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.

Выполнила:

Просветова Валерия

К3141

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024г.

# Содержание отчета

Задача №1 - Сортировка слияния

Задача №3 - Число инверсий

Задача №4 - Бинарный поиск

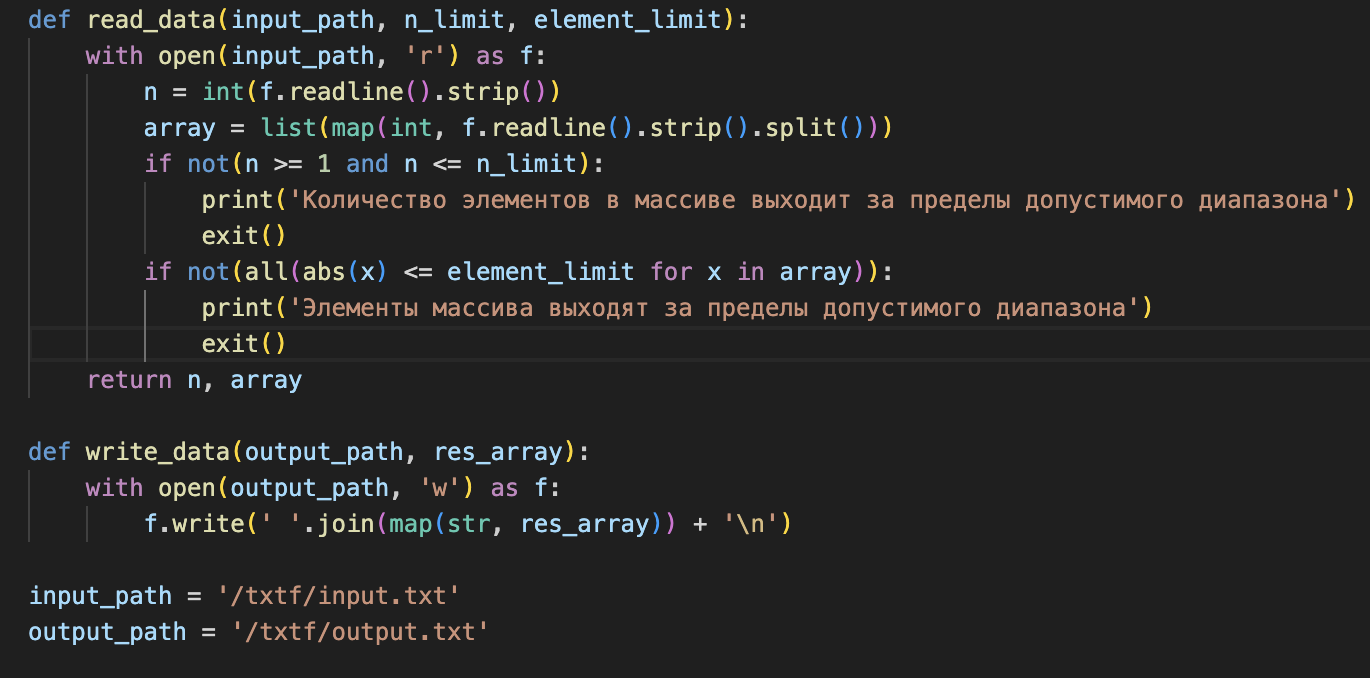
Задача №5 - Представитель большинства

Задача №6 - Поиск максимальной прибыли

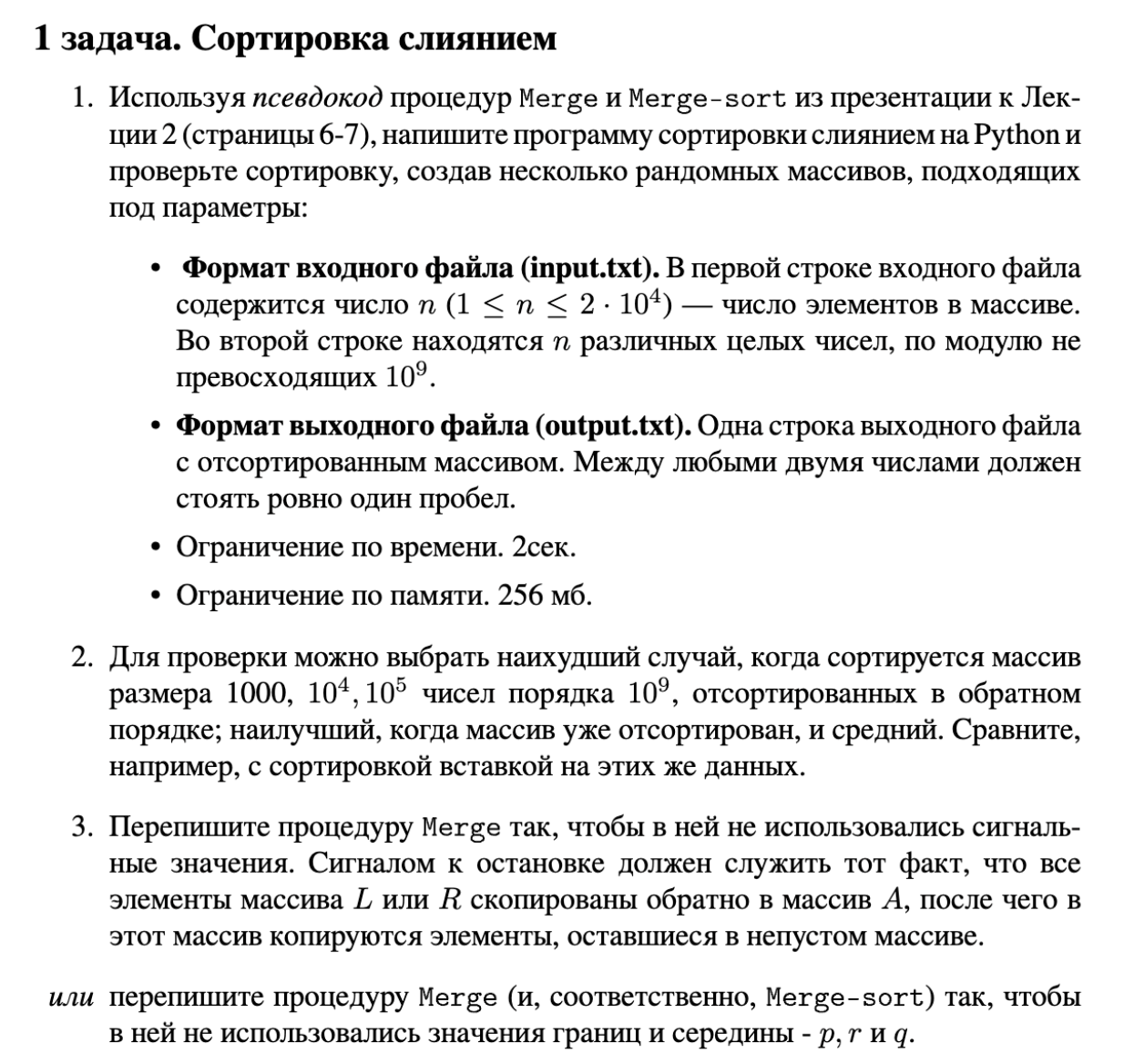
Задача №7 - Поиск максимального подмассива за линейное время

**Вывод**

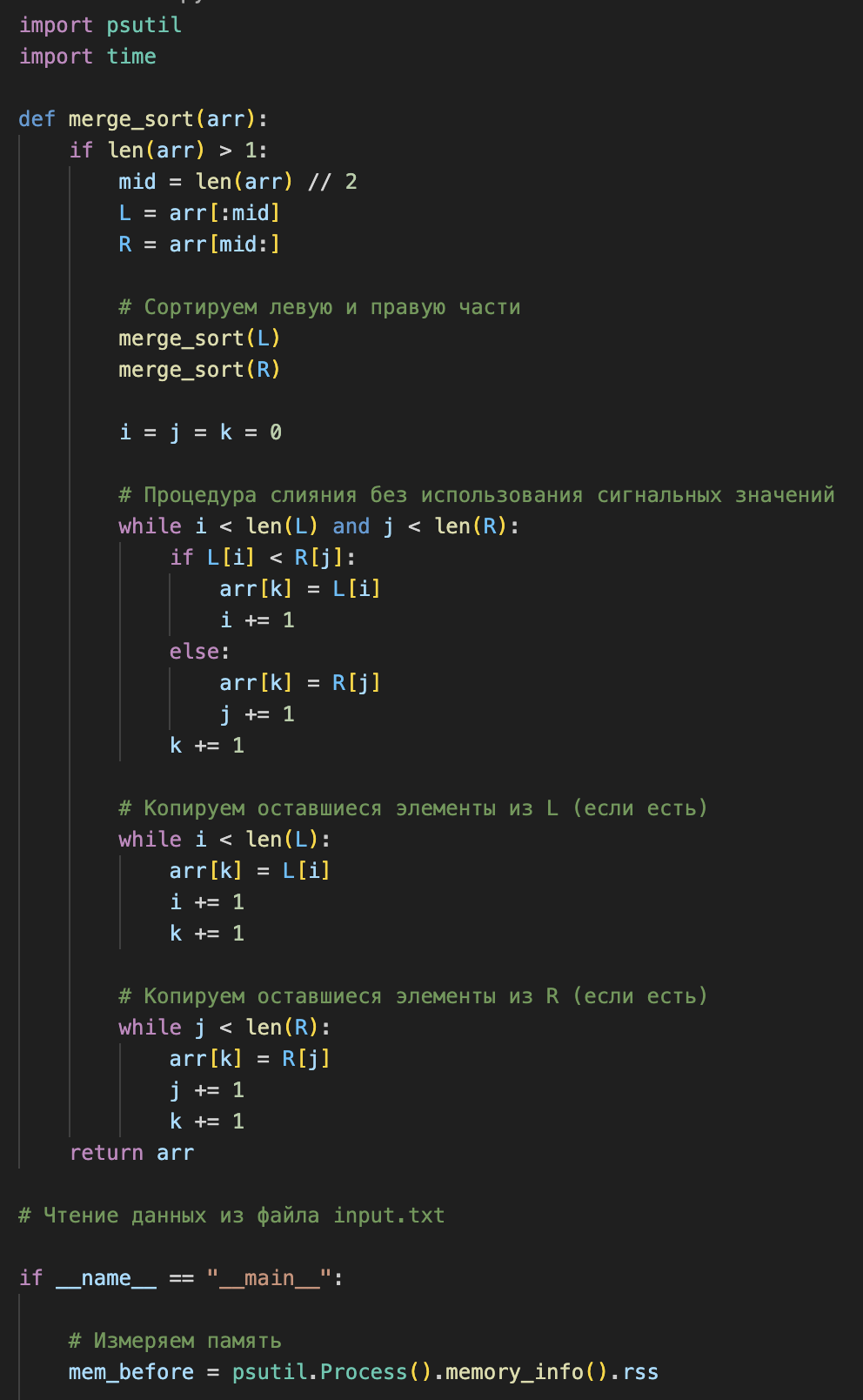
Для автоматизации кодов я использовала utils.py

