

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 11

Выполнил:  
Лютый Никита Артемович  
К3140

Проверил:  
Афанасьев А.В.

Санкт-Петербург  
2024 г.

## Содержание отчета

|   |           |
|---|-----------|
| <b>Содержание отчета</b>                                    | <b>2</b>  |
| <b>Задачи по варианту</b>                                   | <b>3</b>  |
| Задача №1. Сортировка слиянием                              | 3         |
| Задача №3. Число инверсий                                   | 9         |
| Задача №7. Поиск максимального подмассива за линейное время | 15        |
| <b>Дополнительные задачи</b>                                | <b>20</b> |
| Задача №2. Сортировка слиянием +                            | 20        |
| Задача №4. Бинарный поиск                                   | 28        |
| Задача №6. Поиск максимальной прибыли                       | 33        |
| <b>Вывод</b>  | <b>37</b> |

## Задачи по варианту

### Задача №1. Сортировка слиянием

#### 1 задача. Сортировка слиянием

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.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

2. Для проверки можно выбрать наихудший случай, когда сортируется массив размера  $1000, 10^4, 10^5$  чисел порядка  $10^9$ , отсортированных в обратном порядке; наилучший, когда массив уже отсортирован, и средний. Сравните, например, с сортировкой вставкой на этих же данных.
3. Перепишите процедуру Merge так, чтобы в ней не использовались сигнальные значения. Сигналом к остановке должен служить тот факт, что все элементы массива  $L$  или  $R$  скопированы обратно в массив  $A$ , после чего в этот массив копируются элементы, оставшиеся в непустом массиве.

*или* перепишите процедуру Merge (и, соответственно, Merge-sort) так, чтобы в ней не использовались значения границ и середины -  $p, r$  и  $q$ .

#### Листинг кода

```
import sys
def merge(lst, p, q, r):
    n1 = q-p+1
    n2 = r-q
    left = [0]*n1
    for j in range(0,n1):
        left[j]=lst[p+j]
    right = [0]*n2
    for j in range(0,n2):
        right[j]=lst[q+j+1]

    i, j = 0, 0
    k=p
    while i<n1 and j<n2:
        if left[i]<=right[j]:
            lst[k]=left[i]
            i+=1
        else:
```

```

        lst[k]=right[j]
        j+=1
        k+=1

    while i<n1:
        lst[k]=left[i]
        i+=1
        k+=1
    while j<n2:
        lst[k]=right[j]
        j+=1
        k+=1

sys.setrecursionlimit(100000000)
def merge_sort(lst, p,r):
    if p<r:
        q = (r+p)//2
        merge_sort(lst,p,q)
        merge_sort(lst,q+1,r)
        merge(lst,p,q,r)

def main():
    with open("input.txt") as f:
        n = int(f.readline())
        lst = list(map(int, f.readline().split()))
    max_n = 2*10**4
    max_el=10**9
    if n>max_n or n==0 or max(lst)>max_el or len(lst)!=n:
        quit("Incorrect input")
    merge_sort(lst,0,n-1)
    with open("output.txt","w") as f:
        f.write(' '.join(map(str,lst)))

main()

```

Тесты:

1) Тест на время работы:

```

from lab2.task1.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab2.task1.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: " +str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()

```

Текстовое объяснение решения:

Реализовано 3 функции:

1) `main()`:

Открывает файл `input.txt` и считывает из него `n` – количество элементов и сами элементы (записывает в переменную `lst`). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и перезаписывает переменную `lst`, применяя к ней процедуру сортировки `merge_sort()`. Каждый элемент которого переводит в строку и с помощью метода `join()` формирует выходное сообщение, которое далее записывает в файл `output.txt`.

2) `merge_sort()`:

Импортируем библиотеку `sys` и перед функцией выставляем с помощью `sys.setrecursionlimit()` максимальную глубину рекурсии.

Получает на вход начальный, конечный индекс списка и сам список. Если начальный индекс меньше конечного, то вычисляет индекс середины списка и вызывает себя с текущим списком и серединным индексом, как индексом конца. Затем вызывает себя с текущим списком и серединным индексом + 1 как начальным. А затем вызывает процедуру `merge()`, содержащую метод декомпозиции.

3) `merge()`:

Получает на вход список, индекс начала, середины и конца диапазона, в котором будет работать. Затем делит список на 2 части относительно середины: левую и правую. После этого в цикле `while`, пока 1 из списков (или оба) не кончатся, перебирает элементы этих списков, сравнивая их и ставя на соответствующую позицию в исходном списке. Как только элементы 1 списка кончились, элементы 2-го он дописывает до конца диапазона.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

| main.py input.txt |                     |
|-------------------|---------------------|
| 1                 | 10                  |
| 2                 | 1 5 6 4 3 8 9 2 0 3 |

| main.py output.txt |                     |
|--------------------|---------------------|
| 1                  | 0 1 2 3 3 4 5 6 8 9 |

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

| input.txt |   |
|-----------|---|
| 1         | 1 |
| 2         | 1 |

| output.txt |   |
|------------|---|
| 1          | 1 |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task1\tests\test\_time\_main.py  
Время работы программы 0.00038640000275336206 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task1\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.4794921875 KB

2) Значения из примера:

| input.txt |                     |
|-----------|---------------------|
| 1         | 10                  |
| 2         | 1 5 6 4 3 8 9 2 0 3 |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task1\tests\test\_time\_main.py  
Время работы программы 0.00046170002315193415 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task1\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.564453125 KB

