САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №3

по курсу «Алгоритмы и структуры данных»

Тема: Быстрая сортировка, сортировки за линейное время

Выполнил:

Просветова Валерия Дмитриевна

К3141

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

Задача №1: Улучшение Quick sort

Задача №2: Анти - quick sort

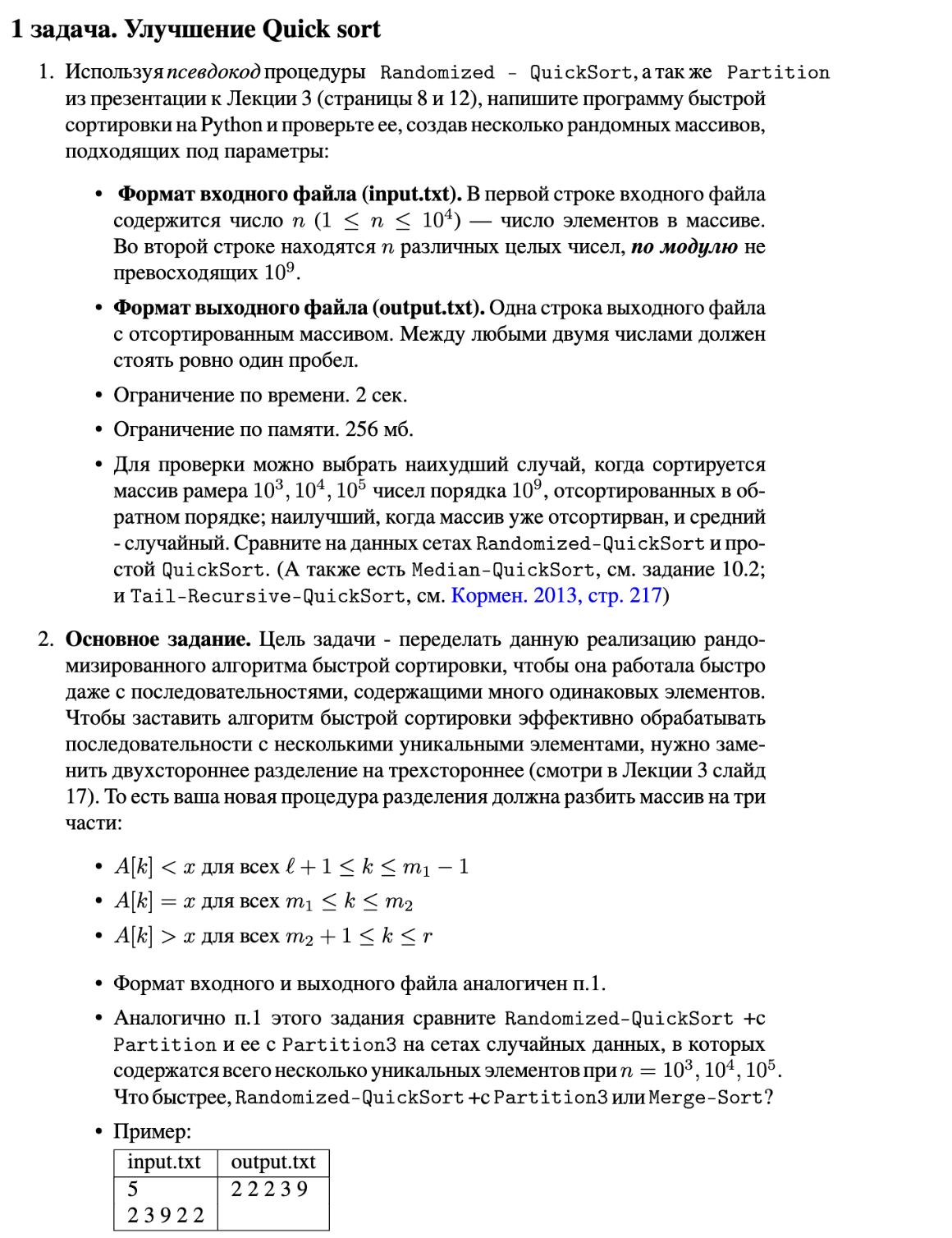
Задача №3: Сортировка пугалом