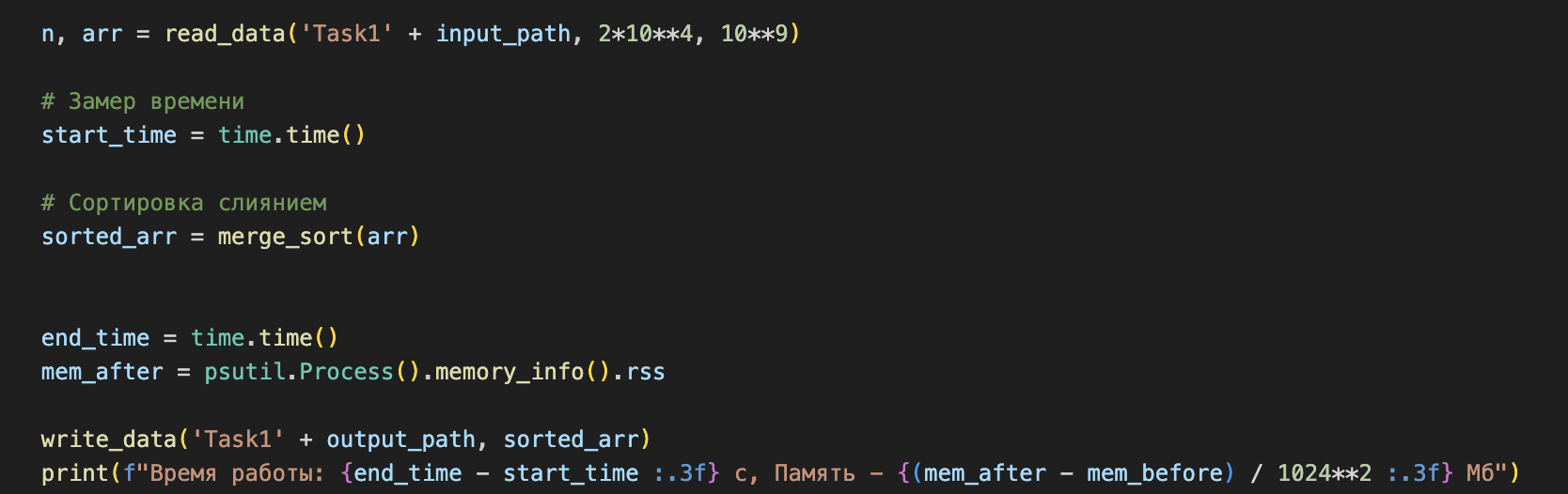


Задача №1:

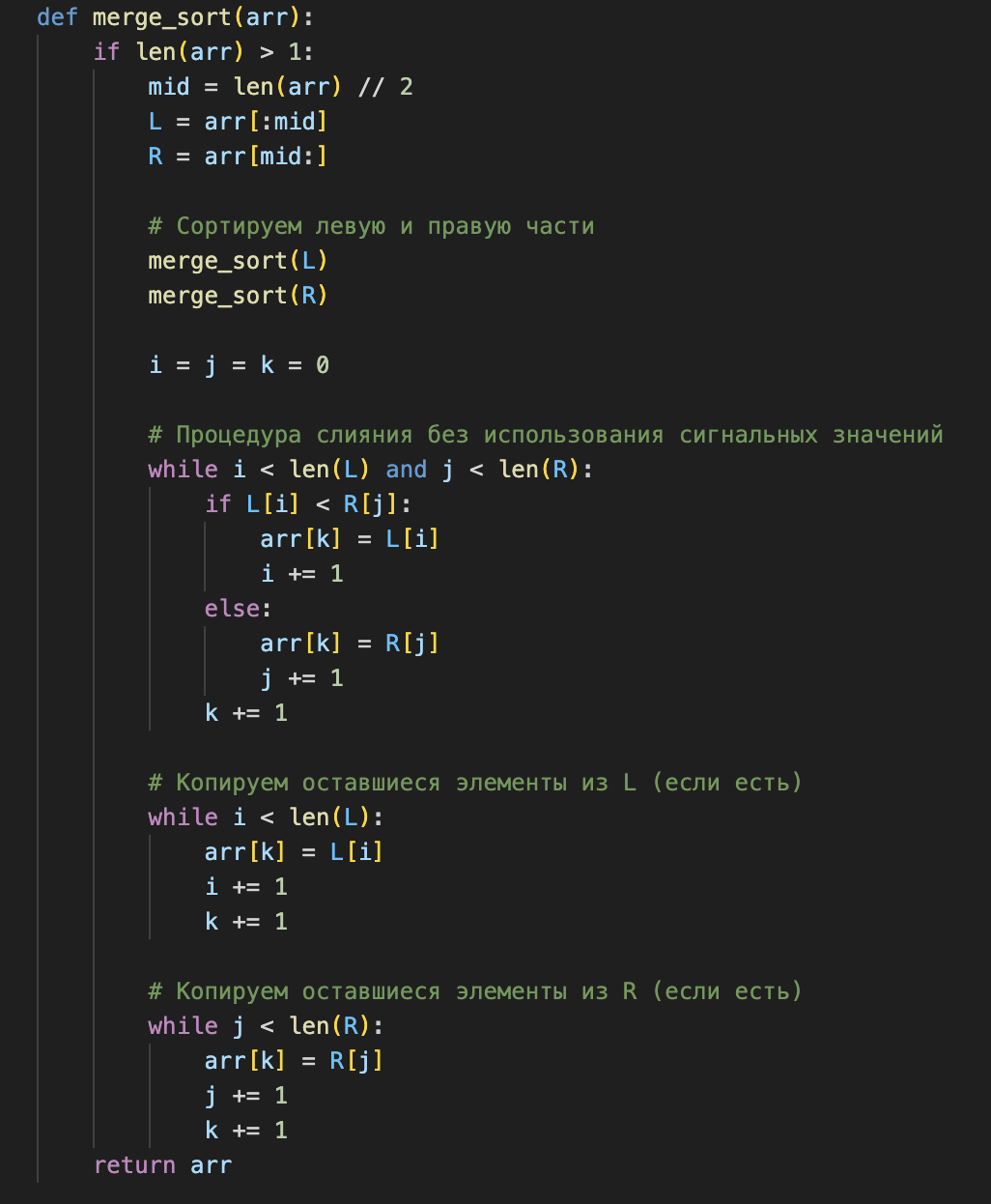


Решение №1 :





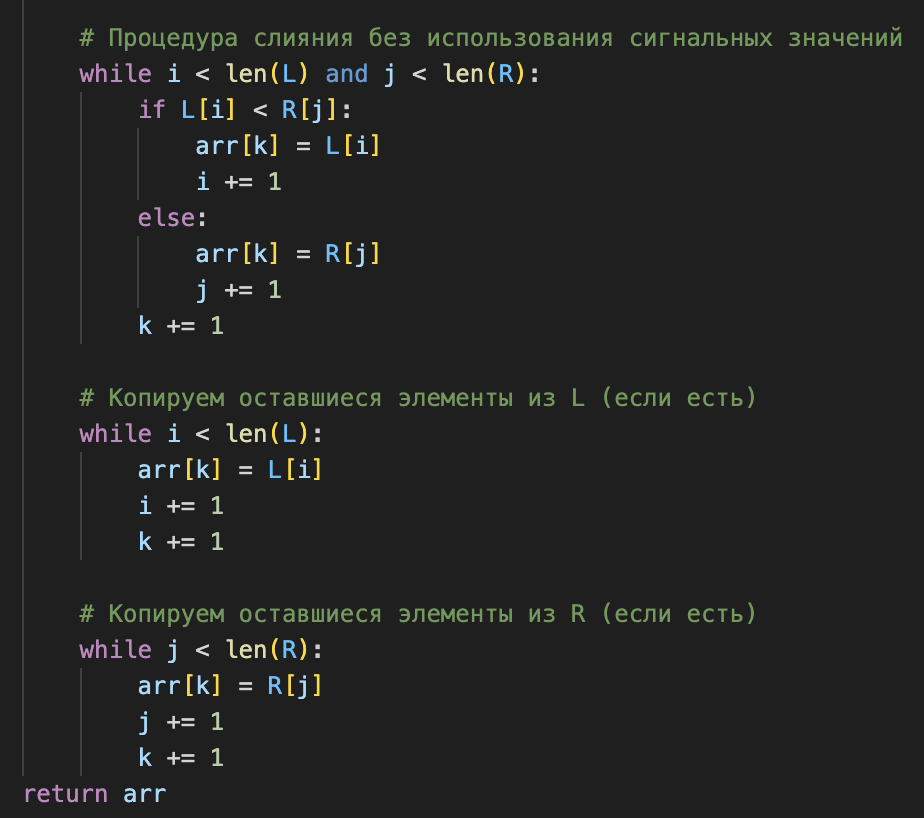
Данная функция merge\_sort реализует алгоритм сортировки слиянием для сортировки массива целых чисел.



* Функция проверяет, что длина массива arr больше 1.
* Если длина массива равна 1 или меньше, функция возвращает исходный массив, поскольку он уже отсортирован.
* Функция находит середину массива mid путем целочисленного деления длины массива на 2.
* Функция создает две части массива: L (левая часть) и R (правая часть).
* L содержит элементы массива от начала до середины (arr[:mid]).
* R содержит элементы массива от середины до конца (arr[mid:]).
* Функция вызывает себя рекурсивно для сортировки левой части L.
* Функция вызывает себя рекурсивно для сортировки правой части R.

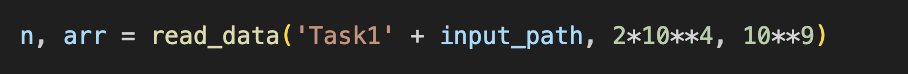
Функция инициализирует три индекса: i, j и k.

i используется для обхода левой части L. j используется для обхода правой части R.k используется для обхода исходного массива arr.



* Функция выполняет процедуру слияния левой и правой частей.
* Цикл продолжается, пока не будут обработаны все элементы левой и правой частей.
* Если элемент левой части L[i] меньше элемента правой части R[j], функция копирует элемент левой части в исходный массив arr[k] и инкрементирует индекс i.
* Если элемент правой части R[j] меньше или равен элементу левой части L[i], функция копирует элемент правой части в исходный массив arr[k] и инкрементирует индекс j.
* Индекс k инкрементируется после копирования каждого элемента.
* Если остались элементы в левой части L, функция копирует их в исходный массив arr.
* Если остались элементы в правой части R, функция копирует их в исходный массив arr.
* Функция возвращает отсортированный массив arr.
* Затем полученный результат записываю в output.txt.Также произвожу подсчет времени и памяти.

