

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.

Выполнил: Криличевский М. Е.

Номер группы: К3139

Проверил: Афанасьев А. В.

Санкт-Петербург

2024 г.

Содержание

<i>Задание 1</i>	3
<i>Задание 3</i>	7
<i>Задание 4</i>	9
<i>Задание 5</i>	12
<i>Задание 6</i>	15
<i>Задание 9</i>	19
<i>Вывод</i>	24

Задание 1

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 .

Решение:

```
from lab_2.utils import *

def merge(left_list, right_list):
    res = []
    i = j = 0

    left_len, right_len = len(left_list), len(right_list)

    for _ in range(left_len + right_len):
        if i < left_len and j < right_len:
            if left_list[i] <= right_list[j]:
                res.append(left_list[i])
                i += 1
            else:
                res.append(right_list[j])
                j += 1
        elif i == left_len:
            res.append(right_list[j])
            j += 1
        elif j == right_len:
            res.append(left_list[i])
            i += 1
    return res

def merge_sort(x):
    if len(x) <= 1:
        return x

    median_x = len(x) // 2
    left_list = merge_sort(x[:median_x])
    right_list = merge_sort(x[median_x:])
    return merge(left_list, right_list)


if __name__ == "__main__":
    INPUT_FILE = "../txtf/test_1_input"
    OUTPUT_FILE = "../txtf/test_1_output"

    data = read_input(INPUT_FILE)
    if data:
        sorted_data = merge_sort(data)
        write_output(output_file, sorted_data)

print(first_check(input_file_n(INPUT_FILE), operation_with_file(INPUT_FILE)))
```

Объяснение решения: с помощью функции Merge () берутся два отсортированных списка и объединяются в один список: с помощью цикла for на каждом шагу отбирается меньший элемент из двух списков и добавляется в итоговый список. Функция Merge sort () работает по принципу «разделяй и властвуй»: исходный список делится на два подсписка, которые продолжают делиться, пока не будут отсортированы. Дальше начиная с самых маленьких подсписков происходит объединение данных подсписков пока не будет достигнут итоговый отсортированный массив.

Входной файл:

main_1.py

≡

test_1_input

×

1

100000

2

100000000

999999999

999999998

999999997

999999996

999999995

999999994

999999988

999999987

999999986

999999985

999999984

999999983

999999982

999999976

999999975

999999974

999999973

999999972

999999971

999999970

999999964

999999963

999999962

999999961

999999960

999999959

999999958

999999952

999999951

999999950

999999949

999999948

999999947

999999946

999999940

999999939

999999938

999999937

999999936

999999935

999999934

999999928

999999927

999999926

999999925

999999924

999999923

999999922

999999916

999999915

999999914

999999913

999999912

999999911

999999910

999999904

999999903

999999902

999999901

999999900

999999899

999999898

999999892

999999891

999999890

999999889

999999888

999999887

999999886

999999880

999999879

999999878

999999877

999999876

999999875

999999874

999999868

999999867

999999866

999999865

999999864

999999863

999999862

999999856

999999855

999999854

999999853

999999852

999999851

999999850

999999844

999999843

999999842

999999841

999999840

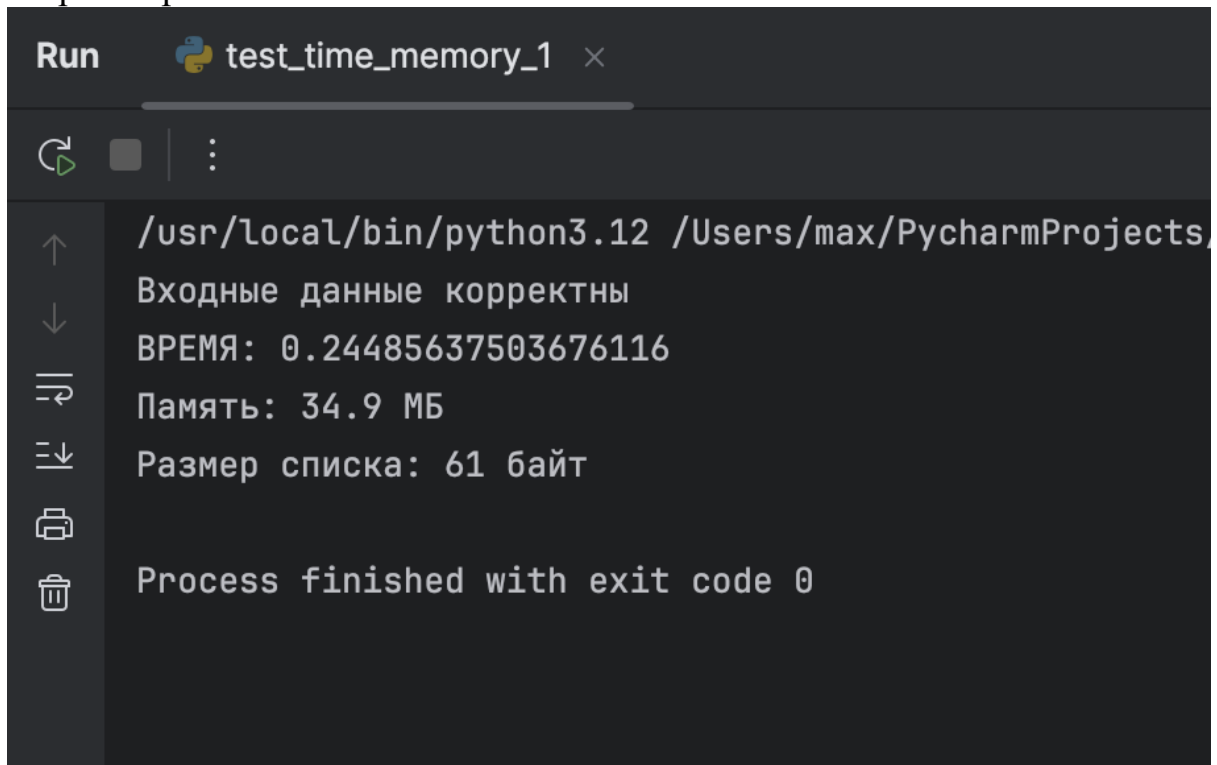
999999839

999999838

Выходной файл:

main_1.py		test_1_input		test_1_output			
1	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						

Затраты времени и памяти:



The image shows a PyCharm Run console window. The title bar reads 'Run' followed by a Python icon and the file name 'test_time_memory_1'. The console output is as follows:

```
/usr/local/bin/python3.12 /Users/max/PycharmProjects/  
Входные данные корректны  
ВРЕМЯ: 0.24485637503676116  
Память: 34.9 МБ  
Размер списка: 61 байт  
  
Process finished with exit code 0
```

On the left side of the console, there is a vertical toolbar with icons for: running (green play button), pausing (square), a menu (three dots), navigating (up/down arrows), undo/redo (curved arrows), stepping (arrows with underlines), printing (printer icon), and deleting (trash icon).

Задание 3

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.

Решение:

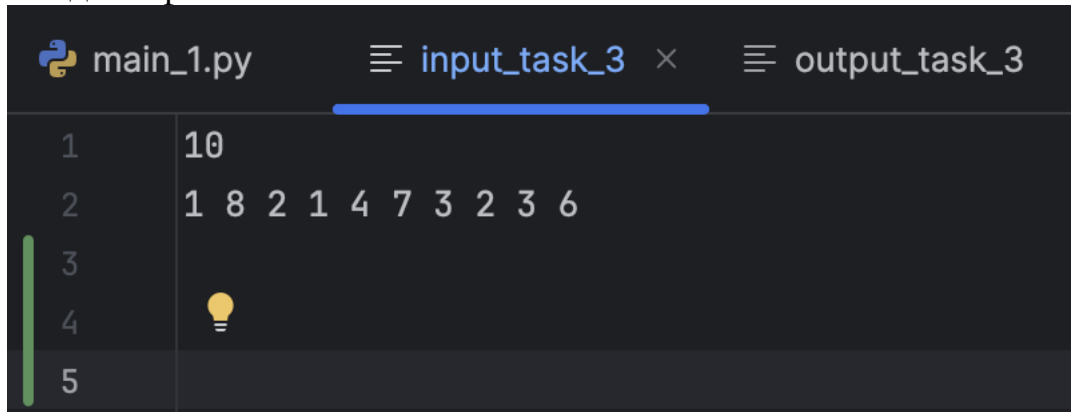
```
from lab_2.utils import *

def inverse(arr):
    count = 0
    n = len(arr)
    for i in range(n):
        for j in range(i + 1, n):
            if arr[i] > arr[j]:
                count += 1
    return count

if __name__ == "__main__":
    FILE_INPUT = "../txtf/input_task_3"
    FILE_OUTPUT = "../txtf/output_task_3"
    file = operation_with_file(FILE_INPUT)
    res = inverse(file)
    check_list =
check(input_file_n(FILE_INPUT), operation_with_file(FILE_INPUT))
    if check_list:
        output = output_file(FILE_OUTPUT, str(res))
        print("Входные данные корректны")
    else:
        output_file(FILE_OUTPUT, "Ошибка входных данных")
        print("Ошибка входных данных")
```

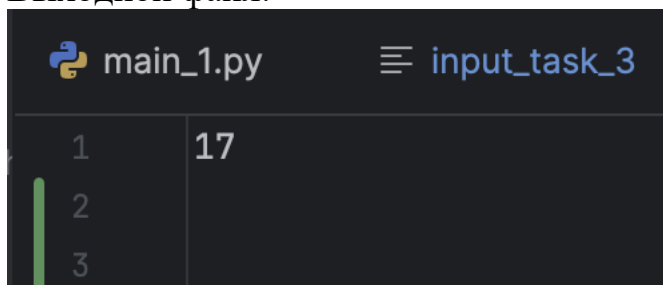
Объяснение решения: с помощью функции `inverse()` мы перебираем пары элементов списка, поступающего на вход как аргумент функции. Если первый элемент пары больше второго и индекс первого элемента пары меньше индекса второго, то мы увеличиваем счетчик на единицу, в результате возвращаем его.

Входной файл:



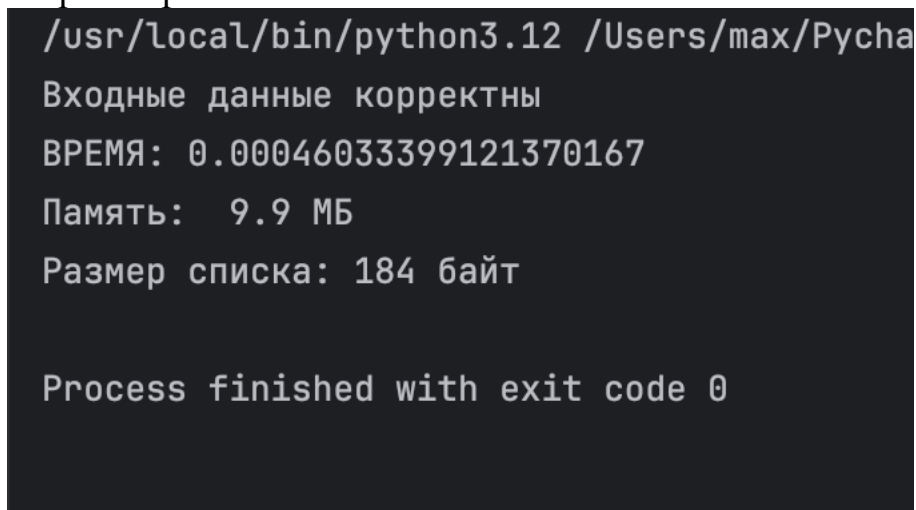
```
main_1.py  input_task_3  output_task_3
1 10
2 1 8 2 1 4 7 3 2 3 6
3
4
5
```

