

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ  
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №2  
по курсу «Алгоритмы и структуры данных»  
Тема: Сортировка слиянием. Метод декомпозиции  
Вариант 24

Выполнил:  
Субботин А. С.  
К3139

Проверила:  
Артамонова В.Е.

Санкт-Петербург  
2023 г.

## Содержание отчета

<b><u>ЗАДАЧИ ПО ВАРИАНТУ.....</u></b>	<b><u>3</u></b>
1 Задача – Сортировка вставкой .....	3
5 Задача – Представитель большинства [Вместо обязательного 6-го].....	5
8 Задача – Умножение многочленов.....	8
<b><u>ДОПОЛНИТЕЛЬНЫЕ ЗАДАЧИ.....</u></b>	<b><u>9</u></b>
2 Задача – Сортировка слиянием + .....	9
3 Задача – Число инверсий.....	12
4 Задача – Бинарный поиск .....	13
7 Задача – Поиск максимального подмассива за линейное время .....	16
9 Задача – Метод Штрассена для умножения матриц.....	18
<b><u>ВЫВОД ПО ЛАБОРАТОРНОЙ РАБОТЕ .....</u></b>	<b><u>19</u></b>

## Задачи по варианту

### 1 Задача – Сортировка вставкой

1. Используя *псевдокод* процедур Merge и Merge-sort из презентации к Лекции 2 (страницы 6-7), напишите программу сортировки слиянием на Python и проверьте сортировку, создав несколько случайных массивов, подходящих под параметры:

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число  $n$  ( $1 \leq n \leq 2 \cdot 10^4$ ) — число элементов в массиве. Во второй строке находятся  $n$  различных целых чисел, по модулю не превосходящих  $10^9$ .
- **Формат выходного файла (output.txt).** Одна строка выходного файла с отсортированным массивом. Между любыми двумя числами должен стоять ровно один пробел.

Листинг кода:

```
# Writing a merge_sort function
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i+=1
            else:
                arr[k] = right[j]
                j+=1
            k+=1
        while i < len(left):
            arr[k] = left[i]
            i+=1
            k+=1
        while j < len(right):
            arr[k] = right[j]
            j+=1
            k+=1
        return arr

# open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().split()))

a = merge_sort(a)

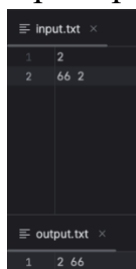
# Writing a function result into output.txt
```

```
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(' '.join(list(map(str, a))))
```

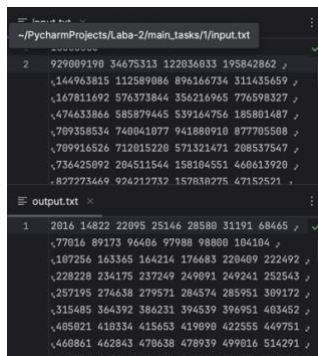
Описание решения:

Принимаем на вход массив, который дробим до тех пор, пока длина массива не будет равна 1 (применяется рекурсия) и возвращаем в левую часть массива меньшее значение, в правую большее и так до тех пор, пока массив не будет полностью отсортирован

Пример при минимальных значениях:



Пример при максимальных значениях:



	Время выполнения	Затраты памяти
Минимальное значение	0.0009090900421142578	15.57812

Максимальное значение	0.1432960033416748	26.703125	
Минимальное значение (Сортировка вставкой)	0.0007412433624267578	16.75	
Максимальное значение (Сортировка вставкой)	87.43030786514282	17.0625	

Все время указано в секундах, а память в Мб.

Вывод по задаче

Сортировка слиянием привлекает в быстроедействие, учитывая то, что она более ресурсо-затратная

Сортировка вставкой, соответственно, менее ресурсоёмкая, но больше занимает времени

По моему мнению производительней будет сортировка слиянием

## 5 Задача – Представитель большинства [Вместо обязательного 6-го]

Правило большинства - это когда выбирается элемент, имеющий больше половины голосов. Допустим, есть последовательность  $A$  элементов  $a_1, a_2, \dots, a_n$ , и нужно проверить, содержит ли она элемент, который появляется больше, чем  $n/2$  раз. Наивный метод это сделать:

```

Majority(A):
for i from 1 to n:
    current_element = a[i]
    count = 0
    for j from 1 to n:
        if a[j] = current_element:
            count = count+1
    if count > n/2:
        return a[i]
return "нет элемента большинства"

```

Очевидно, время выполнения этого алгоритма квадратично. Ваша цель - использовать метод "Разделяй и властвуй" для разработки алгоритма проверки, содержится ли во входной последовательности элемент, который встречается больше половины раз, за время  $O(n \log n)$ .