Считываем данные файла с помощью utils и записываем результат:



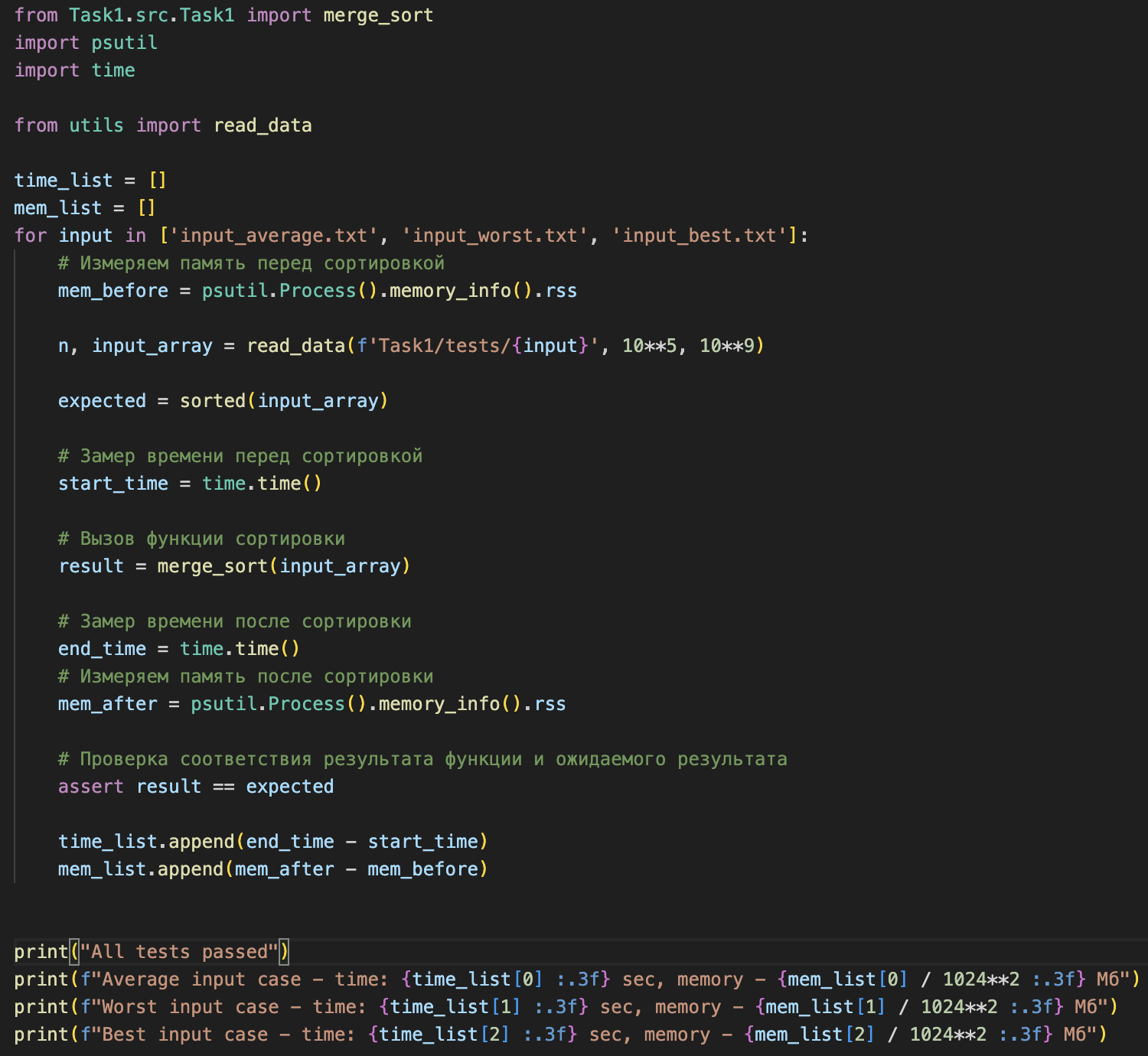


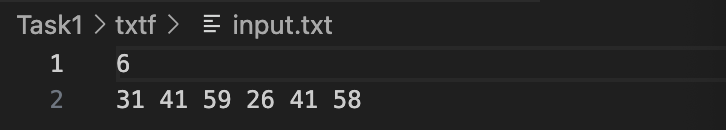
Произвожу подсчет времени:

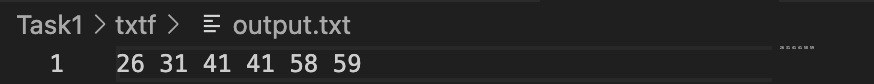




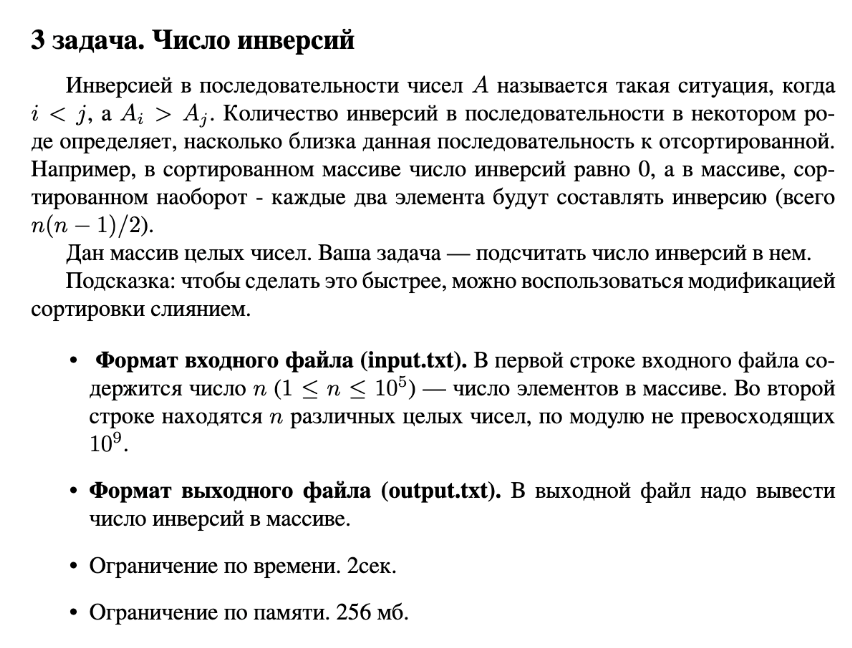
Тест для проверки кода(с учетом на худший, средний и лучший случаи):

  
  
Файлы input.txt и output.txt :

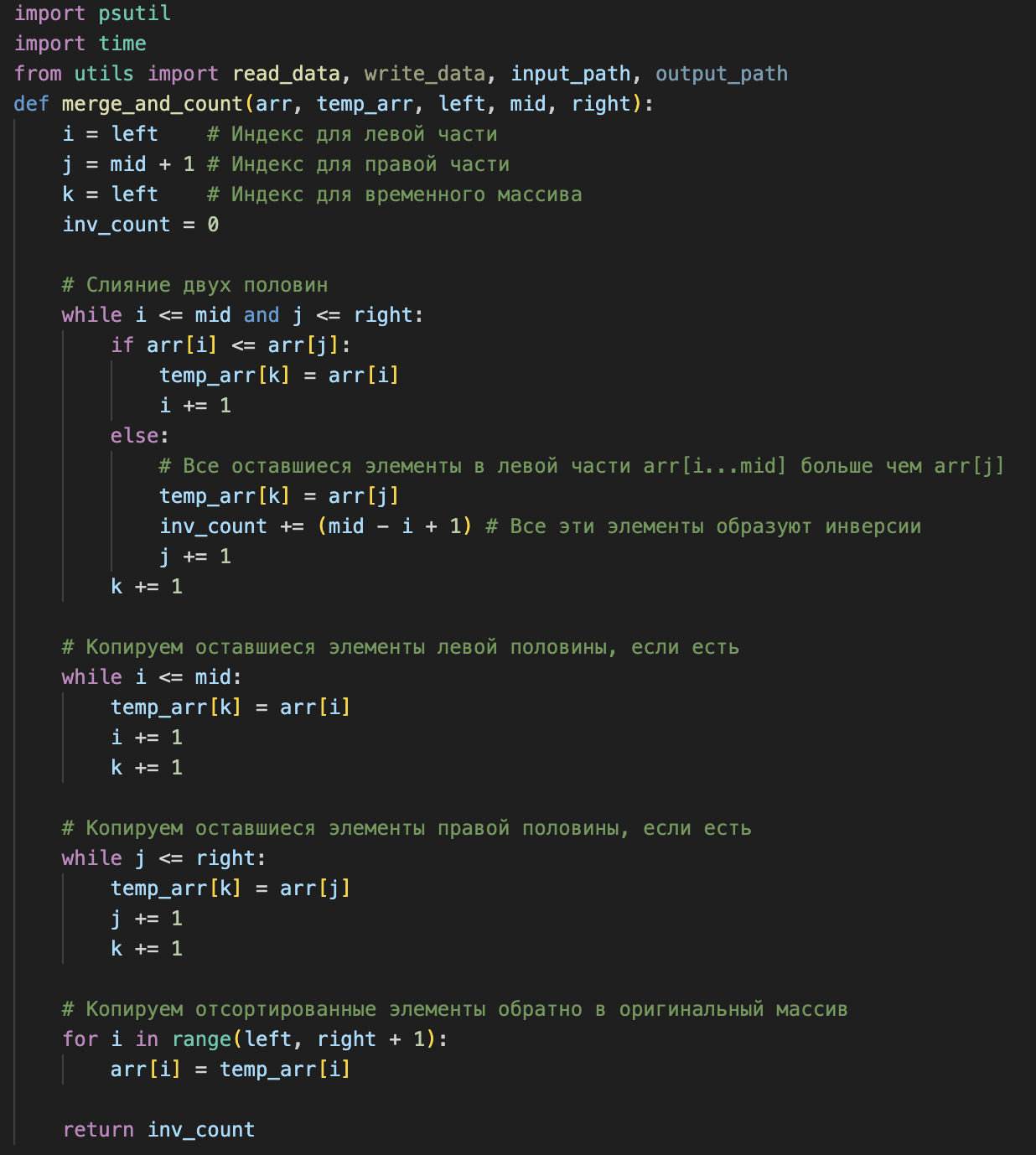




Задача №3



Решение №3:





* Функция merge\_and\_count выполняет слияние двух частей массива arr и подсчет инверсий.
* Функция принимает массив arr, временный массив temp\_arr, индексы left, mid, right и возвращает количество инверсий.
* Функция использует алгоритм сортировки слиянием для слияния двух частей массива arr и подсчета инверсий.
* Функция инициализирует индексы i, j и k для обхода левой, правой и временного массивов.
* Функция выполняет цикл слияния двух частей массива arr.
* Если элемент левой части arr[i] меньше или равен элементу правой части arr[j], функция копирует элемент в временный массив temp\_arr и инкрементирует индекс i.
* Если элемент правой части arr[j] меньше элемента левой части arr[i], функция копирует элемент в временный массив temp\_arr, увеличивает счетчик инверсий и инкрементирует индекс j.
* После завершения цикла слияния функция копирует отсортированные элементы обратно в оригинальный массив arr.
* Функция возвращает количество найденных инверсий.
* Переменная inv\_count используется для хранения количества инверсий в массиве.
* Код проверяет, что индекс left меньше индекса right.
* Если это условие не выполняется, функция возвращает количество инверсий inv\_count, поскольку массив уже отсортирован.
* Код находит середину массива mid путем целочисленного деления суммы индексов left и right на 2.
* Середина массива используется для разделения массива на две части: левую и правую.