Выходной файл:



```
main_1.py  input_task_3
1 17
2
3
```

Затраты времени и памяти:



```
/usr/local/bin/python3.12 /Users/max/Pycha
Входные данные корректны
ВРЕМЯ: 0.00046033399121370167
Память: 9.9 МБ
Размер списка: 184 байт

Process finished with exit code 0
```


Задание 4

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.

Решение:

```
from lab_2.utils import *

from lab_2.utils import
second_file_operation_list1, second_file_operation_list2, second_check
, second_file_operation_line1, second_file_operation_line3,
output_file, string

def binary_search(arr, target):
    left, right = 0, len(arr) - 1
    while left <= right:
        mid = (left + right) // 2
        if arr[mid] == target:
            return mid
        elif arr[mid] < target:
            left = mid + 1
        else:
            right = mid - 1
    return -1

def answer(a, b):
    return string([binary_search(a, index) for index in b])

if __name__=="__main__":
    FILE_INPUT = "../txtf/input_task_4"
    FILE_OUTPUT = "../txtf/output_task_4"
    file_1 = second_file_operation_list1(FILE_INPUT)
    file_2 = second_file_operation_list2(FILE_INPUT)
    line1 = second_file_operation_line1(FILE_INPUT)
    line3 = second_file_operation_line3(FILE_INPUT)
    check = second_check(line1, line3, file_1, file_2)
    if check:
        results = answer(file_1, file_2)
        file_o = output_file(FILE_OUTPUT, results)
        print("Входные данные корректны")
    else:
        file_o = output_file(FILE_OUTPUT, "Ошибка входных данных")
        print("Ошибка входных данных")
```

Объяснение решения:

Данный код реализует поиск индексов элементов из массива `b` в массиве `a` с помощью бинарного поиска. Программа читает два массива из входного файла, используя функции из `lab_2. utils`, проверяет корректность данных и, если они валидны, выполняет поиск индексов каждого элемента из второго массива в первом. Результаты записываются в строку и сохраняются в выходной файл. Если данные некорректны, в файл записывается сообщение об ошибке. Алгоритм бинарного поиска используется, так как массив `a` предполагается отсортированным, обеспечивая эффективность для каждого элемента.

Входной файл:

input_task_4		output_task_4	
1	5		
2	1 5 8 12 13		
3	5		
4	8 1 23 1 11		
5	💡		

Выходной файл:

input_task_4		output_task_4	
1	2 0 -1 0 -1		
2			
3			

Затраты времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/PycharmI
Входные данные корректны
ВРЕМЯ: 0.000189124999451451
Память: 9.6 МБ
Размер списка №1: 120 байт
Размер списка №2: 120 байт

Process finished with exit code 0
```

Задание 5

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 как 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 $O(n \log n)$.

- **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 положительных целых чисел, по модулю не превосходящих 10^9 , $0 \leq a_i \leq 10^9$.
- **Формат выходного файла (output.txt).** Выведите 1, если во входной последовательности есть элемент, который встречается строго больше половины раз; в противном случае - 0.
- Ограничение по времени. 2сек.
- Ограничение по памяти. 256 мб.

- Пример 1:

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

Число "2" встречается больше $5/2$ раз.

- Пример 2:

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

Нет элемента, встречающегося больше $n/2$ раз.

Решение:

```
from lab_2.utils import *

def majority(a):
    for i in range(len(a)):
        current_element = a[i]
        count = 0
        for j in range(len(a)):
            if a[j] == current_element:
                count += 1
        if count > (len(a)/2):
            return 1
    return 0

if __name__=="__main__":
    FILE_INPUT = "../txtf/input_task_5"
    FILE_OUTPUT = "../txtf/output_task_5"
    file = operation_with_file(FILE_INPUT)
    check_f = check(input_file_n(FILE_INPUT),file)
    if check_f:
        res = majority(file)
        file_o = output_file(FILE_INPUT,str(res))
        print("Входные данные корректны")
    else:
        output_file(FILE_OUTPUT, "Ошибка входных данных")
        print("Ошибка входных данных")
```

Объяснение решения:

Этот код ищет элемент, который встречается более половины раз от общей длины массива. Выполняется проверка каждого элемента массива: для каждого из них подсчитывается, сколько раз он встречается в массиве. Данные для анализа считываются из входного файла, их корректность проверяется с помощью функций из модуля `lab_2.utils`. В случае успешной проверки строковое значение единицы или нуля записывается в выходной файл. Если входные данные некорректны — в выходной файл указывается сообщение об ошибке.

Входной файл:

test_time_memory_5.py	
1	4
2	1 2 3 4
3	

Выходной файл:

test_time_memory_5.py	
1	0
2	
3	

Затраты времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/Pycharm
Входные данные корректны
ВРЕМЯ: 0.0004195420042378828
Память: 10.1 МБ
Размер списка: 88 байт

Process finished with exit code 0
```

Задание 6

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

- **Формат входного файла** в данном случае на ваше усмотрение.
- **Формат выходного файла (output.txt).** Выведите название фирмы, рассматриваемый вами срок изменения акций, дату покупки и дату продажи единицы акции, чтобы получилась максимальная выгода; и сумма этой прибыли.

Решение:

```
def max_subarray(prices, dates):
    if not prices or len(prices) != len(dates):
        return [0, 0, 0, None, None]

    min_price = float('inf')
    min_price_index = -1
    max_profit = 0

    for i in range(len(prices)):
        if prices[i] < min_price:
            min_price = prices[i]
            min_price_index = i

        potential_profit = prices[i] - min_price
        if potential_profit > max_profit and i > min_price_index :
            max_profit = potential_profit
            best_buy_index = min_price_index
            best_sell_index = i

    if max_profit > 0 and best_buy_index != -1 and best_sell_index !=
-1:
        best_buy_price = prices[best_buy_index]
        best_sell_price = prices[best_sell_index]
        best_buy_date = dates[best_buy_index]
        best_sell_date = dates[best_sell_index]
        return [best_buy_price, best_sell_price, max_profit,
best_buy_date, best_sell_date]
    else:
        return [0, 0, 0, None, None]

if __name__=="__main__":
    FILE_INPUT = "../txtf/input_task_6"
    FILE_OUTPUT = "../txtf/output_task_6"
    max_subarray(FILE_INPUT, FILE_OUTPUT)
```

Объяснение решения:

Этот код рассчитывает наибольшую возможную прибыль от покупки и продажи акций, основываясь на списке цен и соответствующих дат. Программа находит такие значения, чтобы покупка происходила по минимальной цене, а продажа — по максимальной цене, следующей за этой минимальной. Результат возвращается в виде цены покупки, цены продажи, прибыли, даты покупки, даты продажи.

Входной файл:

	≡ input_task_6 ×	≡ output_task_6
1	Date	Price
2	27.09.2024	4024.0
3	30.09.2024	4007.5
4	01.10.2024	3963.0
5	02.10.2024	3907.5
6	03.10.2024	3969.0
7	04.10.2024	3945.5
8	07.10.2024	3928.5
9	08.10.2024	3982.5
10	09.10.2024	3933.5
11	10.10.2024	3979.5
12	11.10.2024	3975.0
13	14.10.2024	4074.0
14	15.10.2024	4091.0
15	16.10.2024	4057.0
16	17.10.2024	3983.5
17	18.10.2024	3968.5
18	21.10.2024	4031.0
19	22.10.2024	3964.5
20	23.10.2024	3883.5
21	24.10.2024	3969.5
22	25.10.2024	3834.5

