

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:

Янин Е. В.

К3141

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание отчета

Оглавление

Содержание отчета	2
Задачи по варианту	3
Задача №1. Сортировка слиянием.....	3
Задача №4. Бинарный поиск.....	4
Задача №5. Представитель большинства.....	5
Задача №6. Поиск максимальной прибыли	6
Задача №8. Умножение многочленов	8
Вывод.....	10

Задачи по варианту

Задача №1. Сортировка слиянием

В данной задаче требуется написать алгоритм сортировки слиянием и проверить его на нескольких массивах.

```
from math import floor
from utils import read_n_array, write_vars

def merge(lst, p, q, r):
    n1 = q - p + 1
    n2 = r - q
    left = [lst[p + i] for i in range(0, n1)]
    right = [lst[q + j + 1] for j in range(0, n2)]
    i, j = 0, 0
    for k in range(p, r + 1):
        if i == len(left):
            lst[k:r + 1] = right[j:len(right)]
            break
        elif j == len(right):
            lst[k:r + 1] = left[i:len(left)]
            break
        elif left[i] <= right[j]:
            lst[k] = left[i]
            i += 1
        else:
            lst[k] = right[j]
            j += 1
    return lst

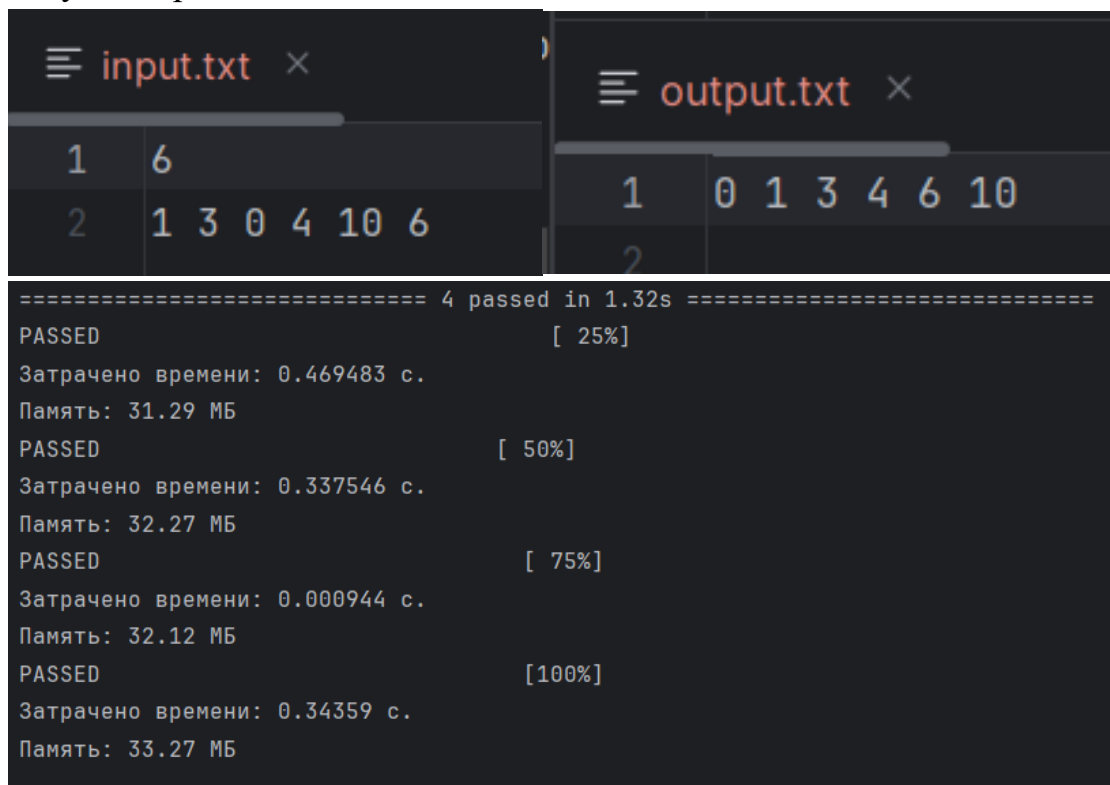
def merge_sort(lst, p, r):
    if p < r:
        q = floor((p + r) / 2)
        lst = merge_sort(lst, p, q)
        lst = merge_sort(lst, q + 1, r)
        lst = merge(lst, p, q, r)
        return lst
    else:
        return lst

def main(file):
    m, n = read_n_array(file)[0]
    write_vars('../tests/output.txt', merge_sort(m, 0, n - 1))
```