Код вызывает функцию merge\_sort\_and\_count рекурсивно для сортировки левой части массива от индекса left до индекса mid.

Код вызывает функцию merge\_sort\_and\_count рекурсивно для сортировки правой части массива от индекса mid + 1 до индекса right.

Результаты рекурсивных вызовов функции складываются в переменную inv\_count, чтобы получить общее количество инверсий в массиве.

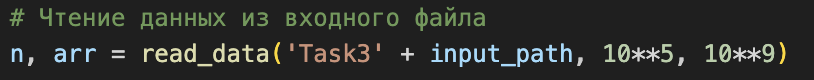
После рекурсивной сортировки левой и правой частей массива, функция merge\_and\_count будет вызвана для слияния двух частей и подсчета инверсий.

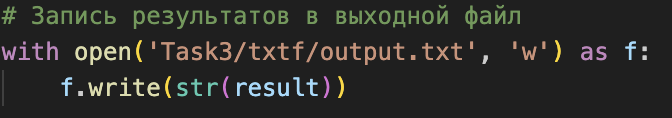
* Затем полученный результат записываю в output.txt.Также произвожу подсчет времени и памяти.





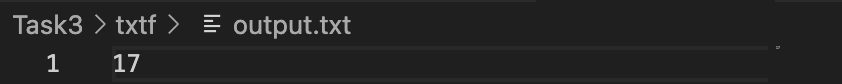
Считываем данные файла с помощью utils и записываем результат:



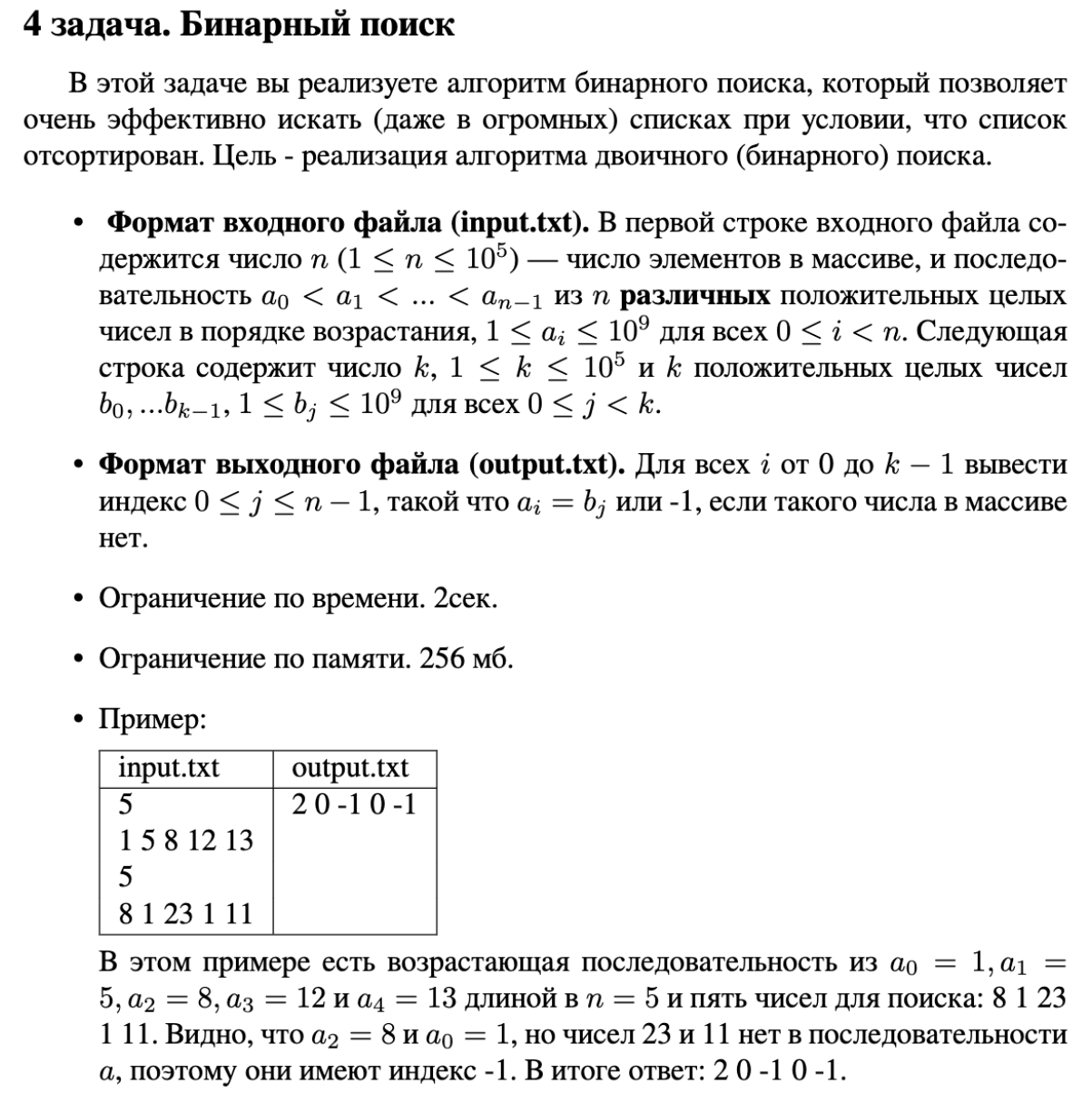


Файлы input.txt и output.txt :

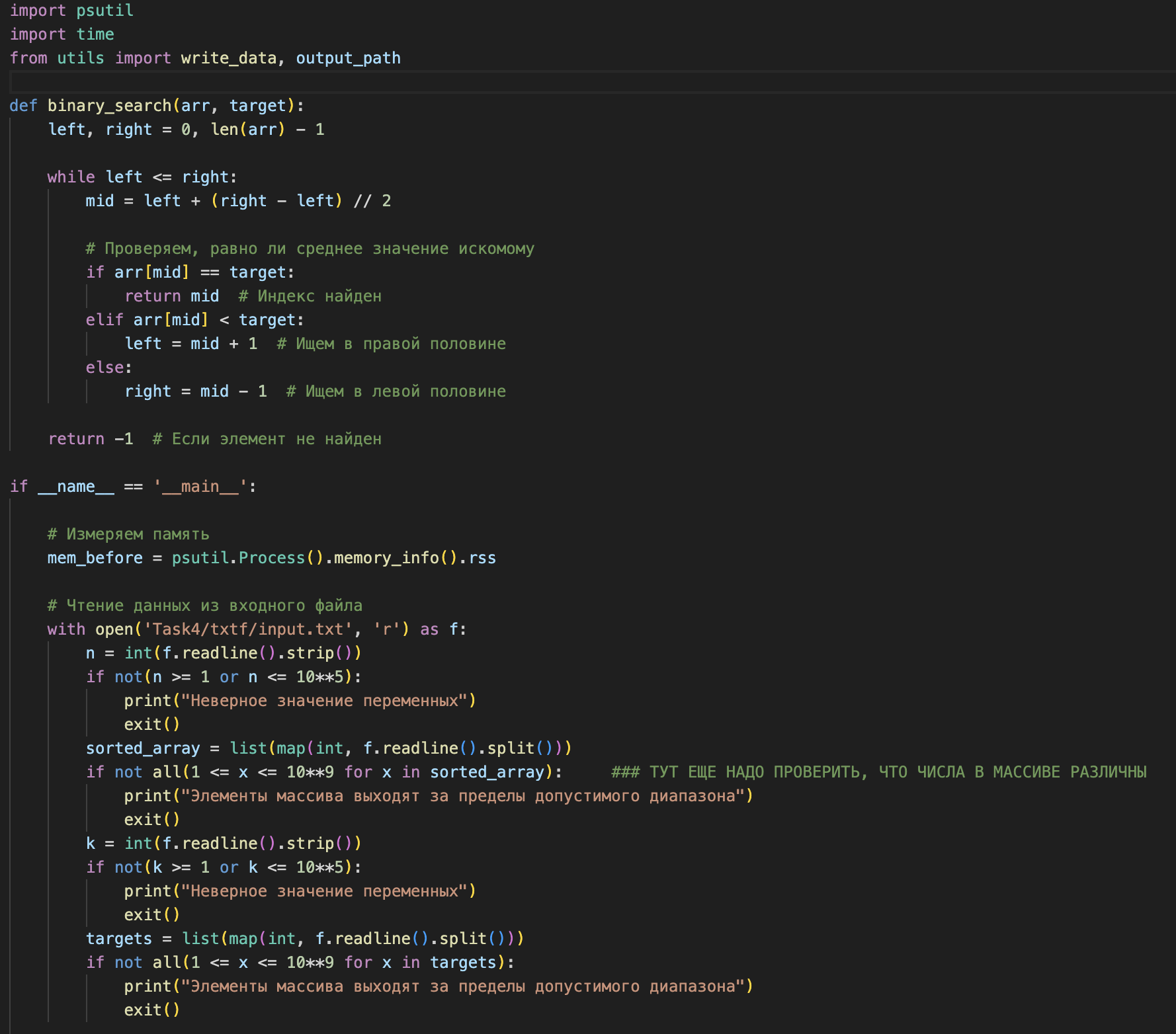


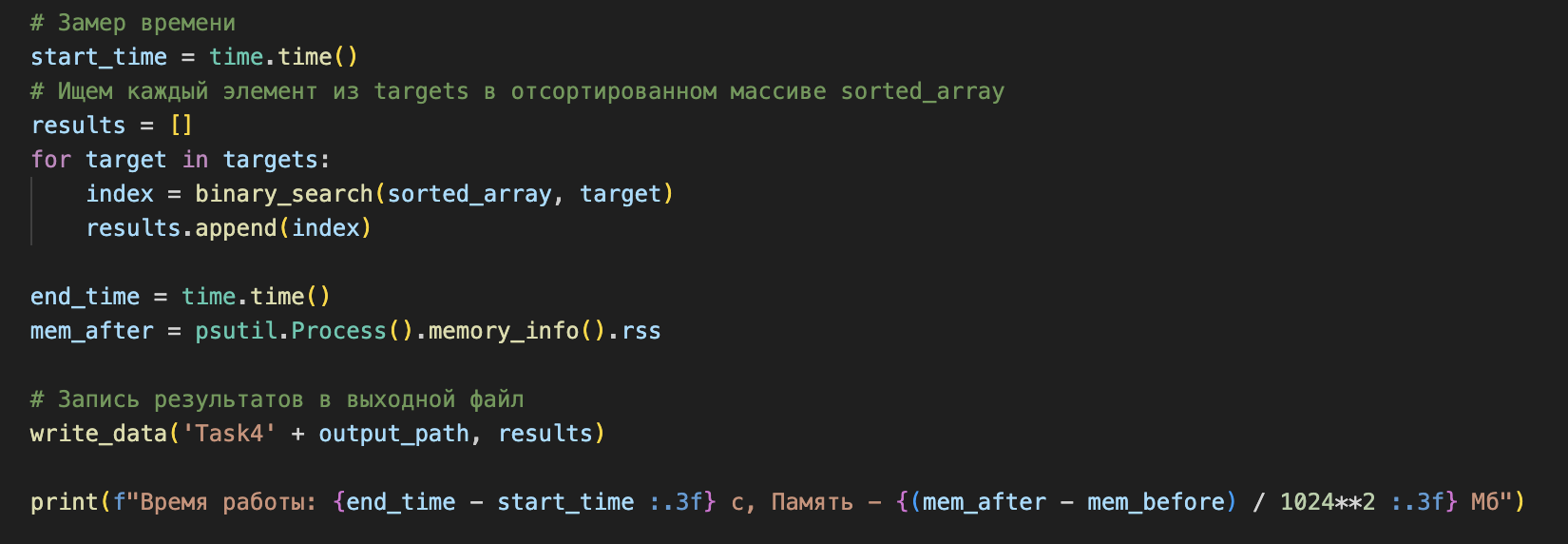


Задача№4



Решение №4:

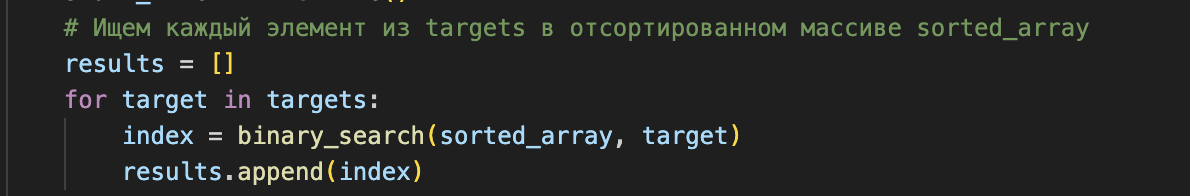




Данная функция binary\_search реализует алгоритм бинарного поиска для поиска элемента в отсортированном массиве.

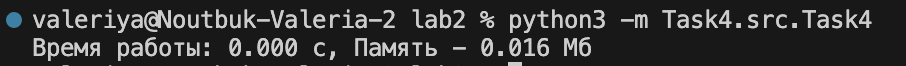
* Переменные left и right используются для хранения индексов начала и конца поискового диапазона в массиве.
* Переменная left инициализируется значением 0, что соответствует началу массива.
* Переменная right инициализируется значением len(arr) - 1, что соответствует концу массива.
* Цикл продолжается, пока поисковый диапазон не станет пустым, то есть пока left не станет больше right.
* В каждой итерации цикла вычисляется средний индекс mid поискового диапазона.
* Средний индекс рассчитывается как среднее значение индексов left и right.
* Если значение элемента в среднем индексе mid равно искомому значению target, функция возвращает индекс mid, указывающий на найденный элемент.
* Если значение элемента в среднем индексе mid меньше искомого значения target, поисковый диапазон обновляется, чтобы искать в правой половине массива. Индекс left устанавливается в значение mid + 1.
* Если значение элемента в среднем индексе mid больше искомого значения target, поисковый диапазон обновляется, чтобы искать в левой половине массива. Индекс right устанавливается в значение mid - 1.

Если цикл бинарного поиска завершается без нахождения искомого значения, функция возвращает значение -1, указывающее на то, что элемент не найден.



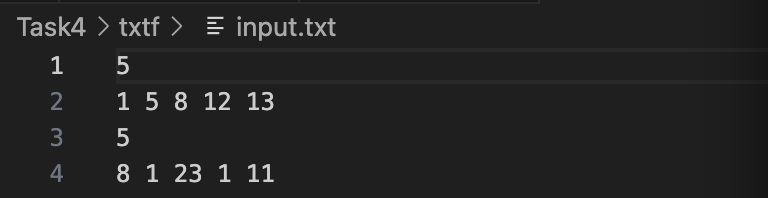
* Функция бинарного поиска используется для поиска нескольких целевых значений targets в отсортированном массиве sorted\_array.
* Для каждого целевого значения target вызывается функция binary\_search с массивом sorted\_array и значением target.
* Результат поиска, который представляет собой индекс найденного элемента или значение -1, если элемент не найден, добавляется в список results.
* Затем полученный результат записываю в output.txt.Также произвожу подсчет времени и памяти.

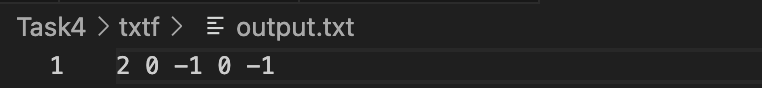




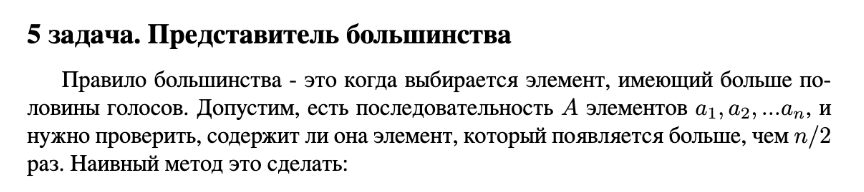
Чтение данных происходит обычным способом, а вот записываем с помощью utils

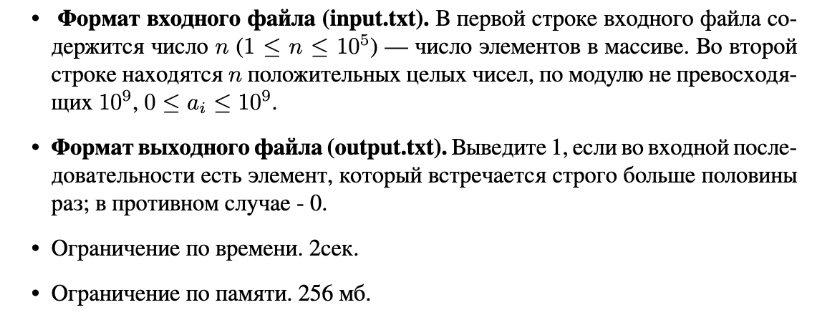
Файлы input.txt и output.txt:



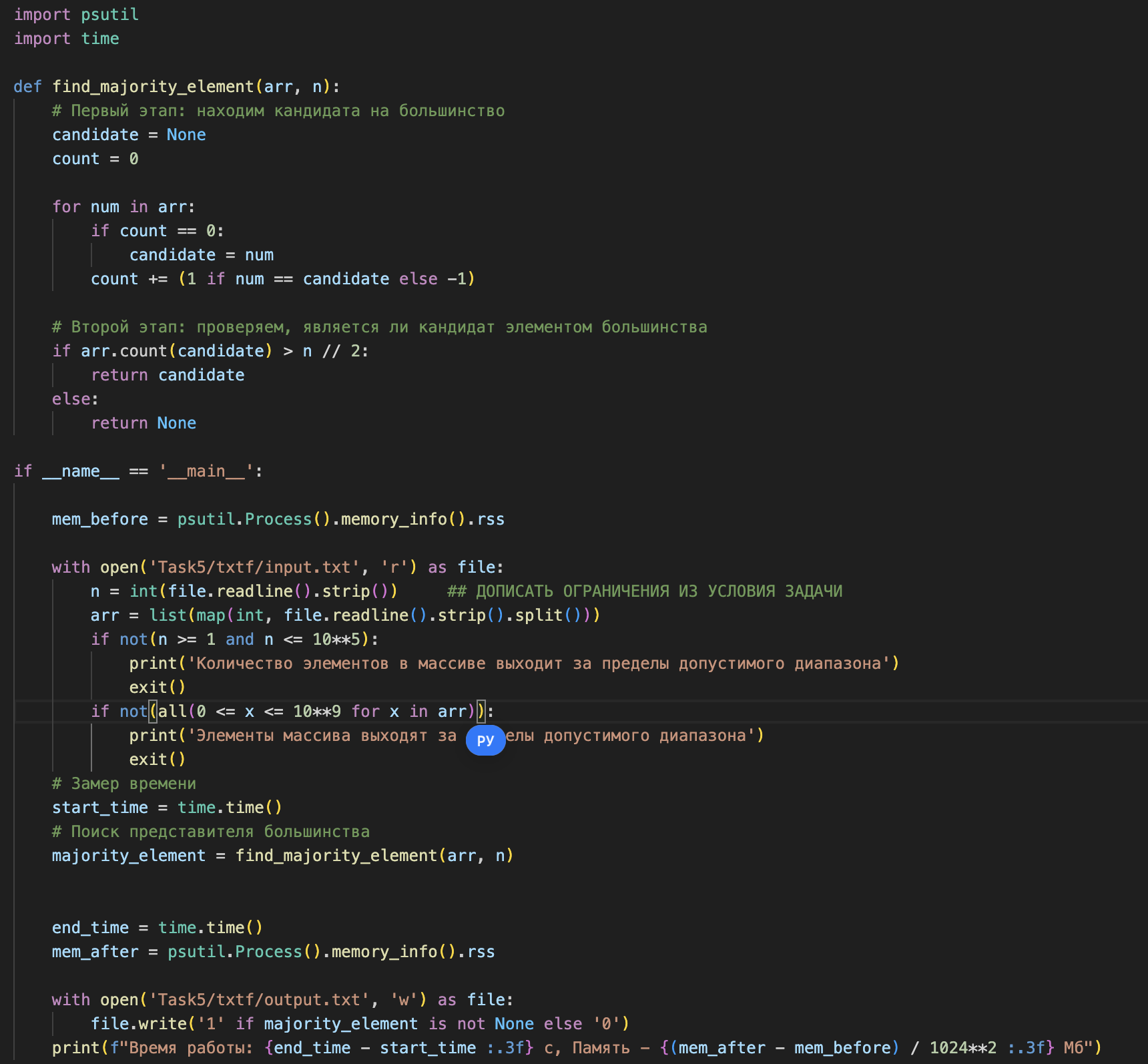


Задача№5





Решение №5:



* Функция find\_majority\_element принимает два параметра: arr - массив, в котором нужно найти элемент большинства, и n - размер массива.
* Переменная candidate используется для хранения кандидата на большинство.
* Переменная count используется для хранения счетчика, который будет увеличиваться, когда встречается кандидат, и уменьшаться, когда встречается другой элемент.

Цикл проходит по каждому элементу в массиве.

Если счетчик count равен 0, то текущий элемент становится кандидатом на большинство.

Если текущий элемент равен кандидату, счетчик увеличивается на 1, иначе уменьшается на 1.

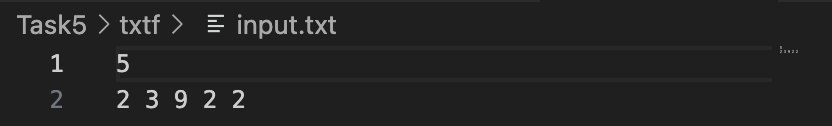
**Второй этап: проверяем, является ли кандидат элементом большинства:**

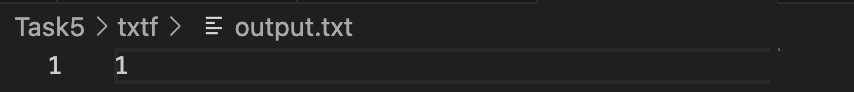
* После первого этапа кандидат на большинство известен.
* Функция проверяет, является ли кандидат элементом большинства, т.е. встречается ли он более чем в половине массива.
* Если кандидат является элементом большинства, функция возвращает его, иначе возвращает None.
* Функция find\_majority\_element вызывается с массивом arr и его размером n.
* Результат поиска, который представляет собой элемент большинства или None, если такого элемента нет, присваивается переменной majority\_element.
* Затем полученный результат записываю в output.txt.Также произвожу подсчет времени и памяти.