Выходной файл:

```
input_task_6  output_task_6  test_time_memory_6.py
1  Yandex
2  Change period: 1 month
3  Date of purchase: 02.10.2024 Purchase price: 3907.5
4  Date of sale: 15.10.2024 Sale price: 4091.0
5  Profit: 183.5
```

Затраты времени и памяти:

```
/usr/local/bin/python3.12 /Users/max/Pycharm
ВРЕМЯ: 0.00028433398983906955
Память: 9.6 МБ

Process finished with exit code 0
```

Задание 9

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.

```
Matrix_Multiply(X, Y)::
  n = X.rows
  Z - квадратная матрица размера n
  for i = 1 to n:
    for j = 1 to n:
      z[i,j] = 0
      for k = 1 to n:
        z[i,j] = z[i,j] + x[i,k]*y[k,j]
  return Z
```

Задачу умножения матриц достаточно легко разбить на подзадачи, поскольку произведение можно составлять из *блоков*. Разобьём каждую из матриц X и Y на четыре блока размера $\frac{n}{2} \times \frac{n}{2}$:

$$X = \begin{bmatrix} A & B \\ C & D \end{bmatrix}, Y = \begin{bmatrix} E & F \\ G & H \end{bmatrix},$$

Тогда их произведение выражается в терминах этих блоков по обычной формуле умножения матриц:

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \cdot \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

Вычислив рекурсивно восемь произведений $AE, BG, AF, BH, CE, DG, CF, DH$ и просуммировав их за время $O(n^2)$, мы вычислим необходимое нам произведение матриц. Соответствующее рекуррентное соотношение на время работы алгоритма

$$T(n) = 8T\left(\frac{n}{2}\right) + O(n^2).$$

Какое получилось время у предыдущего рекурсивного алгоритма? Да, ничуть не лучше наивного. Однако его можно ускорить с помощью алгебраического трюка: для вычисления произведения XY достаточно перемножить *семь* пар матриц размера $\frac{n}{2} \times \frac{n}{2}$, после чего хитрым образом (и как только Штрассен догадался?) получить ответ:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

где

$$\begin{aligned} P_1 &= A(F - H), & P_5 &= (A + D)(E + H), \\ P_2 &= (A + B)H, & P_6 &= (B - D)(G + H), \\ P_3 &= (C + D)E, & P_7 &= (A - C)(E + F), \\ P_4 &= D(G - E). \end{aligned}$$

- **Цель.** Применить метод Штрассена для умножения матриц и сравнить его с простым методом. *Найти размер матриц n , при котором метод Штрассена работает существенно быстрее простого метода.*
- **Формат входа.** Стандартный ввод или input.txt. Первая строка - размер квадратных матриц n для умножения. Следующие строки соответственно сами значения матриц A и B .
- **Формат выхода.** Стандартный вывод или output.txt. Матрица $C = A \cdot B$.

Решение:
Multiply:

```
def multiply(n, X, Y):
    Z = [[0]*n for _ in range(n)]
    for i in range(n):
        for j in range(n):
            for k in range(n):
                Z[i][j] += X[i][k]
                Z[i][j] += Y[k][j]
    return Z

def check(file_i, file_o):
    with open(file_i, 'r') as f:
        n = int(f.readline())
        A = [list(map(int, f.readline().split())) for k in range(n)]
        B = [list(map(int, f.readline().split())) for k in range(n)]

        C = multiply(n, A, B)

        with open(file_o, 'w') as f:
            for res1 in C:
                f.write(' '.join(map(str, res1)) + '\n')

if __name__ == "__main__":
    FILE_INPUT = "../txtf/input_task_9"
    FILE_OUTPUT = "../txtf/output_task_9"
    check(FILE_INPUT, FILE_OUTPUT)
```