Объяснение решения:

- 1) Открываются два файла, входной и выходной
- 2) Массив рекурсивно делится на подмассивы, базовый случай – массив из одного элемента
- 3) Подмассивы собираются в отсортированном порядке с помощью функции merge
- 4) В файл output.txt выводится отсортированный массив.

Результат работы кода:



The screenshot shows a code editor with two tabs: 'input.txt' and 'output.txt'. The 'input.txt' tab contains two lines of data: the first line has '1' and '6', and the second line has '2' and '1 3 0 4 10 6'. The 'output.txt' tab contains two lines: the first line has '1' and '0 1 3 4 6 10', and the second line has '2' and an empty space. Below the tabs, the test results are displayed, showing four tests passed in 1.32s. Each test entry includes 'PASSED', a progress bar, execution time, and memory usage.

```
===== 4 passed in 1.32s =====
PASSED [ 25%]
Затрачено времени: 0.469483 с.
Память: 31.29 МБ
PASSED [ 50%]
Затрачено времени: 0.337546 с.
Память: 32.27 МБ
PASSED [ 75%]
Затрачено времени: 0.000944 с.
Память: 32.12 МБ
PASSED [100%]
Затрачено времени: 0.34359 с.
Память: 33.27 МБ
```

Вывод: я реализовал алгоритм сортировки слиянием и протестировал его на разных массивах.

Задача №4. Бинарный поиск

В данной задаче требуется реализовать алгоритм бинарного поиска.

```
from math import floor
from utils import read_n_array, write_vars

def binary_search(lst, low, high, key):
    if high < low:
        return low - 1
    if low == high:
        return -1
    mid = low + floor((high - low) / 2)
    if key == lst[mid]:
        return mid
    elif key < lst[mid]:
        return binary_search(lst, low, mid - 1, key)
    else:
        return binary_search(lst, mid + 1, high, key)

def main(file='input.txt'):
    input_data = read_n_array(file, num=2)
    a, b = input_data[0][0], input_data[1][0]
    ans = []
    for i in b:
        ans.append(binary_search(a, 0, len(a), i))
    write_vars('../tests/output.txt', ans)
```