Листинг кода:

```

# Writing a Merge_Sort function
def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] < right[j]:
                arr[k] = left[i]
                i+=1
            else:
                arr[k] = right[j]
                j+=1
            k+=1
        while i < len(left):
            arr[k] = left[i]
            i+=1
            k+=1
        while j < len(right):
            arr[k] = right[j]
            j+=1
            k+=1
        return arr

# Поиск подмассива
def find(a):
    left, right = 0, 0
    while True:
        right += 1
        if right - left > len(a) / 2:
            return 1
        if a[right] != a[left]:
            left = right
        if right == len(a) - 1:
            return 0

# Open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().split()))

# Calling merge_sort function in a
merge_sort(a)

# Writing a find function result into output.txt
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(str(find(a)))

```

Описание решения:

Открываем input.txt и создаем переменные со значениями длины массива и самого массива

Сортируем массив с помощью сортировки вставки

С помощью функции `find` проверяем, повторяется ли больше  $n/2$  раз какое либо число

Открываем `output.txt` и записываем ответ

Пример из задачи 1:

```
input.txt x
1 5
2 2 3 9 2 2
output.txt x
1 1
```

Пример из задачи 2:

```
input.txt x
1 4
2 1 2 3 4
output.txt x
1 0
```

	Время выполнения	Затраты памяти
Пример из задачи 1	0.000619888305664062 5	14.765625
Пример из задачи 2	0.000166893005371093 75	15.203125

Вывод по задаче:

Я научился находить подмассивы

## 8 Задача – Умножение многочленов

Выдающийся немецкий математик Карл Фридрих Гаусс (1777—1855) заметил, что хотя формула для произведения двух комплексных чисел  $(a + bi)(c + di) = ac - bd + (bc + ad)i$  содержит *четыре* умножения вещественных чисел, можно обойтись и *тремя*: вычислим  $ac$ ,  $bd$  и  $(a + b)(c + d)$  и воспользуемся тем, что  $bc + ad = (a + b)(c + d) - ac - bd$ .

Листинг кода:

```
# Open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().split()))
    b = list(map(int, file.readline().split()))

# Multiplication of polynomials
res = [0] * (2*n)
for i in range(n):
    for j in range(n):
        res[i+j] += (a[i]*b[j])

while 0 in res:
    res.pop(res.index(0))

# Writing result into output.txt
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(' '.join(list(map(str, res))))
```

Описание решения:

Открываем input.txt и создаем переменные, в которых хранятся

- 1) порядок многочленов А и В
- 2) Коэффициенты многочлена А
- 3) Коэффициенты многочлена В

Создаем результирующий массив, с длиной порядка многочленов умноженной на 2 (возможны лишние элементы массива со знач 0)

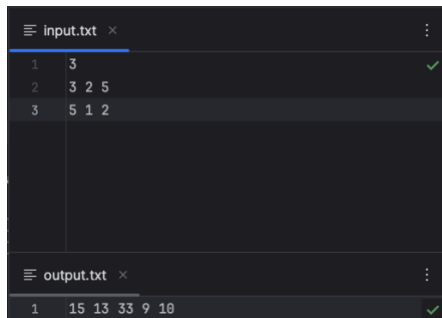
Через цикл фор проходимся по массивам и добавляем в I+J индекс рез. Массива перемножение многочленов  $A[i]*B[j]$



Удаляем лишние элементы массива

Записываем результат в output.txt

Пример из задания:



```
input.txt
1 3
2 3 2 5
3 5 1 2

output.txt
1 15 13 33 9 10
```

	Время выполнения	Затраты памяти
Пример из задачи	0.000652074813842773 4	14.828125

Вывод:

Я научился перемножать многочлены

## Дополнительные задачи

### 2 Задача – Сортировка слиянием +

Дан массив целых чисел. Ваша задача — отсортировать его в порядке неубывания *с помощью сортировки слиянием*.

Чтобы убедиться, что Вы действительно используете сортировку слиянием, мы просим Вас, после каждого осуществленного слияния (то есть, когда соответствующий подмассив уже отсортирован!), выводить индексы граничных элементов и их значения.

Листинг кода:

```

def merge_sort(arr):
    if len(arr) > 1:
        mid = len(arr)//2
        left = arr[:mid]
        right = arr[mid:]
        merge_sort(left)
        merge_sort(right)
        i = j = k = 0
        while i < len(left) and j < len(right):
            if left[i] > right[j]:
                arr[k] = left[i]
                i+=1
            else:
                arr[k] = right[j]
                j+=1
            k+=1
        while i < len(left):
            arr[k] = left[i]
            i+=1
            k+=1
        while j < len(right):
            arr[k] = right[j]
            j+=1
            k+=1
        return arr

# open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().split()))

a = merge_sort(a)

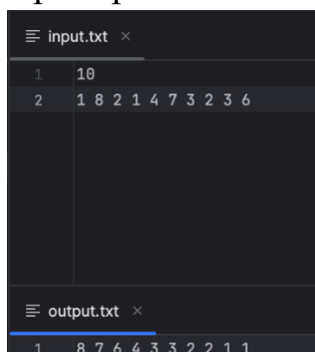
# Writing a function result into output.txt
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(' '.join(list(map(str, a))))