Strassen:

THRESHOLD = 16

```
def split(M):
    n = len(M) // 2
    return (
        [[M[i][j] for j in range(n)] for i in range(n)], # A11
        [[M[i][j] for j in range(n, n * 2)] for i in range(n)], #
A12
        [[M[i][j] for j in range(n)] for i in range(n, n * 2)], #
A21
        [[M[i][j] for j in range(n, n * 2)] for i in range(n, n *
2)], # A22
    )

def merge(c11, c12, c21, c22):
    n = len(c11)
    C = [[0 for _ in range(n * 2)] for _ in range(n * 2)]

    for i in range(n):
        for j in range(n):
            C[i][j] = c11[i][j]
            C[i][j + n] = c12[i][j]
            C[i + n][j] = c21[i][j]
            C[i + n][j + n] = c22[i][j]

    return C

def strassen(A, B):
    n = len(A)
    if n <= THRESHOLD:
        return [[sum(A[i][k] * B[k][j] for k in range(n)) for j in
range(n)] for i in range(n)]
    else:
        A11, A12, A21, A22 = split(A)
        B11, B12, B21, B22 = split(B)

        P1 = strassen(A11 + A22, B11 + B22)
        P2 = strassen(A21 + A22, B11)
        P3 = strassen(A11, B12 - B22)
        P4 = strassen(A22, B21 - B11)
        P5 = strassen(A11 + A12, B22)
        P6 = strassen(A21 - A11, B11 + B12)
        P7 = strassen(A12 - A22, B21 + B22)

        C11 = P1 + P4 - P5 + P7
        C12 = P3 + P5
        C21 = P2 + P4
        C22 = P1 - P2 + P3 + P6

        return merge(C11, C12, C21, C22)

def check(file_inp, file_out):
    with open(file_inp, 'r') as f:
        n = int(f.readline())
        A = [list(map(int, f.readline().split())) for k in range(n)]
```

```

B = [list(map(int, f.readline().split())) for k in range(n)]
C = strassen(A, B)

with open(file_out, 'w') as f:
    for res1 in C:
        f.write(' '.join(map(str, res1)) + '\n')

if __name__=="__main__":
    check("../txtf/input_task_9", "../txtf/output_task_9")

```

Объяснение решения:

Этот код выполняет перемножение квадратных матриц с использованием двух подходов: стандартного метода и ускоренного алгоритма Штрассена. Если размер матрицы меньше порога `THRESHOLD`, применяется обычный метод, иначе используется рекурсивный алгоритм Штрассена. Для больших матриц он делит их на 4 подматрицы, вычисляет промежуточные результаты, а затем объединяет их в результирующую матрицу. Чтение данных (размер и значения матриц) производится из входного файла, после чего результат записи сохраняется в выходной файл.

Входной файл:

input_task_9 ×		output_task_9	
1	3		
2	1 2 7		
3	3 4 5		
4	9 10 11		
5	16 8 3		
6	6 9 2		
7	8 3 1		

Выходные файлы:

Multiply:

input_task_9		output_task_9 ×	
1	40 30 16		
2	42 32 18		
3	60 50 36		

Strassen:

input_task_9		output_task_9	
1	84 47 14		
2	112 75 22		
3	292 195 58		

Затраты времени и памяти:

Multiply:

```
/usr/local/bin/python3.12 /Users/max/Pycharm
ВРЕМЯ: 0.0002573749952716753
Память: 9.5 МБ

Process finished with exit code 0
```

Strassen:

```
/usr/local/bin/python3.12 /Users/max/Pycharm
ВРЕМЯ: 0.0003852079971693456
Память: 10.4 МБ

Process finished with exit code 0
```

Вывод

В данной лабораторной работе познакомился и поработал с новыми алгоритмами сортировки и поиска, а также решил несколько интересных задач: про акции и матрицы.