3) Максимальные значения:

```
input.txt
This document contains very long lines. Soft wraps were enabled to improve editor performance.
1 10000
2 100000000 99999999 99999998 99999997 99999996 99999995 99999994 99999993 99999992 99999991
99999990 99999989 99999988 99999987 99999986 99999985 99999984 99999983 99999982 99999981
99999980 99999979 99999978 99999977 99999976 99999975 99999974 99999973 99999972 99999971
99999970 99999969 99999968 99999967 99999966 99999965 99999964 99999963 99999962 99999961
99999960 99999959 99999958 99999957 99999956 99999955 99999954 99999953 99999952 99999951
99999950 99999949 99999948 99999947 99999946 99999945 99999944 99999943 99999942 99999941
99999940 99999939 99999938 99999937 99999936 99999935 99999934 99999933 99999932 99999931
99999930 99999929 99999928 99999927 99999926 99999925 99999924 99999923 99999922 99999921
99999920 99999919 99999918 99999917 99999916 99999915 99999914 99999913 99999912 99999911
99999910 99999909 99999908 99999907 99999906 99999905 99999904 99999903 99999902 99999901
99999900 99999899 99999898 99999897 99999896 99999895 99999894 99999893 99999892 99999891
99999890 99999889 99999888 99999887 99999886 99999885 99999884 99999883 99999882 99999881
99999880 99999879 99999878 99999877 99999876 99999875 99999874 99999873 99999872 99999871
99999870 99999869 99999868 99999867 99999866 99999865 99999864 99999863 99999862 99999861
99999860 99999859 99999858 99999857 99999856 99999855 99999854 99999853 99999852 99999851
99999850 99999849 99999848 99999847 99999846 99999845 99999844 99999843 99999842 99999841
99999840 99999839 99999838 99999837 99999836 99999835 99999834 99999833 99999832 99999831
99999830 99999829 99999828 99999827 99999826 99999825 99999824 99999823 99999822 99999821
99999820 99999819 99999818 99999817 99999816 99999815 99999814 99999813 99999812 99999811
99999810 99999809 99999808 99999807 99999806 99999805 99999804 99999803 99999802 99999801
99999800 99999799 99999798 99999797 99999796 99999795 99999794 99999793 99999792 99999791
```

```
output.txt
This document contains very long lines. Soft wraps were enabled to improve editor performance.
1 99998001 99998002 99998003 99998004 99998005 99998006 99998007 99998008 99998009 99998010 ,
99998011 99998012 99998013 99998014 99998015 99998016 99998017 99998018 99998019 99998020 ,
99998021 99998022 99998023 99998024 99998025 99998026 99998027 99998028 99998029 99998030 ,
99998031 99998032 99998033 99998034 99998035 99998036 99998037 99998038 99998039 99998040 ,
99998041 99998042 99998043 99998044 99998045 99998046 99998047 99998048 99998049 99998050 ,
99998051 99998052 99998053 99998054 99998055 99998056 99998057 99998058 99998059 99998060 ,
99998061 99998062 99998063 99998064 99998065 99998066 99998067 99998068 99998069 99998070 ,
99998071 99998072 99998073 99998074 99998075 99998076 99998077 99998078 99998079 99998080 ,
99998081 99998082 99998083 99998084 99998085 99998086 99998087 99998088 99998089 99998090 ,
99998091 99998092 99998093 99998094 99998095 99998096 99998097 99998098 99998099 99998100 ,
99998101 99998102 99998103 99998104 99998105 99998106 99998107 99998108 99998109 99998110 ,
99998111 99998112 99998113 99998114 99998115 99998116 99998117 99998118 99998119 99998120 ,
99998121 99998122 99998123 99998124 99998125 99998126 99998127 99998128 99998129 99998130 ,
99998131 99998132 99998133 99998134 99998135 99998136 99998137 99998138 99998139 99998140 ,
99998141 99998142 99998143 99998144 99998145 99998146 99998147 99998148 99998149 99998150 ,
99998151 99998152 99998153 99998154 99998155 99998156 99998157 99998158 99998159 99998160 ,
99998161 99998162 99998163 99998164 99998165 99998166 99998167 99998168 99998169 99998170 ,
99998171 99998172 99998173 99998174 99998175 99998176 99998177 99998178 99998179 99998180 ,
99998181 99998182 99998183 99998184 99998185 99998186 99998187 99998188 99998189 99998190 ,
99998191 99998192 99998193 99998194 99998195 99998196 99998197 99998198 99998199 99998200 ,
99998201 99998202 99998203 99998204 99998205 99998206 99998207 99998208 99998209 99998210 ,
99998211 99998212 99998213 99998214 99998215 99998216 99998217 99998218 99998219 99998220 ,
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task1\tests\test_time_main.py
Время работы программы 0.04403739998815581 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task1\tests\test_memory_main.py
Максимально занимаемая память: 2222.146484375 KB
```

|  | Время выполнения, с | Затраты памяти, КВ |
|--|---------------------|--------------------|
| Нижняя граница диапазона значений входных данных из текста задачи  | 0.00039             | 17.4795            |
| Пример из задачи   | 0.00046             | 17.5645            |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.04404             | 2222.1465          |

Вывод по задаче:

Сортировка слиянием является более быстрой сортировкой, относительно рассмотренных в прошлой лабораторной, но более затратной по памяти.

## Задача №3. Число инверсий

### 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 сортированном массиве число инверсий равно 0, а в массиве, сортированном наоборот - каждые два элемента будут составлять инверсию (всего  $n(n-1)/2$ ).

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

Подсказка: чтобы сделать это быстрее, можно воспользоваться модификацией сортировки слиянием.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число  $n$  ( $1 \leq n \leq 10^5$ ) — число элементов в массиве. Во второй строке находятся  $n$  различных целых чисел, по модулю не превосходящих  $10^9$ .
- **Формат выходного файла (output.txt).** В выходной файл надо вывести число инверсий в массиве.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

- **Пример:**

| input.txt                 | output.txt |
|---------------------------|------------|
| 10<br>1 8 2 1 4 7 3 2 3 6 | 17         |

Листинг кода. (именно листинг, а не скрины)

```
import sys
def merge(lst, p, q, r, cnt):
    n1 = q-p+1
    n2 = r-q
    left = [0]*n1
    for j in range(0,n1):
        left[j]=lst[p+j]
    right = [0]*n2
    for j in range(0,n2):
        right[j]=lst[q+j+1]

    i, j = 0, 0
    k=p
    while i<n1 and j<n2:
        if left[i]<right[j]:
            lst[k]=left[i]
            cnt+=len(right[:j])
            i+=1
        elif left[i]>right[j]:
            lst[k]=right[j]
            j+=1
        else:
            i+=1
            j+=1
            k+=1
    while i<n1:
        lst[k]=left[i]
        i+=1
        k+=1
    while j<n2:
        lst[k]=right[j]
        j+=1
        k+=1
    return cnt
```



```

        lst[k] = left[i]
        i += 1
        k += 1
    while i < n1:
        lst[k] = left[i]
        cnt += len(right)
        i += 1
        k += 1
    while j < n2:
        lst[k] = right[j]
        j += 1
        k += 1
    return cnt

sys.setrecursionlimit(100000000)
def merge_sort(lst, p, r, cnt):
    if p < r:
        q = (r+p)//2
        cnt = merge_sort(lst, p, q, cnt)
        cnt = merge_sort(lst, q+1, r, cnt)
        cnt = merge(lst, p, q, r, cnt)

    return cnt

def main():
    with open("input.txt") as f:
        n = int(f.readline())
        lst = list(map(int, f.readline().split()))
    max_n = 10**5
    max_el = 10**9
    if n > max_n or n == 0 or max(lst) > max_el or len(lst) != n:
        quit("Incorrect input")
    cnt = 0
    cnt = merge_sort(lst, 0, n-1, cnt)
    with open("output.txt", "w") as f:
        f.write(str(cnt))

main()

```

Тесты:

### 1) Тест на время работы

```

from lab2.task3.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

### 2) Тест на занимаемую память