```

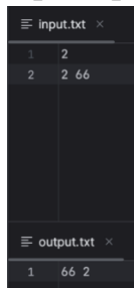
### Описание решения:

Решение аналогично первой задачке. Единственное, что мы меняем, это проверку Лево́й части и право́й, т.е. проверяем, что левая часть больше правой

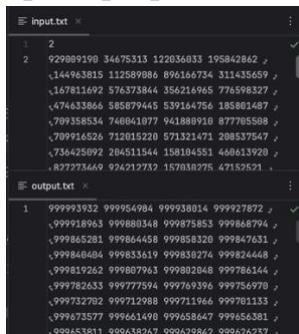
### Пример:



Пример при минимальных значениях:



Пример при максимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000154018402099609 38	14.875
Пример	0.000380277633666992 2	15.015625
Верхняя граница диапазона значений входных данных из текста задачи	0.13926911354064941	27.8125

### 3 Задача – Число инверсий

Инверсией в последовательности чисел  $A$  называется такая ситуация, когда  $i < j$ , а  $A_i > A_j$ . Количество инверсий в последовательности в некотором роде определяет, насколько близка данная последовательность к отсортированной. Например, в отсортированном массиве число инверсий равно 0, а в массиве, отсортированном наоборот - каждые два элемента будут составлять инверсию (всего  $n(n-1)/2$ ).

Дан массив целых чисел. Ваша задача — подсчитать число инверсий в нем.

Листинг кода:

```
# Writing a Score function
def score(a):
    n = len(a)
    k = 0
    for i in range(n):
        for j in range(0, n-i-1):
            if a[j]>a[j+1]:
                dope=a[j]
                a[j]=a[j+1]
                a[j+1]=dope
                k+=1
    return k

# Open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(int, file.readline().split()))

# Calling a score funtion
k = score(a)

# Writing a function result into output.txt
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(str(k))
```

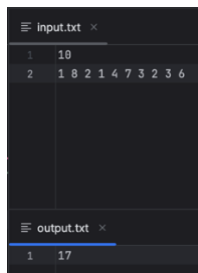
Описание решения:

Открываем input.txt и создаем переменные, которые принимают значения кол-ва массива и его значения

Создаем переменную K, которая принимает значение функции поиска кол-ва инверсий от массива a

Открываем output.txt и записываем ответ

Пример из задачи:



	Время выполнения	Затраты памяти
Пример из задачи	0.000643253326416015 6	14.90625

Вывод по задаче:

Я научился находить кол-во инверсий

## 4 Задача – Бинарный поиск

В этой задаче вы реализуете алгоритм бинарного поиска, который позволяет очень эффективно искать (даже в огромных) списках при условии, что список отсортирован. Цель - реализация алгоритма двоичного (бинарного) поиска.

Листинг кода:

```
# Writing a BinarySearch function
def BinarySearch(lys, val):
    first = 0 # начальный индекс массива
    last = len(lys)-1 # Последний индекс массива
    index = -1
    while (first <= last) and (index == -1):
        mid = (first+last)//2
        if lys[mid] == val:
            index = mid
        else:
            if val < lys[mid]:
                last = mid -1
            else:
                first = mid +1
    return index

# Open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    lys = list(map(int, file.readline().split()))
```

```

ka = int(file.readline())
k = list(map(int, file.readline().split()))

# Вызов функции для каждого числа
for i in range(len(k)):
    k[i] = BinarySearch(lys, k[i])

# Writing indexes into output.txt
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(' '.join(map(str, k)))

```

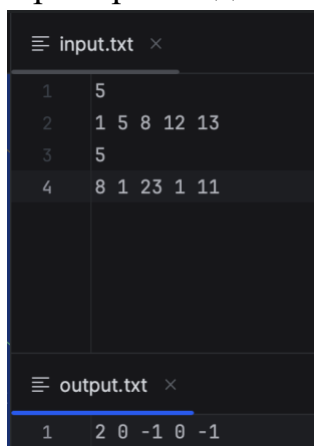
Описание решения:

Открываем input.txt и создаем переменные, которые принимают значения кол-ва массивов и их значений

Вызываем функцию для каждого элемента неотсортированного массива

Открываем output.txt и записываем ответ

Пример из задачи:



	Время выполнения	Затраты памяти
Пример из задачи	0.0001690387725830078	14.875

Вывод по задаче:

Я реализовал бинарный поиск

## 7 Задача – Поиск максимального подмассива за линейное время

Можно найти максимальный подмассив за линейное время, воспользовавшись следующими идеями. Начните с левого конца массива и двигайтесь вправо, отслеживая найденный к данному моменту максимальный подмассив. Зная максимальный подмассив массива  $A[1..j]$ , распространите ответ на поиск максимального подмассива, заканчивающегося индексом  $j + 1$ , воспользовавшись следующим наблюдением: максимальный подмассив массива  $A[1..j + 1]$  представляет собой либо максимальный подмассив массива  $A[1..j]$ , либо подмассив  $A[i..j + 1]$  для некоторого  $1 \leq i \leq j + 1$ . Определите максимальный подмассив вида  $A[i..j + 1]$  за константное время, зная максимальный подмассив, заканчивающийся индексом  $j$ .

Листинг кода:

```
# Writing a Find_Max_Crossing_Subarray function
def find_Max_subarray(array):
    max_sum = 0
    summ = 0
    start_index = 0
    end_index = 0

    for i in range(len(array)):

        if summ == 0:
            start_index = i

        summ += array[i]

        if max_sum < summ:
            max_sum = summ
            end_index = i

        if summ < 0:
            summ = 0

    return [start_index, end_index, max_sum]

# Open input.txt file
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    a = list(map(float, file.readline().split()))

# Находим начало/конец/сумму максимальной подпоследовательности
res = find_Max_subarray(a)
start, end, suma = res[0], res[1], res[2]

# Записываем ответ
with open('output.txt', 'w', encoding='UTF-8') as file:
    file.write(str(start)+' '+str(end)+' '+str(suma))
```



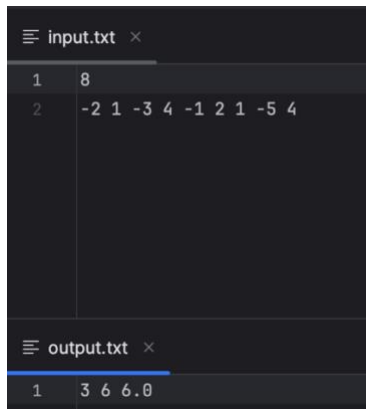
Описание решения:

Открываем input.txt и создаем переменные со значениями кол-ва массива и самого массива

С помощью функции считаем начальный и последний индекс подмассива и его сумму

Открываем output.txt и записываем туда решение

Пример:



```
input.txt x
1 8
2 -2 1 -3 4 -1 2 1 -5 4

output.txt x
1 3 6 6.0
```

	Время выполнения	Затраты памяти
Пример	0.000782966613769531 2	14.84375

Вывод:

Я научился находить начальный/конечный индекс максимального подмассива и его сумму

## 9 Задача – Метод Штрассена для умножения матриц

**Умножение матриц. Простой метод.** Если есть квадратные матрицы  $X = (x_{ij})$  и  $Y = (y_{ij})$ , то их произведение  $Z = X \cdot Y \Rightarrow z_{ij} = \sum_{k=1}^n x_{ik} \cdot y_{kj}$ . Нужно вычислить  $n^2$  элементов матрицы, каждый из которых представляет собой сумму  $n$  значений.

Листинг кода:

```
with open('input.txt', 'r', encoding='UTF-8') as file:
    n = int(file.readline())
    x = [0] * n
    y = [0] * n
    for i in range(n):
        x[i] = list(map(int, file.readline().split()))
    for i in range(n):
        y[i] = list(map(int, file.readline().split()))

# Результирующий массив
matr = [[0]*n for i in range(n)]

# Записываем в рез. массив перемноженные элементы
for i in range(n):
    for q in range(n):
        for k in range(n):
            matr[i][q] += x[i][k] * y[k][q]

with open('output.txt', 'w', encoding='UTF-8') as file:
    for i in range(n):
        file.write('\n')
        for j in range(n):
            file.write(str(matr[i][j])+' ')
```

Описание решения:

Открываем input.txt в котором:

Создаем переменные, в которых будут находиться размер матрицы и сами матрицы

Создаем результирующий массив, где будет наша матрица

Записываем результат перемножения матриц

Записываем ответ в output.txt

Пример:

input.txt	
1	2
2	5 3
3	7 5
4	3 6
5	9 2
output.txt	
1	
2	42 36
3	66 52

	Время выполнения	Затраты памяти
Пример из задачи	0.00044798851013183594	14.734375

Вывод:

Я научился перемножать матрицы с помощью метода Штрассена

### Вывод по лабораторной работе

Я понял как работает сортировка слиянием, алгоритмы перемножения матриц, поиска инверсий, поиска максимального подмассива, представителя большинства и перемножения многочленов