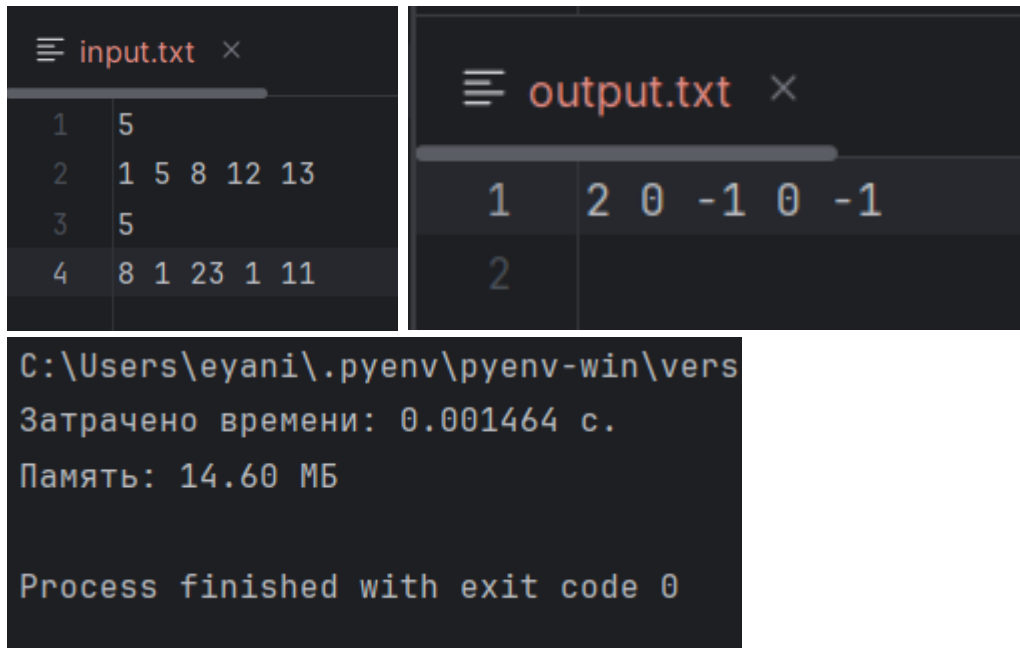
Объяснение решения:

1) Открываются два файла, входной и выходной, считываются элементы отсортированного массива и элементы, индекс которых нужно найти.

2) Массив делится пополам, если искомое значение больше медианного значения, то поиск продолжается в правой половине массива, если меньше – в левой, если равно, то возвращается индекс медианного значения

3) В файл output.txt выводятся индексы искомых элементов либо -1, если элементы не найдены.

Пример работы кода:



The image shows a code editor with three panels. The left panel shows 'input.txt' with the following content:

	input.txt
1	5
2	1 5 8 12 13
3	5
4	8 1 23 1 11

The right panel shows 'output.txt' with the following content:

	output.txt
1	2 0 -1 0 -1
2	

The bottom panel shows a terminal window with the following output:

```
C:\Users\eyani\.pyenv\pyenv-win\vers
Затрачено времени: 0.001464 с.
Память: 14.60 МБ
Process finished with exit code 0
```

Вывод: я реализовал алгоритм бинарного поиска.

Задача №5. Представитель большинства

В данной задаче требуется написать алгоритм, который будет проверять наличие в массиве такого элемента, который встречается более $n/2$ раз, используя метод «Разделяй и властвуй».

```
from utils import read_n_array, write_vars

def majority(lst):
    n = len(lst)
    mid = n // 2
    if n == 1:
        return lst[0], 1
    left_list = lst[:mid]
    right_list = lst[mid:]
    left_m = majority(left_list)
    left_count = right_list.count(left_m[0])
    right_m = majority(right_list)
    right_count = left_list.count(right_m[0])
    if left_count + left_m[1] > n // 2:
        return left_m[0], left_m[1] + left_count
```

```

if right_count + right_m[1] > n // 2:
    return right_m[0], right_m[1] + right_count
return None, 0

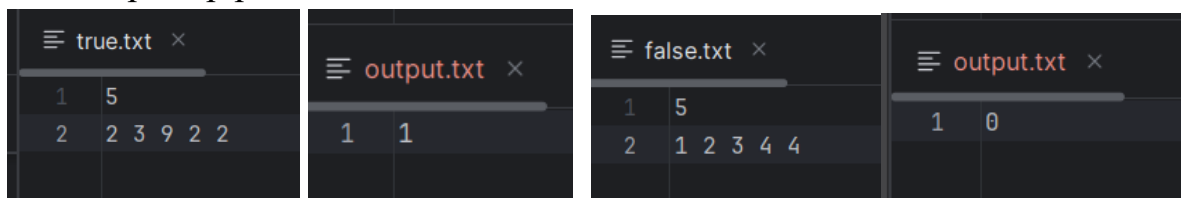
def main(file='input.txt'):
    A = read_n_array(file)[0][0]
    m = majority(A)
    if m[0] is None:
        write_vars('../tests/output.txt', '0')
    else:
        write_vars('../tests/output.txt', '1')