```

from lab2.task3.src.main import main
import tracemalloc

tracemalloc.start()

main()

```

```
print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()
```

Текстовое объяснение решения:

Реализовано 3 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Создает переменную cnt (счетчик инверсий) и перезаписывает его, вызывая функцию сортировки merge\_sort(). Затем выводит данный счетчик в файл output.txt.

2) merge\_sort():

Импортируем библиотеку sys и перед функцией выставяем с помощью setrecursionlimit() максимальную глубину рекурсии.

Получает на вход начальный, конечный индекс списка, сам список и счетчик инверсий. Если начальный индекс меньше конечного, то вычисляет индекс середины списка и вызывает себя с текущим списком и серединным индексом, как индексом конца, записывая результат работы в cnt. Затем вызывает себя с текущим списком и серединным индексом + 1 как начальным, записывая результат работы в cnt. А затем вызывает функцию merge(), содержащую метод декомпозиции, записывая результат работы в cnt.

3) merge():

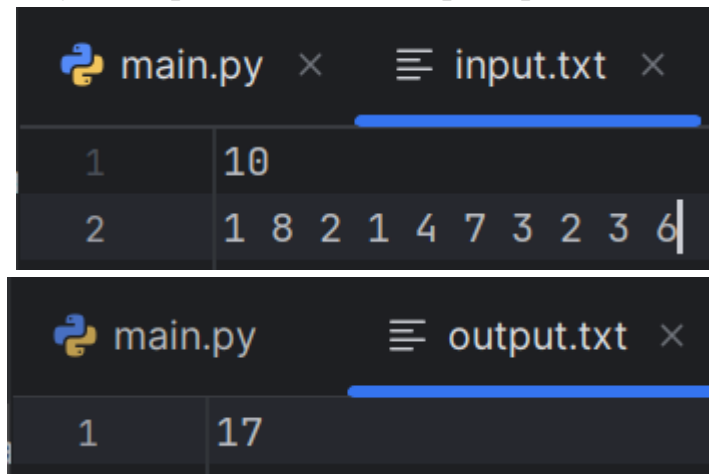
Получает на вход список, индекс начала, середины, конца диапазона, в котором будет работать и счетчик инверсий. Затем делит список на 2 части относительно середины: левую и правую. После этого в цикле while, пока 1 из списков (или оба) не кончатся, перебирает элементы этих списков, сравнивая их и ставя на соответствующую позицию в исходном списке, при этом каждый раз, когда элемент из левого списка оказывается меньше элемента из правого, прибавляет к cnt число записанных в исходный массив элементов из правого списка. Как только элементы 1 списка кончились, элементы 2-го он дописывает до конца диапазона. Если оставшиеся элементы из левого списка, то прибавляет к cnt количество элементов правого, после чего возвращает cnt.

Тесты:

1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.

- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из задачи:




```
main.py x input.txt x
1 10
2 1 8 2 1 4 7 3 2 3 6

main.py output.txt x
1 17
```

Результат работы кода на максимальных и минимальных значениях:

- 1) Минимальные значения:

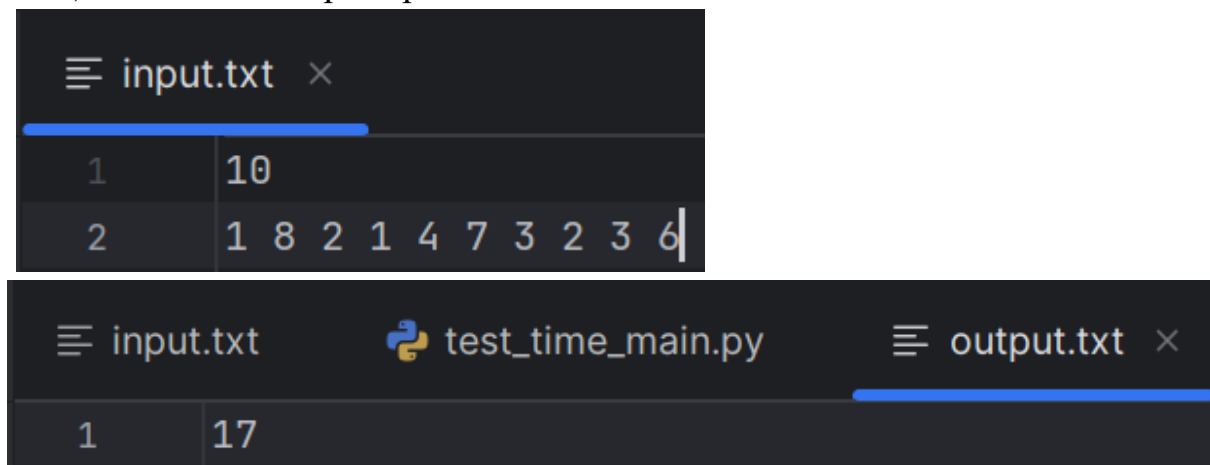


```
input.txt x
1 1
2 1

output.txt x
1 0
```

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task3\tests\test\_time\_main.py  
Время работы программы 0.00031380000291392207 секунд.  
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task3\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.4794921875 KB

- 2) Значения из примера:



```
input.txt x test_time_main.py output.txt x
1 10
2 1 8 2 1 4 7 3 2 3 6

1 17
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task3\tests\test_memory_main.py
Максимально занимаемая память: 17.564453125 KB

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task3\tests\test_time_main.py
Время работы программы 0.0003823000006377697 секунд.
```

### 3) Максимальные значения:



```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task3\tests\test_time_main.py
Время работы программы 0.332141800026875 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task3\tests\test_memory_main.py
Максимально занимаемая память: 9973.416015625 KB
```

|  | Время выполнения, с | Затраты памяти, KB |
|--|---------------------|--------------------|
| Нижняя граница диапазона значений входных данных из текста задачи  | 0.00031             | 17.4795            |
| Пример из задачи   | 0.00038             | 17.5645            |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.33214             | 9973.4160          |

Вывод по задаче:

Сортировка слиянием является более быстрой сортировкой, относительно рассмотренных в прошлой лабораторной, но более затратной по памяти.

## Задача №7. Поиск максимального подмассива за линейное время

### 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  $j$ .

В этом случае у вас возможны 2 варианта тестирования: первый предполагает создание случайного массива чисел, аналогично задаче №1 (в этом случае формат входного и выходного файла смотрите там). Второй вариант - взять любые данные по акциям какой-либо компании, аналогично задаче №6.

Листинг кода:

```
def find_subarr(lst, n):

    maxim = max(lst)
    if maxim <= 0:
        ind = lst.index(maxim)
        out = (ind, ind, maxim)
        return out

    ind_st = 0
    ind_fin = 0
    max_sum = 0
    max_pred_sum = 0
    out = (ind_st, ind_fin, max_sum)
    for i in range(n):
        max_pred_sum += lst[i]
        if max_pred_sum > 0:
            ind_fin = i
        else:
            ind_st = i + 1
            max_pred_sum = 0
            ind_fin = i + 1
        if max_sum <= max_pred_sum:
            max_sum = max_pred_sum
            out = (ind_st, ind_fin, max_sum)
    return out

def main():
    with open("input.txt") as f:
```

```

n = int(f.readline())
lst = list(map(int, f.readline().split()))
max_n = 2*10**4
max_el = 10**9
if n > max_n or n == 0 or max(lst) > max_el or len(lst) != n:
    quit("Incorrect input")

with open("output.txt", "w") as f:
    tuple_answ = find_subarr(lst, n)
    f.write(str(tuple_answ[2]) + '\n')
    f.write(' '.join(map(str, lst[tuple_answ[0]:tuple_answ[1]+1])))

main()

```

Тесты:

1) Тест на время работы:

```

from lab2.task7.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter() - time_st))

```

2) Тест на занимаемую память:

```

from lab2.task7.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: " + str(tracemalloc.get_traced_memory()[1]/1024) + " KB")
tracemalloc.stop()

```

Текстовое объяснение решения:

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). Записывает в переменную tuple\_answ результат работы функции find\_subarr(). Затем записывает ее содержимое в соответствующем порядке в файл output.txt (сначала максимальную сумму, а потом соответствующий срез списка).

2) find\_subarr():

Получает на вход длину списка и сам список. Проверяем максимальный элемент списка. Если он отрицательный или 0, то это и есть максимальная подпоследовательность и возвращаем ее. В противном случае создаем переменные ind\_st и ind\_fin (границы нашего максимального подмассива),

max\_sum и max\_pred\_sum, в которые будем записывать максимальную сумму и out – кортеж для вывода. Далее бежим по списку циклом for и методом префиксных сумм вычисляем максимальный подмассив. Во время перезаписывания максимальной суммы записываем индексы границ выбранного подмассива. После чего возвращаем кортеж out.

Тесты:

- 1) Импортируем нашу функцию main() и с помощью встроенной библиотеки time замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию main() и с помощью встроенной библиотеки tracemalloc замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

| input.txt |                  |
|-----------|------------------|
| 1         | 7                |
| 2         | -4 5 6 -3 -2 9 1 |

| input.txt | output.txt    |
|-----------|---------------|
| 1         | 16            |
| 2         | 5 6 -3 -2 9 1 |

Результат работы кода на максимальных и минимальных значениях:

- 1) Минимальные значения:

| input.txt |   | output.txt |   |
|-----------|---|------------|---|
| 1         | 1 | 1          | 1 |
| 2         | 1 | 2          | 1 |

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task7\tests\test_time_main.py
Время работы программы 0.0003288000007160008 секунд.
```

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task7\tests\test_memory_main.py
Максимально занимаемая память: 17.4794921875 KB
```

- 2) Значения из примера:



| input.txt |                  |
|-----------|------------------|
| 1         | 7                |
| 2         | -4 5 6 -3 -2 9 1 |

| input.txt |               | output.txt |  |
|-----------|---------------|------------|--|
| 1         | 16            |            |  |
| 2         | 5 6 -3 -2 9 1 |            |  |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task7\tests\test\_time\_main.py  
 Время работы программы 0.0003424998861923814 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task7\tests\test\_memory\_main.py  
 Максимально занимаемая память: 17.5576171875 KB

### 3) Максимальные значения:

| input.txt |  |
|-----------|--|
| 1         | 20000  |
| 2         | 1000000000 999999999 999999998 999999997 999999996 999999995 999999994 999999993 999999992 999999991 999999990 999999989 999999988 999999987 999999986 999999985 999999984 999999983 999999982 999999981 999999980 999999979 999999978 999999977 999999976 999999975 999999974 999999973 999999972 999999971 999999970 999999969 999999968 999999967 999999966 999999965 999999964 999999963 999999962 999999961 999999960 999999959 999999958 999999957 999999956 999999955 999999954 999999953 999999952 999999951 999999950 999999949 999999948 999999947 999999946 999999945 999999944 999999943 999999942 999999941 999999940 999999939 999999938 999999937 999999936 999999935 999999934 999999933 999999932 999999931 999999930 999999929 999999928 999999927 999999926 999999925 999999924 999999923 999999922 999999921 999999920 999999919 999999918 999999917 999999916 999999915 999999914 999999913 999999912 999999911 999999910 999999909 999999908 999999907 999999906 999999905 999999904 999999903 999999902 999999901 999999900 999999899 999999898 999999897 999999896 999999895 999999894 999999893 999999892 999999891 999999890 999999889 999999888 999999887 999999886 999999885 999999884 999999883 999999882 999999881 999999880 999999879 999999878 999999877 999999876 999999875 999999874 999999873 999999872 999999871 |

| output.txt |  |
|------------|--|
| 1          | 19999800010000   |
| 2          | 1000000000 999999999 999999998 999999997 999999996 999999995 999999994 999999993 999999992 999999991 999999990 999999989 999999988 999999987 999999986 999999985 999999984 999999983 999999982 999999981 999999980 999999979 999999978 999999977 999999976 999999975 999999974 999999973 999999972 999999971 999999970 999999969 999999968 999999967 999999966 999999965 999999964 999999963 999999962 999999961 999999960 999999959 999999958 999999957 999999956 999999955 999999954 999999953 999999952 999999951 999999950 999999949 999999948 999999947 999999946 999999945 999999944 999999943 999999942 999999941 999999940 999999939 999999938 999999937 999999936 999999935 999999934 999999933 999999932 999999931 999999930 999999929 999999928 999999927 999999926 999999925 999999924 999999923 999999922 999999921 999999920 999999919 999999918 999999917 999999916 999999915 999999914 999999913 999999912 999999911 999999910 999999909 999999908 999999907 999999906 999999905 999999904 999999903 999999902 999999901 999999900 999999899 999999898 999999897 999999896 999999895 999999894 999999893 999999892 999999891 999999890 999999889 999999888 999999887 999999886 999999885 999999884 999999883 999999882 999999881 999999880 999999879 999999878 999999877 999999876 999999875 999999874 999999873 999999872 999999871 |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task7\tests\test\_time\_main.py  
 Время работы программы 0.006743700010702014 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task7\tests\test\_memory\_main.py  
 Максимально занимаемая память: 2222.2607421875 KB



|   | Время выполнения, с | Затраты памяти, КВ |
|---|---------------------|--------------------|
| Нижняя граница<br>диапазона значений<br>входных данных из<br>текста задачи  | 0.00033             | 17.4795            |
| Пример из задачи  | 0.00034             | 17.5576            |
| Верхняя граница<br>диапазона значений<br>входных данных из<br>текста задачи | 0.00674             | 2222.2607          |

Вывод по задаче: реализованный алгоритм выбора максимального подмассива работает за линейное время, из-за чего является довольно быстрым.

### Задача №2. Сортировка слиянием +

#### 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.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число  $n$  ( $1 \leq n \leq 10^5$ ) — число элементов в массиве. Во второй строке находятся  $n$  различных целых чисел, по модулю не превосходящих  $10^9$ .
- **Формат выходного файла (output.txt).** Выходной файл состоит из нескольких строк.
  - В последней строке выходного файла требуется вывести отсортированный в порядке неубывания массив, данный на входе. Между любыми двумя числами должен стоять ровно один пробел.
  - Все предшествующие строки описывают осуществленные слияния, по одному на каждой строке. Каждая такая строка должна содержать по четыре числа:  $I_f, I_l, V_f, V_l$ , где  $I_f$  — индекс начала области слияния,  $I_l$  — индекс конца области слияния,  $V_f$  — значение первого элемента области слияния,  $V_l$  — значение последнего элемента области слияния.
  - Все индексы начинаются с единицы (то есть,  $1 \leq I_f \leq I_l \leq n$ ). **Индексы области слияния должны описывать положение области слияния в исходном массиве!** Допускается не выводить информацию о слиянии для подмассива длиной 1, так как он отсортирован по определению.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- **Приведем небольшой пример:** отсортируем массив  $[9, 7, 5, 8]$ . Рекурсивная часть сортировки слиянием (процедура  $\text{SORT}(A, L, R)$ , где  $A$  — сортируемый массив,  $L$  — индекс начала области слияния,  $R$  — индекс конца области слияния) будет вызвана с  $A = [9, 7, 5, 8]$ ,  $L = 1$ ,  $R = 4$  и выполнит следующие действия:
  - разделит область слияния  $[1, 4]$  на две части,  $[1, 2]$  и  $[3, 4]$ ;
  - выполнит вызов  $\text{SORT}(A, L = 1, R = 2)$ :
    - \* разделит область слияния  $[1, 2]$  на две части,  $[1, 1]$  и  $[2, 2]$ ;
    - \* получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
    - \* осуществит слияние, после чего  $A$  станет равным  $[7, 9, 5, 8]$ ;
    - \* выведет описание слияния:  $I_f = L = 1$ ,  $I_l = R = 2$ ,  $V_f = A_L = 7$ ,  $V_l = A_R = 9$ .
  - выполнит вызов  $\text{SORT}(A, L = 3, R = 4)$ :
    - \* разделит область слияния  $[3, 4]$  на две части,  $[3, 3]$  и  $[4, 4]$ ;
    - \* получившиеся части имеют единичный размер, рекурсивные вызовы можно не делать;
    - \* осуществит слияние, после чего  $A$  станет равным  $[7, 9, 5, 8]$ ;
    - \* выведет описание слияния:  $I_f = L = 3$ ,  $I_l = R = 4$ ,  $V_f = A_L = 5$ ,  $V_l = A_R = 8$ .
  - осуществит слияние, после чего  $A$  станет равным  $[5, 7, 8, 9]$ ;
  - выведет описание слияния:  $I_f = L = 1$ ,  $I_l = R = 4$ ,  $V_f = A_L = 5$ ,  $V_l = A_R = 9$ .

- Описания слияний могут идти в произвольном порядке, необязательно совпадающем с порядком их выполнения. Однако, с целью повышения производительности, рекомендуем выводить эти описания сразу, не храня их в памяти. Именно по этой причине отсортированный массив выводится в самом конце.

- Пример:

| input.txt           | output.txt          |
|---------------------|---------------------|
| 10                  | 1 2 1 8             |
| 1 8 2 1 4 7 3 2 3 6 | 3 4 1 2             |
|                     | 1 4 1 8             |
|                     | 5 6 4 7             |
|                     | 1 6 1 8             |
|                     | 7 8 2 3             |
|                     | 9 10 3 6            |
|                     | 7 10 2 6            |
|                     | 1 10 1 8            |
|                     | 1 1 2 2 3 3 4 6 7 8 |

Любая корректная сортировка слиянием, делящая подмассивы на две части (необязательно равных!), будет зачтена, если успеет завершиться, уложившись в ограничения.

## Листинг кода