Задача №4: Точки и отрезки

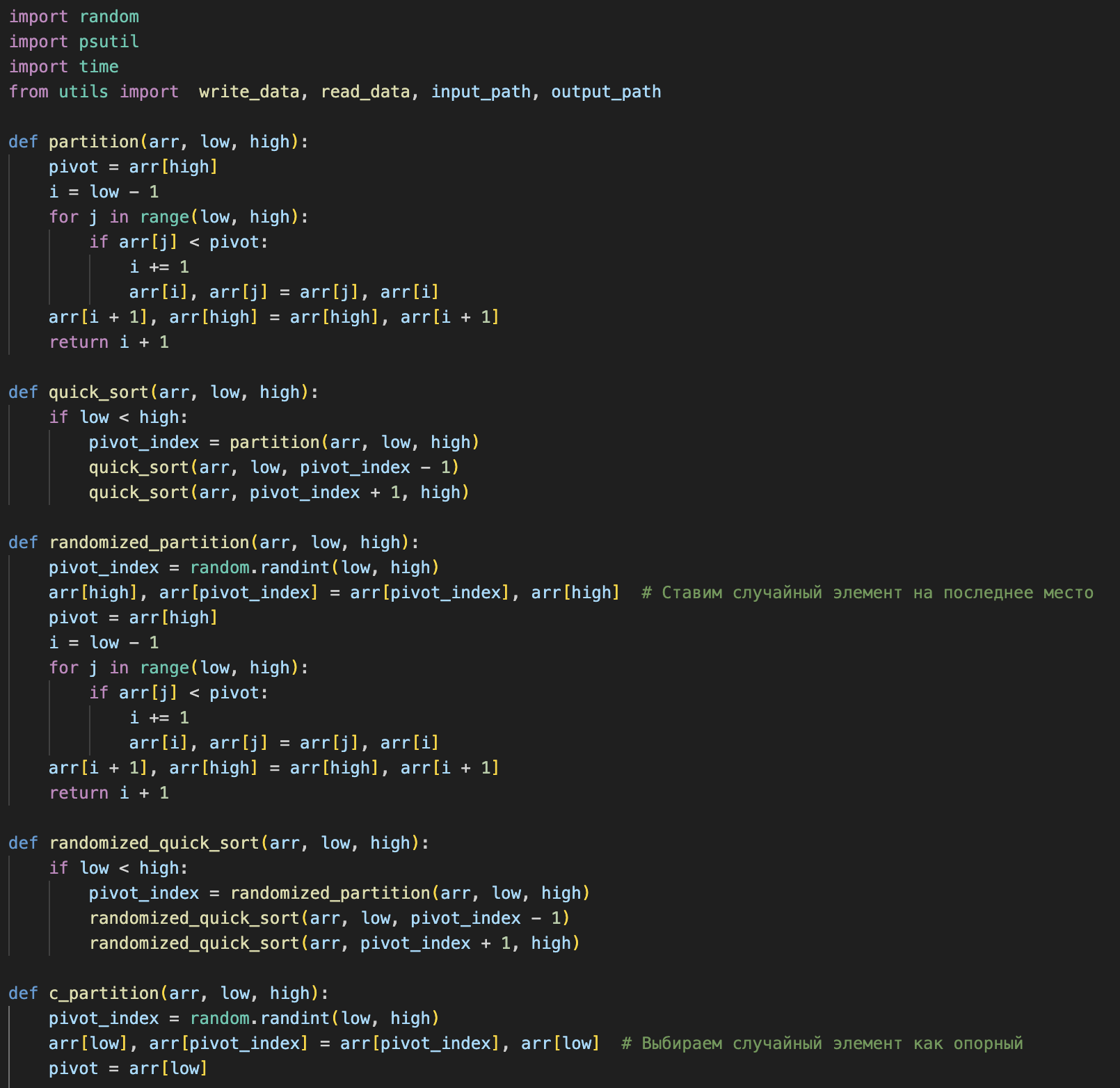
Задача №5: Индекс Хирша

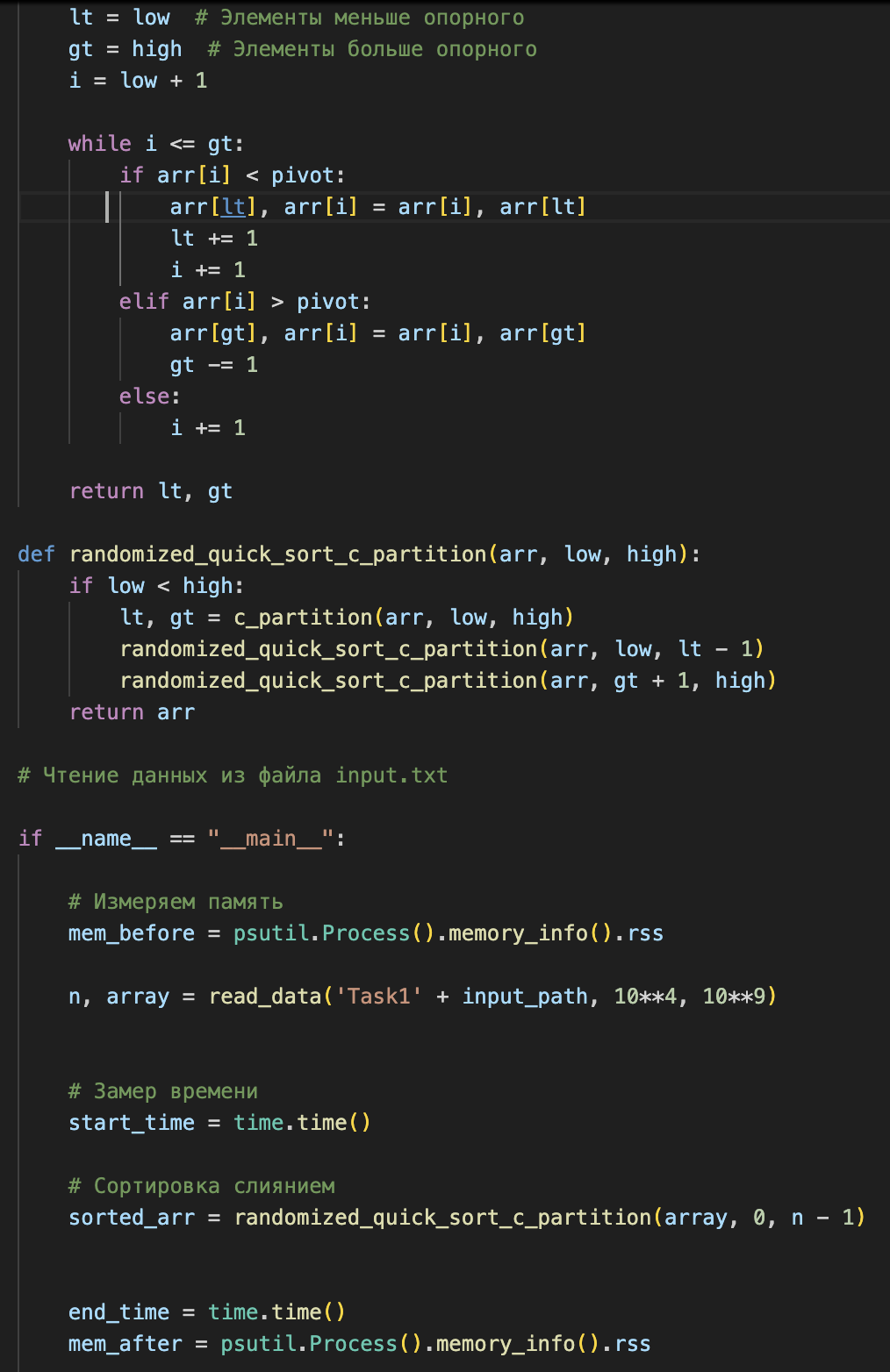
Задача №8: К ближайших точек к началу координат