```

Объяснение решения:

- 1) Открываются два файла, входной и выходной
- 2) Массив рекурсивно делится на 2 подмассива, пока длина подмассива не равна 1, тогда возвращается значение этого массива и единица
- 3) Считается количество самого частого элемента одного подмассива в другом подмассиве
- 4) Если сумма количества элементов в массиве больше $n / 2$, то возвращается этот элемент и количество, иначе возвращается None и 0
- 5) Возвращается искомый элемент и его частота либо None и 0, если такого элемента нет
- 6) В файл output.txt выводится 0 или 1, в зависимости от того, найден ли элемент.

Пример работы кода:



Вывод: алгоритм справляется с нахождением элемента, который встречается более 2 раз.

Задача №6. Поиск максимальной прибыли

В данной задаче требуется написать алгоритм поиска максимального подмассива и с его помощью проанализировать данные по акциям фирмы с целью найти наиболее прибыльное время покупки и продажи ценных бумаг.

```

from math import floor

def find_max_crossing_subarray(A, low, mid, high):
    left_sum = -10**9
    s = 0
    for i in range(mid, low - 1, -1):
        s += A[i]

```

```

        if s > left_sum:
            left_sum = s
            left_max = i
    right_sum = -10**9
    s = 0
    for j in range(mid + 1, high + 1):
        s += A[j]
        if s > right_sum:
            right_sum = s
            right_max = j
    return left_max, right_max, left_sum + right_sum

def find_max_subarray(A, low, high):
    if low == high:
        return low, high, A[low]
    else:
        mid = floor((low + high) / 2)
        left_low, left_high, left_sum = find_max_subarray(A, low, mid)
        right_low, right_high, right_sum = find_max_subarray(A, mid + 1,
high)
        cross_low, cross_high, cross_sum = find_max_crossing_subarray(A,
low, mid, high)
        if left_sum >= right_sum and left_sum >= cross_sum:
            return left_low, left_high, left_sum
        elif right_sum >= left_sum and right_sum >= cross_sum:
            return right_low, right_high, right_sum
        else:
            return cross_low, cross_high, cross_sum

def main(file):
    with open(file) as fin, open('../tests/output.txt', 'w', encoding='UTF-
16') as fout:
        headers = [x for x in fin.readline().split()]
        lst, date = [], []
        name = str()
        for s in fin.readlines():
            data = list(s.split())
            lst.append(float(data[4]))
            date.append(data[2])
            name = data[0]
        delta = [lst[i] - lst[i-1] for i in range(1, len(lst))]
        start, end, stonks = find_max_subarray(delta, 0, len(delta)-1)
        fout.write(f'''{name}
02.09.24 - 01.10.24
Дата покупки: {date[start][:2]}.{date[start][2:4]}.{date[start][4:]}
Дата продажи: {date[end][:2]}.{date[end][2:4]}.{date[end][4:]}
Прибыль: {stonks}''')

```

Объяснение решения:

- 1) Открываются два файла, входной и выходной
- 2) Создаётся массив delta с изменением цены акций
- 3) Если массив состоит из одного элемента, возвращается этот массив (крайние точки и сам элемент)
- 4) Находится медианная точка массива, максимальный подмассив справа от неё, слева от неё и пересекающий её
- 5) Возвращается наибольший из них

6) По крайним индексам найденного подмассива определяется дата покупки и продажи, а по сумме – прибыль.