```
import sys
def merge(lst, p, q, r):
    n1 = q-p+1
    n2 = r-q
    left = [0]*n1
    for j in range(0,n1):
        left[j]=lst[p+j]
    right = [0]*n2
    for j in range(0,n2):
        right[j]=lst[q+j+1]

    i, j = 0, 0
    k=p
    while i<n1 and j<n2:
        if left[i]<=right[j]:
            lst[k]=left[i]
            i+=1
        else:
            lst[k]=right[j]
            j+=1
        k+=1

    while i<n1:
        lst[k]=left[i]
        i+=1
        k+=1
    while j<n2:
        lst[k]=right[j]
        j+=1
        k+=1

sys.setrecursionlimit(100000000)
def merge_sort(lst, p,r,lst_act):
    if p<r:
        q = (r+p)//2
        merge_sort(lst,p,q, lst_act)
        merge_sort(lst,q+1,r, lst_act)
        merge(lst,p,q,r)
```

```

        lst_act.append((p,lst[p],r,lst[r]))

def main():
    with open("input.txt") as f:
        n = int(f.readline())
        lst = list(map(int, f.readline().split()))
        max_n = 2*10**4
        max_el=10**9
        lst_act = []
        if n>max_n or n==0 or max(lst)>max_el or len(lst)!=n:
            quit("Incorrect input")
        merge_sort(lst,0,n-1, lst_act)
        with open("output.txt","w") as f:
            for i in range(len(lst_act)):
                f.write(str(lst_act[i][0]+1)+", "+str(lst_act[i][2]+1)+", "+str(lst_act[i][1])+",
"+str(lst_act[i][3])+"\n")
            f.write(' '.join(map(str,lst)))

main()

```

Тесты:

1) Тест на время работы:

```

from lab2.task2.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab2.task2.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()

```

Текстовое объяснение решения.

Реализовано 3 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается), создает список lst\_act и перезаписывает его и переменную lst, применяя к ней процедуру

сортировки `merge_sort()`. Формирует выходное сообщение, которое далее записывает в файл `output.txt`.

## 2) `merge_sort()`:

Импортируем библиотеку `sys` и перед функцией выставляем с помощью `setrecursionlimit()` максимальную глубину рекурсии.

Получает на вход начальный, конечный индекс списка, сам список и список действий. Если начальный индекс меньше конечного, то вычисляет индекс середины списка и вызывает себя с текущим списком и серединным индексом, как индексом конца. Затем вызывает себя с текущим списком и серединным индексом + 1 как начальным. А затем вызывает процедуру `merge()`, содержащую метод декомпозиции. После этого записываем в список `lst_act` пограничные индексы.

## 3) `merge()`:

Получает на вход список, индекс начала, середины и конца диапазона, в котором будет работать. Затем делит список на 2 части относительно середины: левую и правую. После этого в цикле `while`, пока 1 из списков (или оба) не кончатся, перебирает элементы этих списков, сравнивая их и ставя на соответствующую позицию в исходном списке. Как только элементы 1 списка кончились, элементы 2-го он дописывает до конца диапазона.

## Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

| input.txt × |                     |
|-------------|---------------------|
| 1           | 10                  |
| 2           | 1 8 2 1 4 7 3 2 3 6 |

| input.txt |                     | main.py |  | output.txt × |  |
|-----------|---------------------|---------|--|--------------|--|
| 1         | 1, 2, 1, 8          |         |  |              |  |
| 2         | 1, 3, 1, 8          |         |  |              |  |
| 3         | 4, 5, 1, 4          |         |  |              |  |
| 4         | 1, 5, 1, 8          |         |  |              |  |
| 5         | 6, 7, 3, 7          |         |  |              |  |
| 6         | 6, 8, 2, 7          |         |  |              |  |
| 7         | 9, 10, 3, 6         |         |  |              |  |
| 8         | 6, 10, 2, 7         |         |  |              |  |
| 9         | 1, 10, 1, 8         |         |  |              |  |
| 10        | 1 1 2 2 3 3 4 6 7 8 |         |  |              |  |

Результат работы кода на максимальных и минимальных значениях:

1) Минимальные значения:

| input.txt × |   | output.txt × |   |
|-------------|---|--------------|---|
| 1           | 1 | 1            | 1 |
| 2           | 1 |              |   |