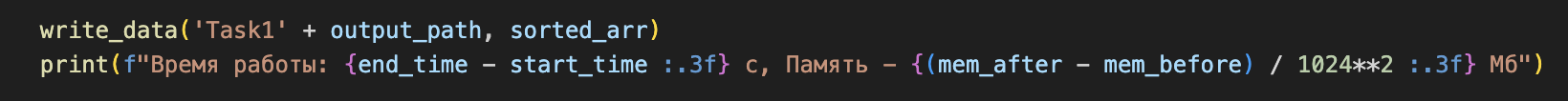
Задача №1:



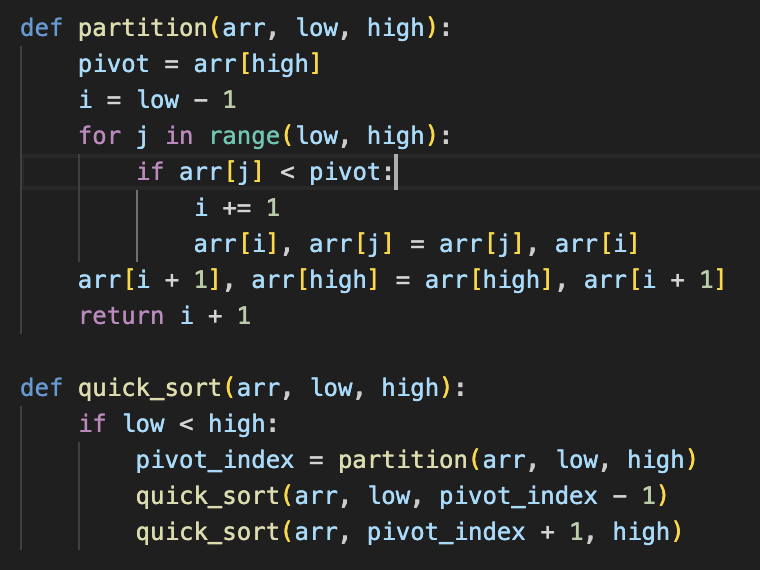
Решение задачи :







Шаг 1:



Этот код реализует алгоритм быстрой сортировки (QuickSort) для сортировки массива элементов.

**Функция partition**

Функция partition принимает три аргумента: arr (массив элементов), low (индекс начала массива) и high (индекс конца массива). Эта функция выполняет следующие действия:

1. Выбирает элемент pivot из массива, который будет использован для разделения массива на две части.
2. Инициализирует индекс i в значение low - 1.
3. Проходит по элементам массива от low до high - 1 и проверяет, меньше ли каждый элемент pivot. Если элемент меньше pivot, то:
   * i+1 .
   * Меняет местами элементы arr[i] и arr[j].
4. После прохода по всем элементам меняет местами элементы arr[i + 1] и arr[high].
5. Возвращает индекс i + 1, который теперь указывает на элемент pivot в отсортированном массиве.

**Функция quick\_sort**

Функция quick\_sort принимает три аргумента: arr (массив элементов), low (индекс начала массива) и high (индекс конца массива). Эта функция выполняет следующие действия:

Проверяет, меньше ли low high. Если да, то:

* + Вызывает функцию partition для разделения массива на две части.
  + Получает индекс pivot\_index от функции partition.
  + Вызывает функцию quick\_sort рекурсивно для двух частей массива:
    - Для части от low до pivot\_index - 1.
    - Для части от pivot\_index + 1 до high.

