рекурсии при помощи Merge .

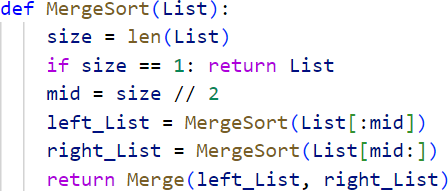


Рисунок 3. (Основная функция сортировки)

1. Функция Merge получает 2 массива и склеивает их сортируя элементы между собой, но зная, что внутри самих массивов элементы уже отсортированы. Так что на этом этапе нам достаточно 1 прохода. Таким образом мы сделаем (в худшем случае)

𝑛 log(𝑛) проходов. Так как количество элементов, которые нужно склеит возрастает как

2𝑛.

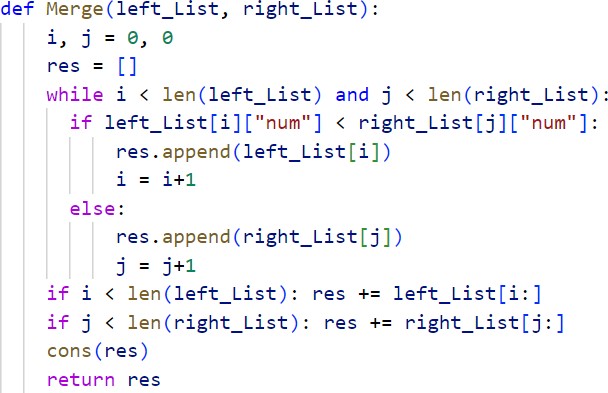
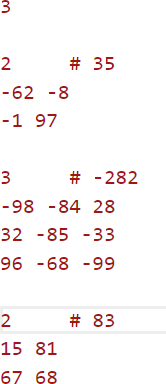


Рисунок 4. (Функция склеивания)

# Тесты.

* 1. Как первый тест запустим следующий тест:



Предполагаем вывод: 1 0 2 Вывод:



Как видим, алгоритм соединил сначала элементы *[1]* и *[2],*

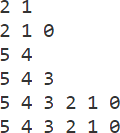
а после *[1, 2]* и *[0]*

* 1. Попробуем 1 и 0 элементов





Алгоритм отработал как и ожидалось. Ничего не вывел в 1 случае и единственный элемент 0 во 2 случае.

* 1. Тест с одинаковыми, нулевыми элементами. Запустим 6 нулевых матриц. Вывод:

Как видим, алгоритм все же провел сортировку и отсортировал их в обратном порядке, что не является ошибкой (Хоть и не совсем эффективно, но проверять отсортировал ли уже массив это еще дольше.)

# Выводы.

В этой лабораторной работе мы познакомились с сортировкой слиянием, разобрались на каждом шаге, как происходит сортировка. Для алгоритма мы использовали рекурсивные методы и сортировку слиянием и ее алгоритм. Объеденение массивов и др. Сложность алгоритма слиянием nlogn

**Приложение.**

Исходный код программы.

def cons(arr, result):

for el in arr:

result["str"] = result["str"] + str(el["index"]) + " "

result["str"] = result["str"] + "\n"

def Merge(left\_List, right\_List,result):

i, j = 0, 0

res = []

while i < len(left\_List) and j < len(right\_List):

if left\_List[i]["num"] < right\_List[j]["num"]:

res.append(left\_List[i])

i = i+1

else:

res.append(right\_List[j])

j = j+1

if i < len(left\_List): res += left\_List[i:]

if j < len(right\_List): res += right\_List[j:]

cons(res,result)

return res

def MergeSort(List,result):

size = len(List)

if size == 1: return List

mid = size // 2

left\_List = MergeSort(List[:mid],result)

right\_List = MergeSort(List[mid:],result)

return Merge(left\_List, right\_List,result)

def run(str):

result = {"str": ""}

myInput = list(str.split("\n"))

mtx = int(myInput.pop(0))

arr = []

for i in range(mtx):

size = int(myInput.pop(0))

summ = 0

for j in range(size):

summ += int(myInput.pop(0).split(" ")[j])

arr.append({"num": summ, "index": i})

if (len(arr) != 0):

cons(MergeSort(arr,result),result)

return result["str"]