```
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task2\tests\test_time_main.py
Время работы программы 0.0007240000122692436 секунд.
"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs_DataStr\lab2\task2\tests\test_memory_main.py
Максимально занимаемая память: 17.4794921875 KB
```

2) Значения из примеров:

| input.txt × main.py |                     |
|---------------------|---------------------|
| 1                   | 10                  |
| 2                   | 1 8 2 1 4 7 3 2 3 6 |

| input.txt × main.py × output.txt × |                     |
|------------------------------------|---------------------|
| 1                                  | 1, 2, 1, 8          |
| 2                                  | 1, 3, 1, 8          |
| 3                                  | 4, 5, 1, 4          |
| 4                                  | 1, 5, 1, 8          |
| 5                                  | 6, 7, 3, 7          |
| 6                                  | 6, 8, 2, 7          |
| 7                                  | 9, 10, 3, 6         |
| 8                                  | 6, 10, 2, 7         |
| 9                                  | 1, 10, 1, 8         |
| 10                                 | 1 1 2 2 3 3 4 6 7 8 |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task2\tests\test\_time\_main.py  
Время работы программы 0.000830000004498288 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task2\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.564453125 KB

### 3) Максимальные значения:

| input.txt × |   |
|-------------|---|
| 1           | 20000   |
| 2           | 1000000000 99999999 99999998 99999997 99999996 99999995 99999994 99999993 99999992 99999991 ↗ |
|             | ↖ 99999990 99999989 99999988 99999987 99999986 99999985 99999984 99999983 99999982 99999981 ↗ |
|             | ↖ 99999980 99999979 99999978 99999977 99999976 99999975 99999974 99999973 99999972 99999971 ↗ |
|             | ↖ 99999970 99999969 99999968 99999967 99999966 99999965 99999964 99999963 99999962 99999961 ↗ |
|             | ↖ 99999960 99999959 99999958 99999957 99999956 99999955 99999954 99999953 99999952 99999951 ↗ |
|             | ↖ 99999950 99999949 99999948 99999947 99999946 99999945 99999944 99999943 99999942 99999941 ↗ |
|             | ↖ 99999940 99999939 99999938 99999937 99999936 99999935 99999934 99999933 99999932 99999931 ↗ |
|             | ↖ 99999930 99999929 99999928 99999927 99999926 99999925 99999924 99999923 99999922 99999921 ↗ |
|             | ↖ 99999920 99999919 99999918 99999917 99999916 99999915 99999914 99999913 99999912 99999911 ↗ |
|             | ↖ 99999910 99999909 99999908 99999907 99999906 99999905 99999904 99999903 99999902 99999901 ↗ |
|             | ↖ 99999900 99999899 99999898 99999897 99999896 99999895 99999894 99999893 99999892 99999891 ↗ |
|             | ↖ 99999890 99999889 99999888 99999887 99999886 99999885 99999884 99999883 99999882 99999881 ↗ |
|             | ↖ 99999880 99999879 99999878 99999877 99999876 99999875 99999874 99999873 99999872 99999871 ↗ |
|             | ↖ 99999870 99999869 99999868 99999867 99999866 99999865 99999864 99999863 99999862 99999861 ↗ |
|             | ↖ 99999860 99999859 99999858 99999857 99999856 99999855 99999854 99999853 99999852 99999851 ↗ |
|             | ↖ 99999850 99999849 99999848 99999847 99999846 99999845 99999844 99999843 99999842 99999841 ↗ |
|             | ↖ 99999840 99999839 99999838 99999837 99999836 99999835 99999834 99999833 99999832 99999831 ↗ |
|             | ↖ 99999830 99999829 99999828 99999827 99999826 99999825 99999824 99999823 99999822 99999821 ↗ |
|             | ↖ 99999820 99999819 99999818 99999817 99999816 99999815 99999814 99999813 99999812 99999811 ↗ |

|    |                              |
|----|------------------------------|
| 1  | 1, 2, 999999999, 1000000000  |
| 2  | 1, 3, 999999998, 1000000000  |
| 3  | 4, 5, 999999996, 999999997   |
| 4  | 1, 5, 999999996, 1000000000  |
| 5  | 6, 7, 999999994, 999999995   |
| 6  | 6, 8, 999999993, 999999995   |
| 7  | 9, 10, 999999991, 999999992  |
| 8  | 6, 10, 999999991, 999999995  |
| 9  | 1, 10, 999999991, 1000000000 |
| 10 | 11, 12, 999999989, 999999990 |
| 11 | 11, 13, 999999988, 999999990 |
| 12 | 14, 15, 999999986, 999999987 |
| 13 | 11, 15, 999999986, 999999990 |
| 14 | 16, 17, 999999984, 999999985 |
| 15 | 16, 18, 999999983, 999999985 |
| 16 | 19, 20, 999999981, 999999982 |
| 17 | 16, 20, 999999981, 999999985 |
| 18 | 11, 20, 999999981, 999999990 |
| 19 | 1, 20, 999999981, 1000000000 |
| 20 | 21, 22, 999999979, 999999980 |
| 21 | 21, 23, 999999978, 999999980 |
| 22 | 24, 25, 999999976, 999999977 |
| 23 | 21, 25, 999999976, 999999980 |
| 24 | 26, 27, 999999974, 999999975 |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task2\tests\test\_time\_main.py  
 Время работы программы 0.11878119999892078 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task2\tests\test\_memory\_main.py  
 Максимально занимаемая память: 4282.255859375 KB



|  | Время выполнения, с | Затраты памяти, КВ |
|--|---------------------|--------------------|
| Нижняя граница диапазона значений входных данных из текста задачи  | 0.00072             | 17.4795            |
| Пример из задачи   | 0.00083             | 17.5645            |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.11878             | 4282.2559          |

Вывод по задаче: Сортировка слиянием является более быстрой сортировкой, относительно рассмотренных в прошлой лабораторной, но более затратной по памяти.

## Задача №4. Бинарный поиск

### 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.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число  $n$  ( $1 \leq n \leq 10^5$ ) — число элементов в массиве, и последовательность  $a_0 < a_1 < \dots < a_{n-1}$  из  $n$  **различных** положительных целых чисел в порядке возрастания,  $1 \leq a_i \leq 10^9$  для всех  $0 \leq i < n$ . Следующая строка содержит число  $k$ ,  $1 \leq k \leq 10^5$  и  $k$  положительных целых чисел  $b_0, \dots, b_{k-1}$ ,  $1 \leq b_j \leq 10^9$  для всех  $0 \leq j < k$ .
- **Формат выходного файла (output.txt).** Для всех  $i$  от 0 до  $k - 1$  вывести индекс  $0 \leq j \leq n - 1$ , такой что  $a_i = b_j$  или -1, если такого числа в массиве нет.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.
- Пример:

| input.txt   | output.txt  |
|-------------|-------------|
| 5           | 2 0 -1 0 -1 |
| 1 5 8 12 13 |             |
| 5           |             |
| 8 1 23 1 11 |             |

В этом примере есть возрастающая последовательность из  $a_0 = 1, a_1 = 5, a_2 = 8, a_3 = 12$  и  $a_4 = 13$  длиной в  $n = 5$  и пять чисел для поиска: 8 1 23 1 11. Видно, что  $a_2 = 8$  и  $a_0 = 1$ , но чисел 23 и 11 нет в последовательности  $a$ , поэтому они имеют индекс -1. В итоге ответ: 2 0 -1 0 -1.

### Листинг кода