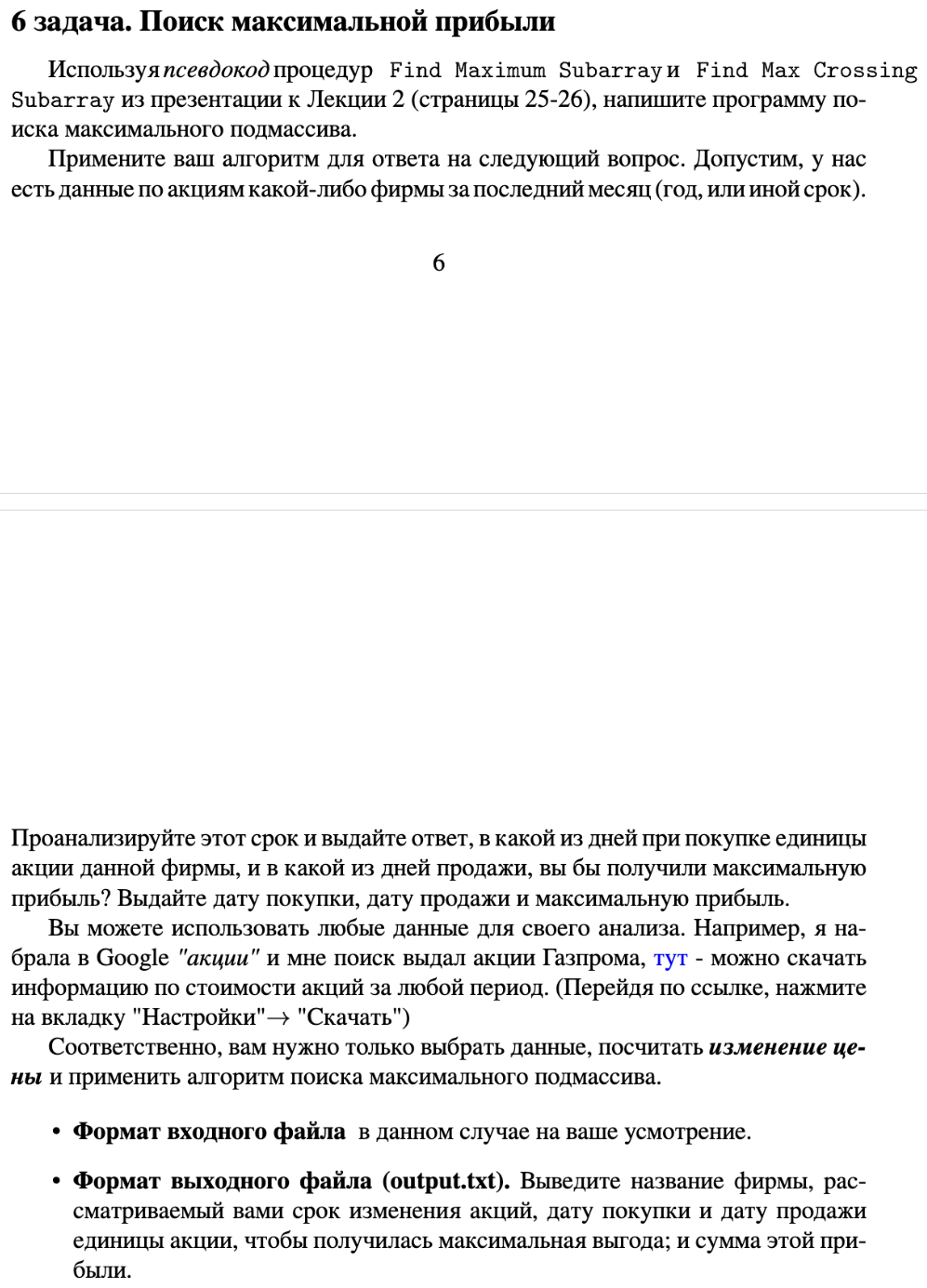
Здесь я не применяю utils.py, так как строки кода не повторяются .

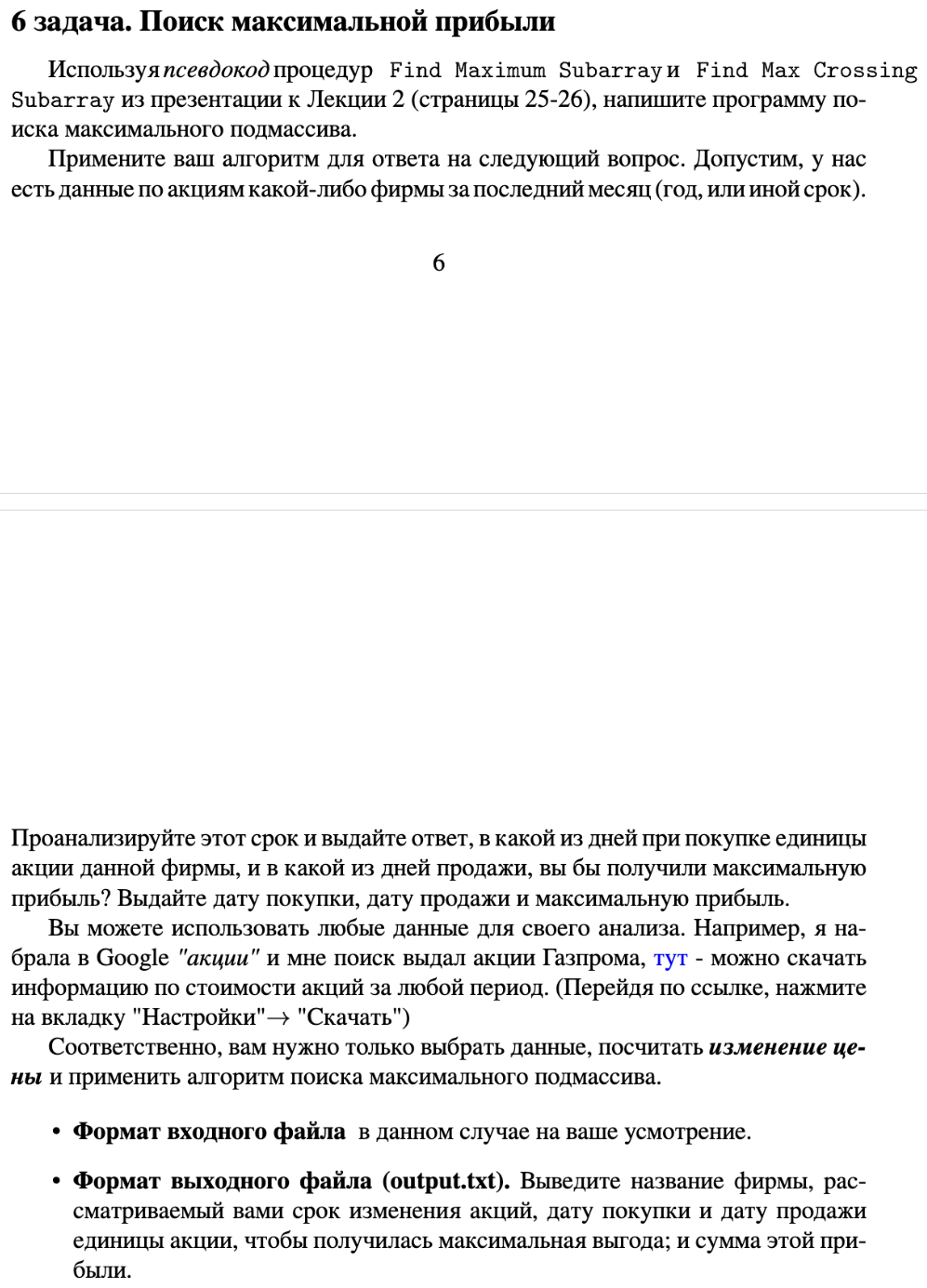
Файлы input.txt и output.txt:



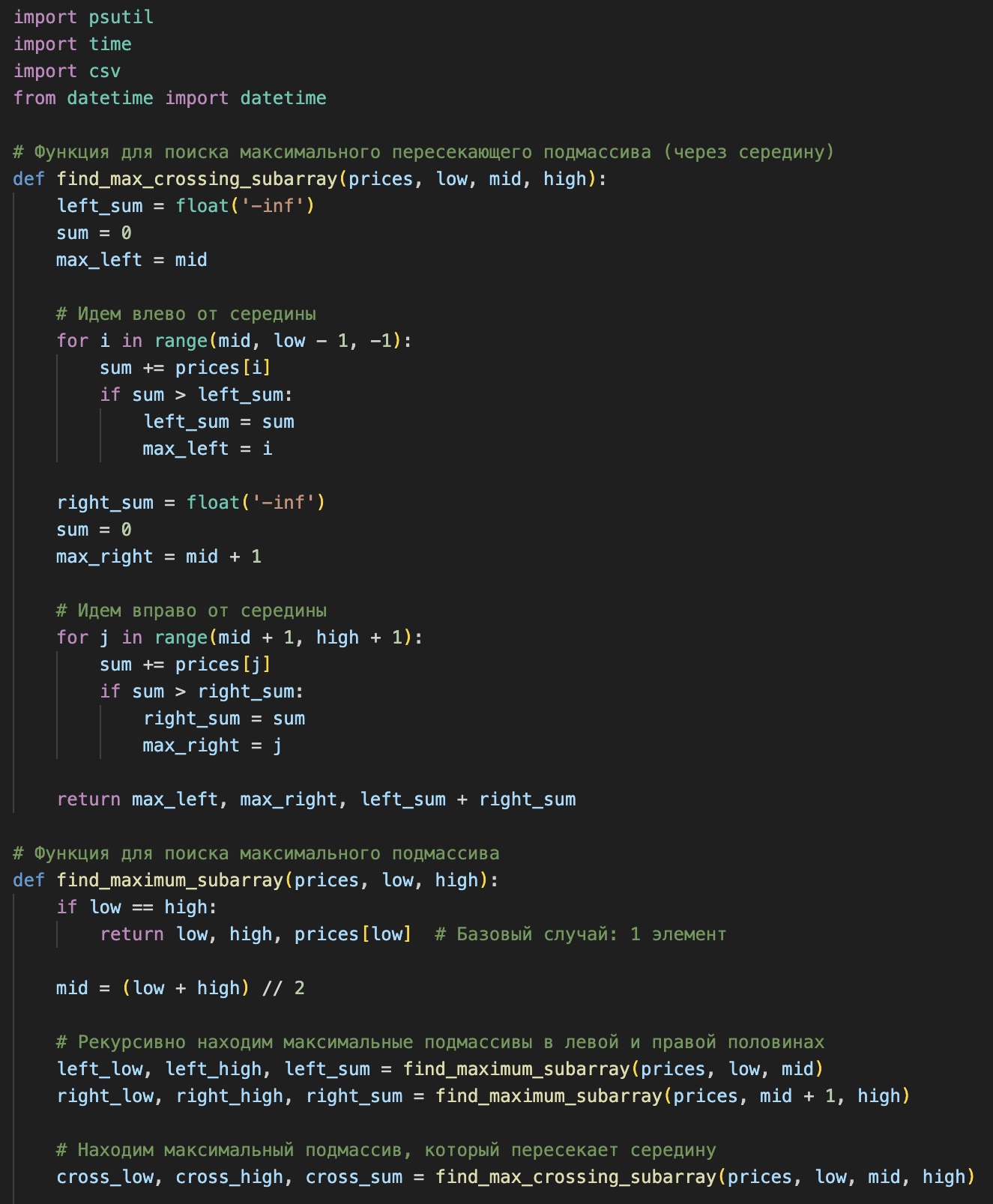


Задача№6

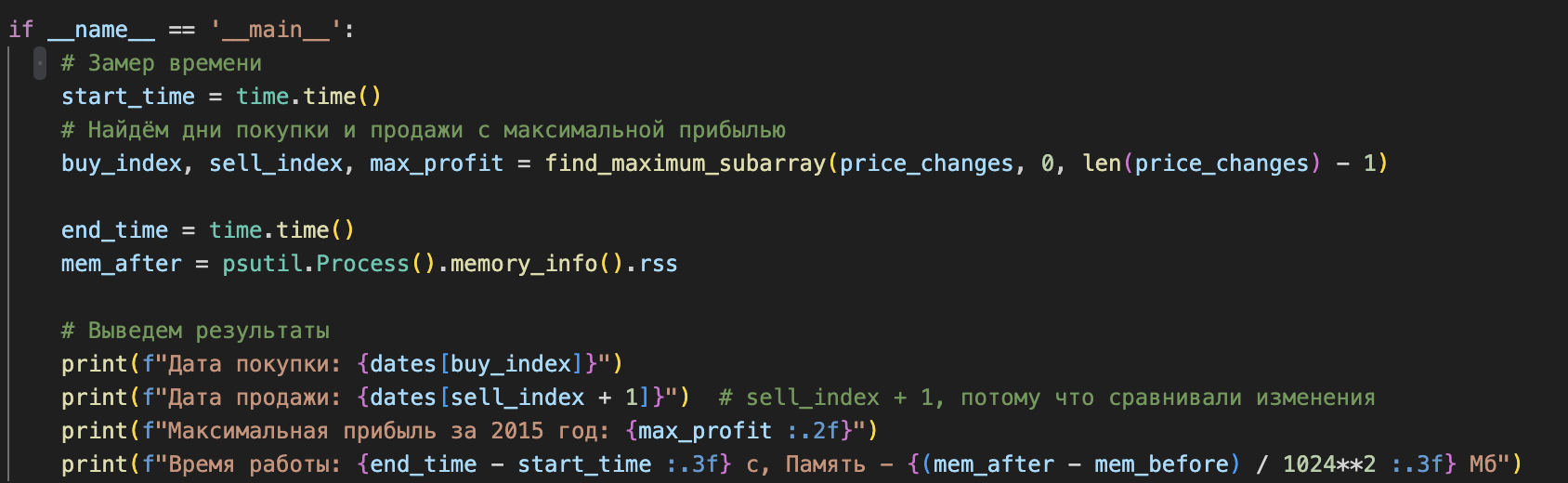




Решение:

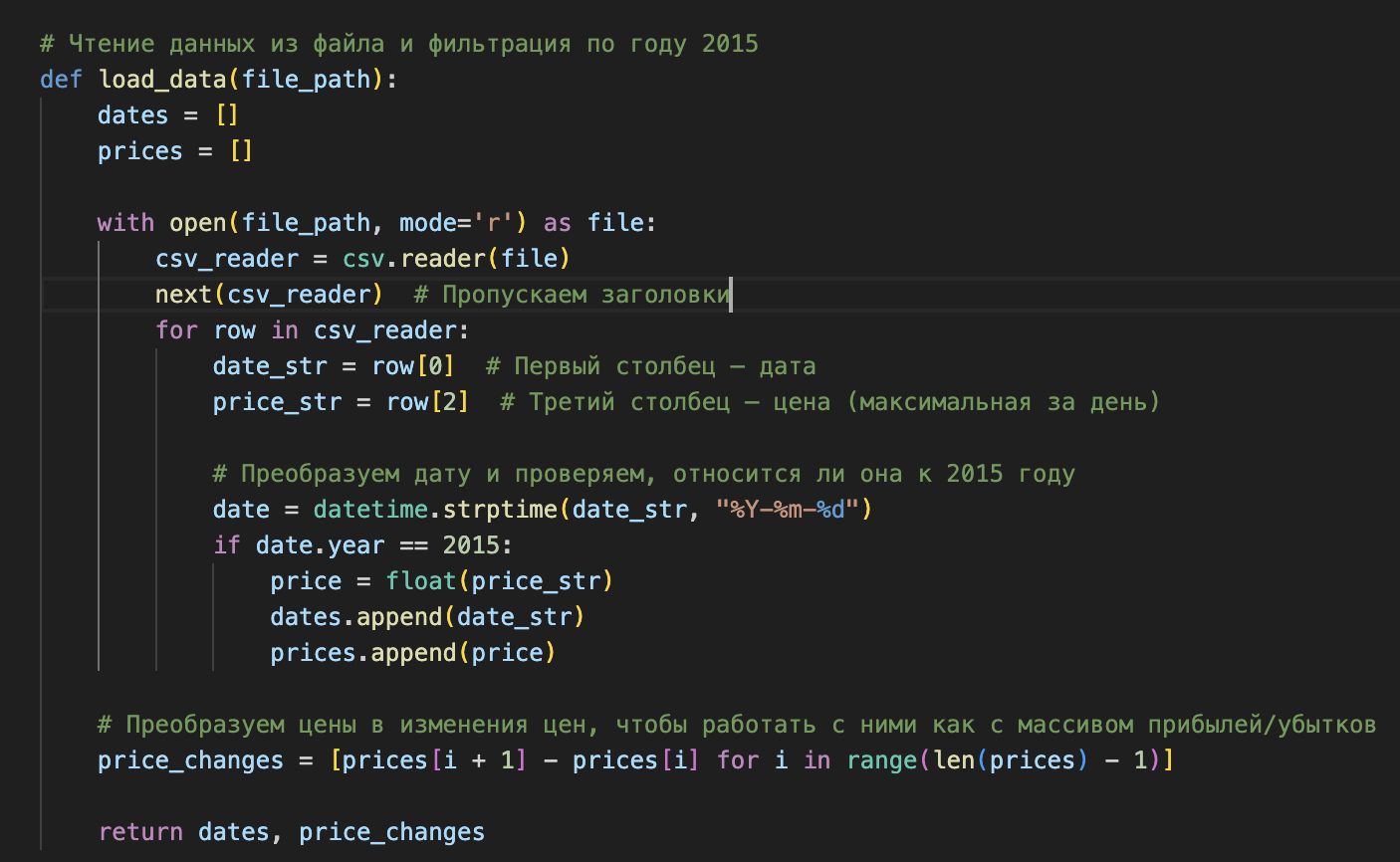




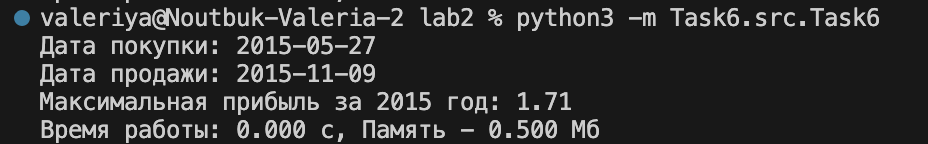


Данная программа реализует алгоритм поиска максимального подмассива в массиве цен.

* Функция find\_max\_crossing\_subarray принимает четыре параметра: prices - массив цен, low - индекс начала массива, mid - индекс середины массива, high - индекс конца массива.
* Функция ищет максимальный подмассив, который пересекает середину массива, идя влево от середины.
* Переменная left\_sum используется для хранения суммы максимального подмассива, который пересекает середину.
* Переменная sum используется для хранения суммы текущего подмассива.
* Переменная max\_left используется для хранения индекса начала максимального подмассива.
* Функция ищет максимальный подмассив, который пересекает середину массива, идя вправо от середины.
* Переменная right\_sum используется для хранения суммы максимального подмассива, который пересекает середину.
* Переменная sum используется для хранения суммы текущего подмассива.
* Переменная max\_right используется для хранения индекса конца максимального подмассива.
* Функция возвращает индекс начала, индекс конца и сумму максимального подмассива, который пересекает середину.
* Функция find\_maximum\_subarray принимает три параметра: prices - массив цен, low - индекс начала массива, high - индекс конца массива.
* Если массив состоит из одного элемента, функция возвращает индекс начала, индекс конца и значение элемента.
* Функция рекурсивно находит максимальные подмассивы в левой и правой половинах массива.
* Функция находит максимальный подмассив, который пересекает середину массива.
* Функция сравнивает суммы максимальных подмассивов и возвращает индекс начала, индекс конца и сумму максимального подмассива.