Пример работы кода:

```
aeroflot.txt ×
1 <TICKER> <PER> <DATE> <TIME> <CLOSE>
2 AFLT D 020924 000000 44.75
3 AFLT D 030924 000000 46.05
4 AFLT D 040924 000000 48.41
5 AFLT D 050924 000000 47.71
6 AFLT D 060924 000000 48.42
7 AFLT D 090924 000000 50.31
8 AFLT D 100924 000000 48.84
9 AFLT D 110924 000000 47.82
10 AFLT D 120924 000000 47.65
11 AFLT D 130924 000000 48.01
12 AFLT D 160924 000000 49.29
13 AFLT D 170924 000000 51.78
14 AFLT D 180924 000000 51.27
15 AFLT D 190924 000000 52.19
16 AFLT D 200924 000000 52.83
17 AFLT D 230924 000000 55
18 AFLT D 240924 000000 54.35
19 AFLT D 250924 000000 53.84
20 AFLT D 260924 000000 53.22
21 AFLT D 270924 000000 54.57
22 AFLT D 300924 000000 54.05
23 AFLT D 011024 000000 54.03
24

output.txt ×
1 AFLT
2 02.09.24 - 01.10.24
3 Дата покупки: 02.09.24
4 Дата продажи: 20.09.24
5 Прибыль: 10.25
```

Вывод: алгоритм справляется с нахождением подмассива с максимальной суммой элементов.

Задача №8. Умножение многочленов

В данной задаче требуется получить произведение двух многочленов, используя метод «Разделяй и властвуй»

```
from utils import read_array, write_vars

def addition(A, B, sign=True):
    if len(A) != len(B):
        if len(A) > len(B):
            n = len(A)
            B = (0,) * (n-len(B)) + B
        else:
            n = len(B)
            A = (0,) * (n - len(A)) + A
    if sign:
        return tuple(a + b for a, b in zip(A, B))
    else:
        return tuple(a - b for a, b in zip(A, B))
```



```

def multiply(A, B):
    if len(A) == len(B) == 1:
        return (A[0] * B[0], )
    n = len(A) // 2 * 2
    mid = len(A) // 2
    if len(A) % 2 == 0:
        a_1, a_2 = A[:mid], A[mid:]
        b_1, b_2 = B[:mid], B[mid:]
    else:
        a_1, a_2 = A[:mid+1], A[mid+1:]
        b_1, b_2 = B[:mid+1], B[mid+1:]
    c_1 = multiply(a_1, b_1)
    c_3 = multiply(a_2, b_2)
    add_1 = addition(a_1, a_2)
    add_2 = addition(b_1, b_2)
    c_2 = addition(addition(multiply(add_1, add_2), c_1, False), c_3,
False)
    m_1 = c_1 + (0, ) * n
    m_2 = c_2 + (0, ) * (n // 2)
    m_3 = c_3
    return addition(addition(m_1, m_2), m_3)

def main(file='input.txt'):
    input_data = read_array(file, 1, 2)
    A, B = map(tuple, input_data)
    write_vars('../tests/output.txt', multiply(A, B))

```

Объяснение решения:

- 1) Открываются два файла, входной и выходной, считываются коэффициенты 2 многочленов
- 2) Если многочлены состоят из 1 коэффициента, то возвращается их произведение
- 3) Иначе многочлены делятся на 2 части и возвращается $C = A(x) * B(x) = (A_1B_1)*x^n + (A_1B_2 + A_2B_1)*x^{n/2} + A_2B_2$
- 4) Сложение многочленов реализовано в функции addition путём параллельной итерации по коэффициентам и их сложения.

Пример работы кода:

≡ input.txt ×

1	3
2	3 2 5
3	5 1 2

≡ output.txt ×

1	15 13 33 9 10
2	

C:\Users\eyani\.pyenv\pyenv-win\versi

Затрачено времени: 0.000938 с.

Память: 15.27 МБ

Process finished with exit code 0

Вывод: я реализовал алгоритм сложения двух многочленов n -ой степени с помощью метода «Разделяй и властвуй».

Вывод

В данной лабораторной работе я реализовал несколько алгоритмов с помощью метода «Разделяй и властвуй», в том числе сортировку слиянием и бинарный поиск.