```
def binary_search(lst, low, high, elem):
    while high >= low:
        mid = (high + low) // 2
        if lst[mid] == elem:
            return mid
        elif lst[mid] < elem:
            low = mid + 1
        else:
            high = mid - 1
    return -1

def main():
    with open("input.txt") as f:
        n = int(f.readline())
        lst = list(map(int, f.readline().split()))
        k = int(f.readline())
        lst_el = list(map(int, f.readline().split()))
    max_n = 10**5
    max_el = 10**9
```

```

    if n>max_n or n==0 or max(lst)>max_el or len(lst)!=n or k>max_n or max(lst_el)>max_el or k==0 or
len(lst_el)!=k:
        quit("Incorrect input")
    lst_answ = []
    for i in range(k):
        lst_answ.append(binary_search(lst, 0, n-1, lst_el[i]))
    with open("output.txt", "w") as f:
        f.write(' '.join(map(str, lst_answ)))

main()

```

Тесты:

1) Тест на время работы:

```

from lab2.task4.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab2.task4.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: " + str(tracemalloc.get_traced_memory()[1]/1024) + " KB")
tracemalloc.stop()

```

Текстовое объяснение решения.

Реализовано 2 функции:

1) main():

Открывает файл input.txt и считывает из него n (длина списка), lst (список), k (длина списка элементов для поиска), lst\_el (список элементов для поиска). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается). В переменную lst\_answ записывает результат работы функции binary\_search() для каждого элемента из списка lst\_el. Затем формирует выходное сообщение, которое далее записывает в файл output.txt.

2) binary\_search():

Получает на вход элемент для поиска, индексы начала и конца списка и сам список. Затем, в цикле вычисляет серединный индекс и проверяет элемент на данной позиции. Если равен нужному – возвращает индекс, меньше –

делает верхней границей значение `mid`, если больше – нижней границей делает значение `mid+1`. Если элемента нет – выводит -1.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

| input.txt |             |  |  | output.txt |             |
|-----------|-------------|--|--|------------|-------------|
| 1         | 5           |  |  |            |             |
| 2         | 1 5 8 12 13 |  |  |            |             |
| 3         | 5           |  |  |            |             |
| 4         | 8 1 23 1 11 |  |  | 1          | 2 0 -1 0 -1 |

Результат работы кода на максимальных и минимальных значениях:

- 1) Минимальные значения:

| input.txt |   |  |  | output.txt |  |
|-----------|---|--|--|------------|--|
| 1         | 1 |  |  |            |  |
| 2         | 1 |  |  |            |  |
| 3         | 1 |  |  |            |  |
| 4         | 1 |  |  |            |  |

| input.txt | test_time_main.py | output.txt |
|-----------|-------------------|------------|
| 1         | 0                 |            |

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task4\tests\test\_time\_main.py  
Время работы программы 0.0007510000141337514 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task4\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.5712890625 KB

## 2) Значения из примера:

input.txt

|   |             |
|---|-------------|
| 1 | 5           |
| 2 | 1 5 8 12 13 |
| 3 | 5           |
| 4 | 8 1 23 1 11 |

output.txt

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

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task4\tests\test\_time\_main.py  
Время работы программы 0.000813300022855401 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task4\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.6806640625 KB

## 3) Максимальные значения:

```
100000
999900001 999900002 999900003 999900004 999900005 999900006 999900007 999900008 999900009 999900010
999900011 999900012 999900013 999900014 999900015 999900016 999900017 999900018 999900019 999900020
999900021 999900022 999900023 999900024 999900025 999900026 999900027 999900028 999900029 999900030
999900031 999900032 999900033 999900034 999900035 999900036 999900037 999900038 999900039 999900040
999900041 999900042 999900043 999900044 999900045 999900046 999900047 999900048 999900049 999900050
999900051 999900052 999900053 999900054 999900055 999900056 999900057 999900058 999900059 999900060
999900061 999900062 999900063 999900064 999900065 999900066 999900067 999900068 999900069 999900070
999900071 999900072 999900073 999900074 999900075 999900076 999900077 999900078 999900079 999900080
999900081 999900082 999900083 999900084 999900085 999900086 999900087 999900088 999900089 999900090
999900091 999900092 999900093 999900094 999900095 999900096 999900097 999900098 999900099 999900100
999900101 999900102 999900103 999900104 999900105 999900106 999900107 999900108 999900109 999900110
999900111 999900112 999900113 999900114 999900115 999900116 999900117 999900118 999900119 999900120
999900121 999900122 999900123 999900124 999900125 999900126 999900127 999900128 999900129 999900130
999900131 999900132 999900133 999900134 999900135 999900136 999900137 999900138 999900139 999900140
999900141 999900142 999900143 999900144 999900145 999900146 999900147 999900148 999900149 999900150
999900151 999900152 999900153 999900154 999900155 999900156 999900157 999900158 999900159 999900160
999900161 999900162 999900163 999900164 999900165 999900166 999900167 999900168 999900169 999900170
999900171 999900172 999900173 999900174 999900175 999900176 999900177 999900178 999900179 999900180
999900181 999900182 999900183 999900184 999900185 999900186 999900187 999900188 999900189 999900190
999900191 999900192 999900193 999900194 999900195 999900196 999900197 999900198 999900199 999900200
999900201 999900202 999900203 999900204 999900205 999900206 999900207 999900208 999900209 999900210
999900211 999900212 999900213 999900214 999900215 999900216 999900217 999900218 999900219 999900220
999900221 999900222 999900223 999900224 999900225 999900226 999900227 999900228 999900229 999900230
```



## Задача №6. Поиск максимальной прибыли

### 6 задача. Поиск максимальной прибыли

Используя *псевдокод* процедур Find Maximum Subarray и Find Max Crossing Subarray из презентации к Лекции 2 (страницы 25-26), напишите программу поиска максимального подмассива.

Примените ваш алгоритм для ответа на следующий вопрос. Допустим, у нас есть данные по акциям какой-либо фирмы за последний месяц (год, или иной срок).

Проанализируйте этот срок и выдайте ответ, в какой из дней при покупке единицы акции данной фирмы, и в какой из дней продажи, вы бы получили максимальную прибыль? Выдайте дату покупки, дату продажи и максимальную прибыль.

Вы можете использовать любые данные для своего анализа. Например, я набрала в Google "*акции*" и мне поиск выдал акции Газпрома, [тут](#) - можно скачать информацию по стоимости акций за любой период. (Перейдя по ссылке, нажмите на вкладку "Настройки" → "Скачать")

Соответственно, вам нужно только выбрать данные, посчитать *изменение цены* и применить алгоритм поиска максимального подмассива.

### Листинг кода

```
import sys

sys.setrecursionlimit(10000000)
def find_max_subarr(lst, low, high):
    if high == low:
        return (low, high, 0)
    else:
        mid = (low + high) // 2
        leftlow, lefthigh, leftsum = find_max_subarr(lst, low, mid)
        rightlow, righthigh, rightsum = find_max_subarr(lst, mid + 1, high)
        crosslow, crosshigh, crosssum = find_max_cross_subarr(lst, low, mid, high)
        if leftsum >= rightsum and leftsum >= crosssum:
            return (leftlow, lefthigh, leftsum)
        elif rightsum >= crosssum and rightsum >= leftsum:
            return (rightlow, righthigh, rightsum)
        else:
            return (crosslow, crosshigh, crosssum)

def find_max_cross_subarr(lst, low, mid, high):
    leftsum = -10**9
    rightsum = -10**9
    elem = lst[mid]
    sum = 0
    maxleft = mid
    for i in range(mid, low - 1, -1):
        sum = elem - lst[i]
        if sum > leftsum:
            leftsum = sum
            maxleft = i
    sum = 0
    maxright = mid
    for i in range(mid + 1, high):
```