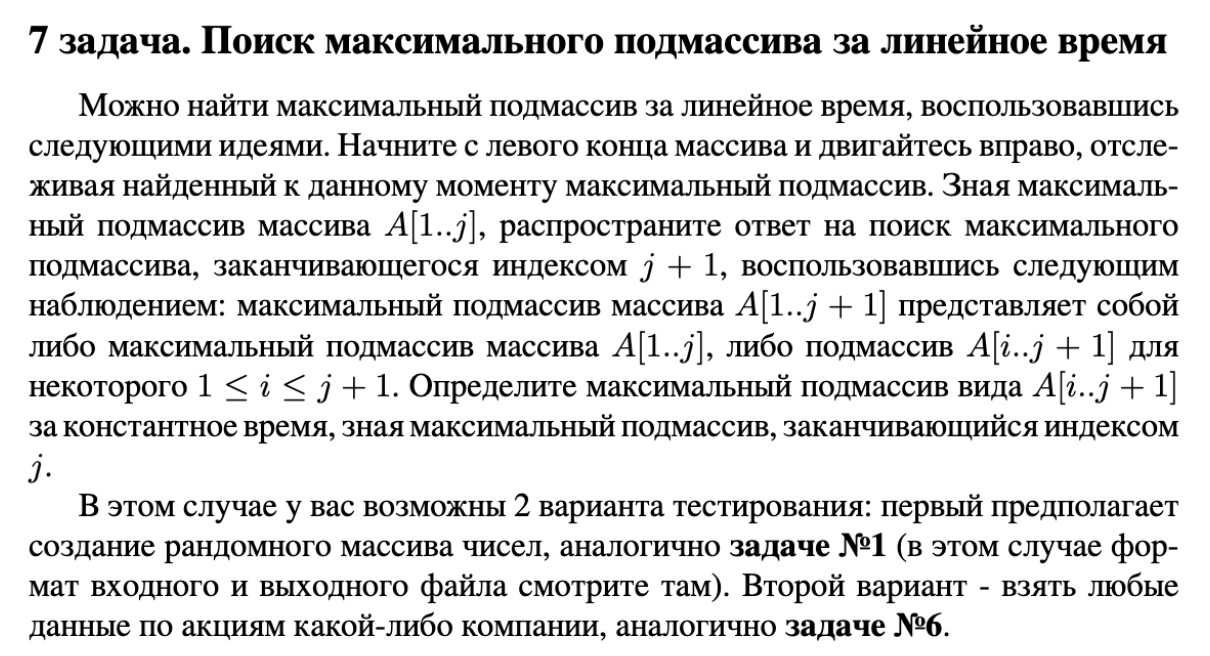
* Функция открывает файл по указанному пути file\_path в режиме чтения.
* Объект csv\_reader используется для чтения файла в формате CSV.
* Функция пропускает первую строку файла, содержащую заголовки.
* Функция читает каждую строку файла и извлекает значения даты и цены.
* Функция преобразует строку даты в объект даты с помощью метода strptime.
* Функция проверяет, относится ли дата к 2015 году, и если да, то преобразует строку цены в число с плавающей точкой и добавляет значения даты и цены в списки dates и prices.
* Функция преобразует список цен в список изменений цен, вычитая каждую цену из предыдущей.
* Функция использует генератор списков для создания нового списка изменений цен.
* Функция возвращает списки dates и price\_changes, содержащие преобразованные данные.
* Затем полученный результат записываю в output.txt.Также произвожу подсчет времени и памяти.

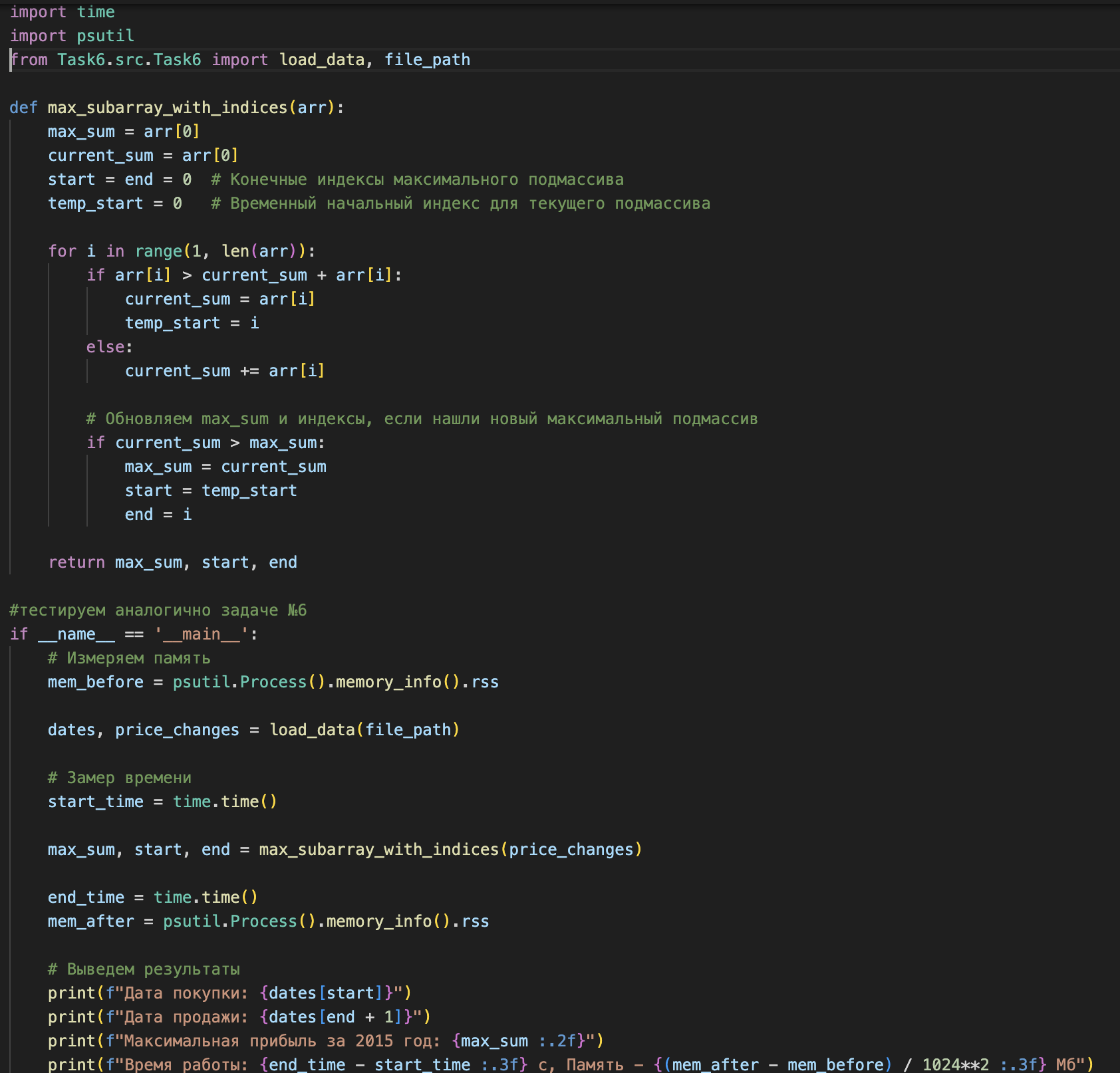


Здесь я не применяю utils.py, так как строки кода не повторяются .

Файл input.txt находится README.md, а результат выводится в терминал

Задача№7



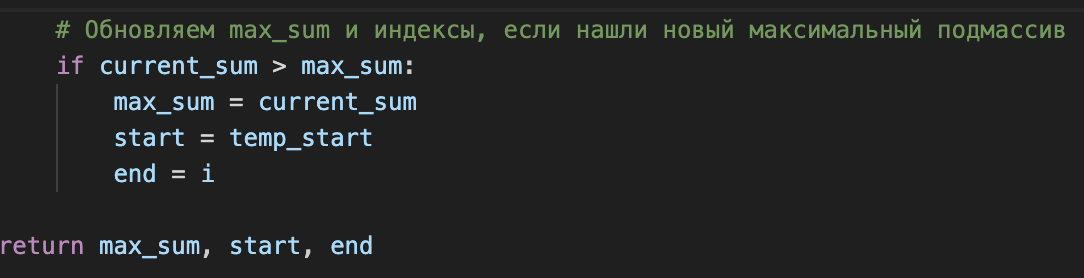


Часть кода я импортировала из 6 задания.

**Инициализация переменных:**

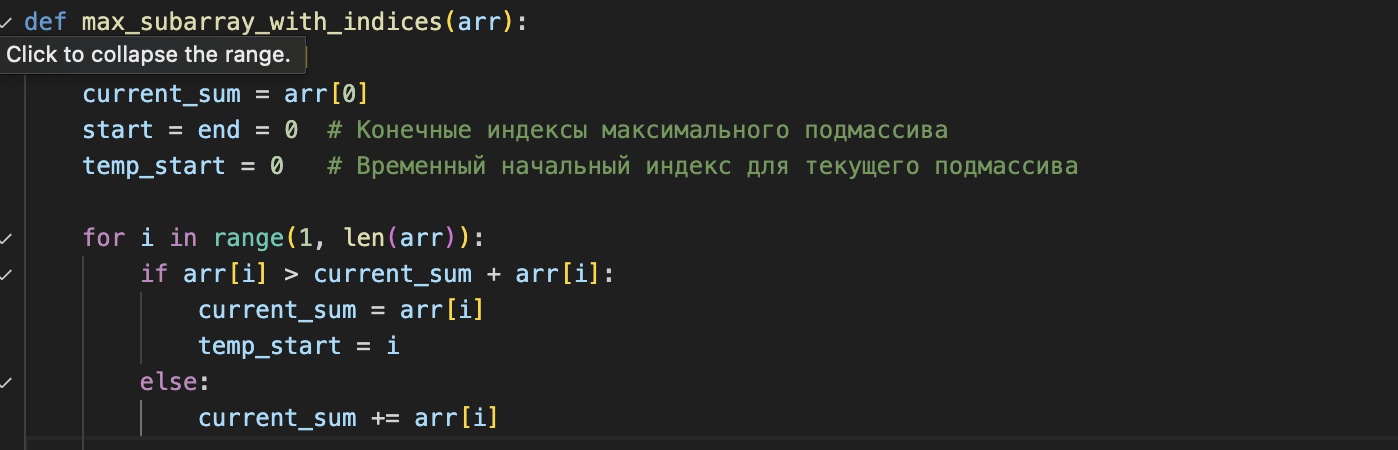
* max\_sum: переменная для хранения максимальной суммы подмассива. Инициализируется первым элементом массива arr[0].
* current\_sum: переменная для хранения текущей суммы подмассива. Инициализируется первым элементом массива arr[0].
* start и end: переменные для хранения конечных индексов максимального подмассива. Инициализируются 0.
* temp\_start: переменная для хранения временного начального индекса для текущего подмассива. Инициализируется 0.

**Условие для обновления текущей суммы:**

****

* Если текущий элемент массива arr[i] больше суммы текущего подмассива current\_sum плюс текущий элемент arr[i], то:
* Обновляем current\_sum на значение текущего элемента arr[i].
* Обновляем temp\_start на значение текущего индекса i.
* Если условие выше не выполнено, то просто добавляем текущий элемент arr[i] к текущей сумме current\_sum.

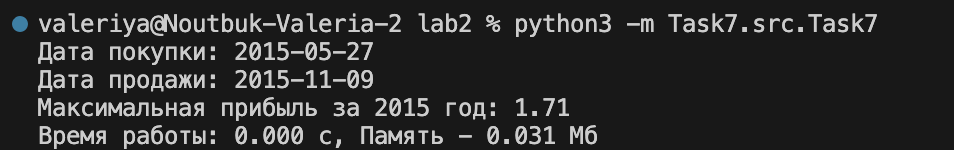
**Обновление максимальной суммы и индексов:**

****

* Если текущая сумма current\_sum больше максимальной суммы max\_sum, то:
* Обновляем max\_sum на значение текущей суммы current\_sum.
* Обновляем start на значение временного начального индекса temp\_start.
* Обновляем end на значение текущего индекса i.
* Функция возвращает три значения:
* max\_sum: максимальная сумма подмассива.
* start: начальный индекс максимального подмассива.
* end: конечный индекс максимального подмассива.

В общем, функция max\_subarray\_with\_indices находит максимальный подмассив в массиве arr и возвращает его сумму, начальный и конечный индексы.

**Вывожу результат в терминал.**



Вывод:

В данной лабораторной работе я практиковала знания по сортировке слиянием . Также решала задачи с различными условиями, которые применяют алгоритм сортировки слиянием.