```

    sum = lst[i]-elem
    if sum > rightsum:
        rightsum = sum
        maxright = i
    return (maxleft, maxright, rightsum+leftsum)

def main():
    with open("input.txt") as f:
        n = int(f.readline())
        lst = list(map(float, f.readline().split()))
        max_n = 10**5
        max_el=10**9

    if n>max_n or n==0 or max(lst)>max_el or len(lst)!=n:
        quit("Incorrect input")

    with open("output.txt", "w") as f:
        f.write('Yandex, month (september): \n')
        lst_ans = find_max_subarr(lst, 0, n)
        f.write("Buy: "+str(lst_ans[0]+1)+".09, Sell: "+str(lst_ans[1]+1)+".09\n")
        f.write("Result: "+str(lst_ans[2])+" rub.")

main()

```

Тесты:

1) Тест на время работы:

```

from lab2.task6.src.main import main
import time

time_st = time.perf_counter()

main()

print(f'Время работы программы %s секунд.' % (time.perf_counter()-time_st))

```

2) Тест на занимаемую память:

```

from lab2.task6.src.main import main
import tracemalloc

tracemalloc.start()

main()

print("Максимально занимаемая память: "+str(tracemalloc.get_traced_memory()[1]/1024)+" KB")
tracemalloc.stop()

```

Текстовое объяснение решения:

Реализовано 3 функции:

1) main():

Открывает файл input.txt и считывает из него n – количество элементов и сами элементы (записывает в переменную lst). Проверяет входные данные на соответствие условию задачи (в случае несоответствия выводится ошибка и программа останавливается) и записывает в переменную lst\_ans



результат поиска функции `find_max_subarr()`. Затем с помощью метода `join()` и обращения элементов списка в строку формирует выходное сообщение, которое далее записывает в файл `output.txt`.

## 2) `find_max_subarr()`:

Получает на вход стартовый и конечный индексы списка и сам список. Если индексы равны – возвращаем кортеж из них и нуля. В противном случае вычисляем серединный индекс и проверяем 3 случая: ищем максимальный подмассив слева и справа (рекурсивным вызовом функции) и ищем его на пересечении середины с помощью функции `find_max_cross_subarr()`.

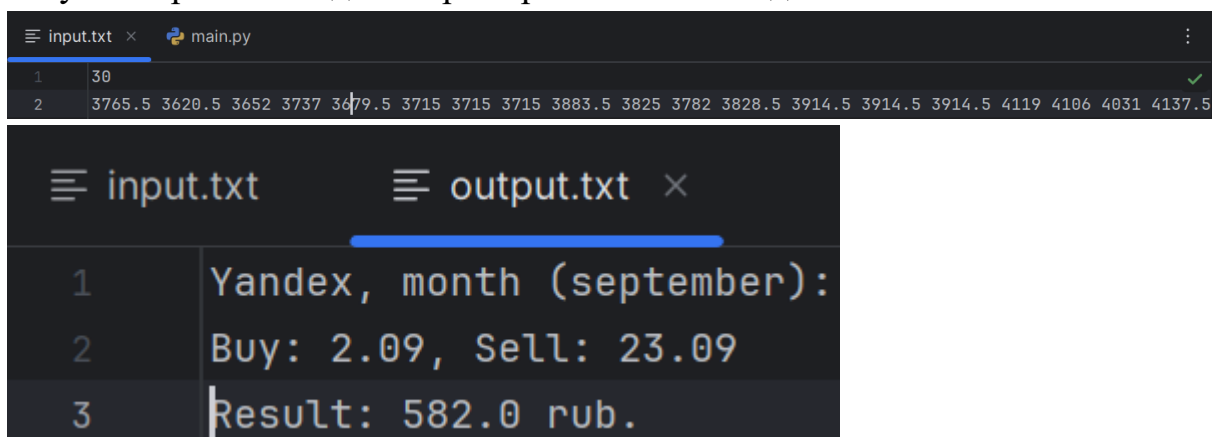
## 3) `find_max_cross_subarr()`:

Получает на вход список, стартовый, серединный и конечный индекс диапазона. Ищет слева и справа максимальную разность элементов, относительно середины, а затем складывает их и возвращает индексы границ и сумму.

Тесты:

- 1) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `time` замеряем время работы программы и выводим его.
- 2) Импортируем нашу функцию `main()` и с помощью встроенной библиотеки `tracemalloc` замеряем занимаемую память в ходе выполнения программы и выводим пиковое значение.

Результат работы кода на примерах из текста задачи:

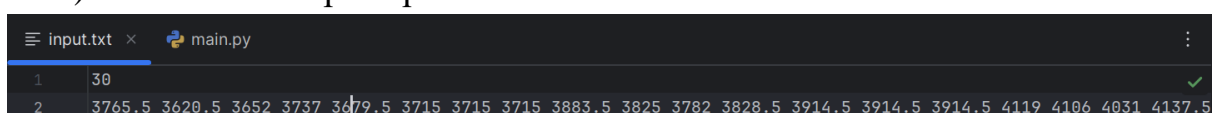


```
input.txt x main.py
1 30
2 3765.5 3620.5 3652 3737 3679.5 3715 3715 3715 3883.5 3825 3782 3828.5 3914.5 3914.5 3914.5 4119 4106 4031 4137.5

input.txt output.txt x
1 Yandex, month (september):
2 Buy: 2.09, Sell: 23.09
3 Result: 582.0 rub.
```

Результат работы кода на максимальных и минимальных значениях:

- 1) Значения из примера:



```
input.txt x main.py
1 30
2 3765.5 3620.5 3652 3737 3679.5 3715 3715 3715 3883.5 3825 3782 3828.5 3914.5 3914.5 3914.5 4119 4106 4031 4137.5
```

≡ input.txt

≡ output.txt ×

1

Yandex, month (september):

2

Buy: 2.09, Sell: 23.09

3

Result: 582.0 rub.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task6\tests\test\_time\_main.py  
Время работы программы 0.000857400094085932 секунд.

"C:\Program Files\Python311\python.exe" C:\Users\smaf1\OneDrive\Desktop\ITM0\Algs\_DataStr\lab2\task6\tests\test\_memory\_main.py  
Максимально занимаемая память: 17.869140625 KB

|                  |                     |                    |
|------------------|---------------------|--------------------|
|                  | Время выполнения, с | Затраты памяти, KB |
| Пример из задачи | 0.00086             | 17.8691            |

Вывод по задаче: Был реализован поиск максимальной прибыли с помощью псевдо-кода поиска максимального подмассива из лекции.

## **Вывод**

Был изучен метод декомпозиции, бинарный поиск и сортировка слиянием, имеющие среднюю сложность  $O(n\log(n))$ , и применены на практике.