Шаг 2:



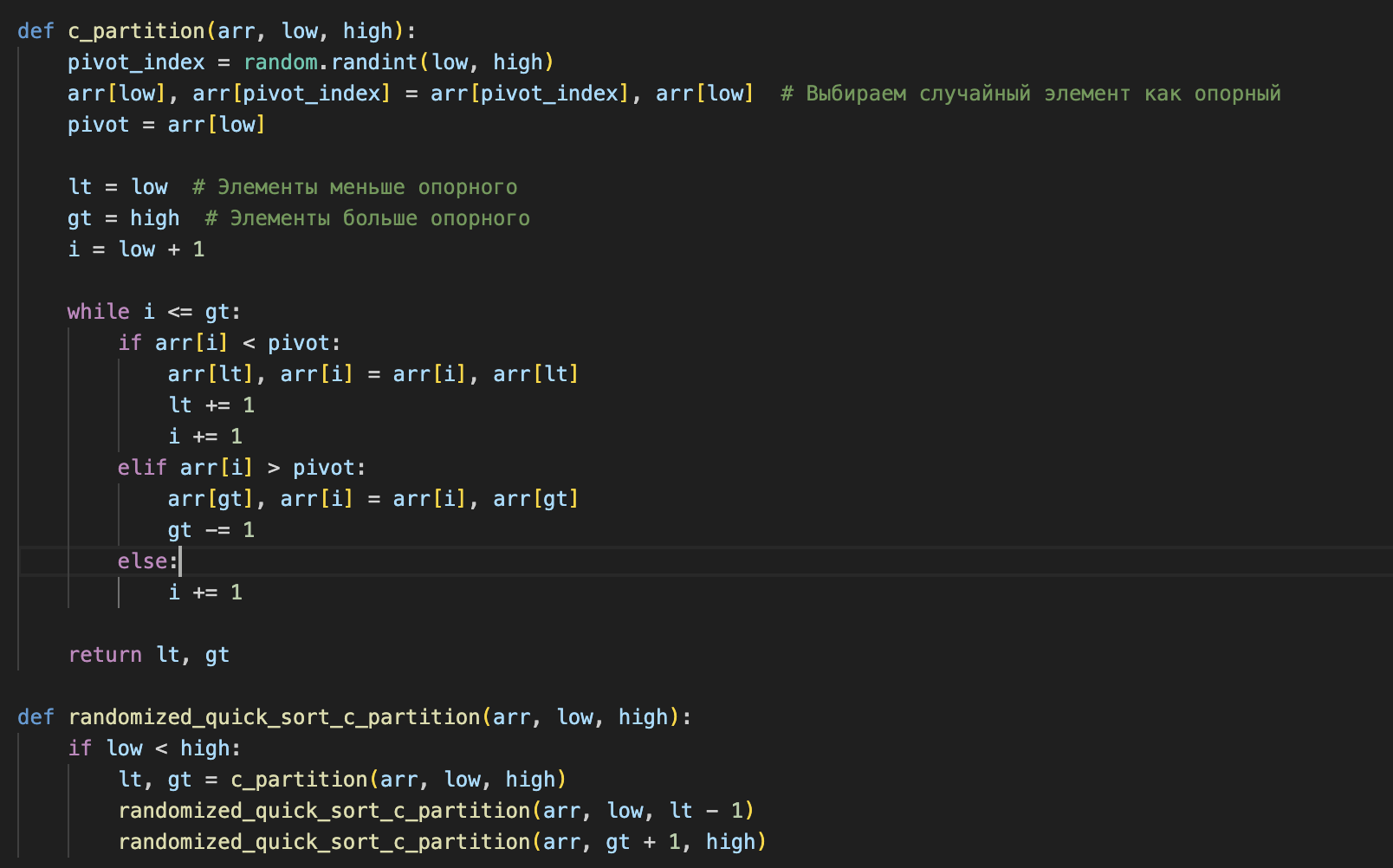
Этот код реализует алгоритм быстрой сортировки с рандомизированным выбором элемента pivot (Randomized QuickSort) для сортировки массива элементов.

**Функция randomized\_partition**

Функция randomized\_partition принимает три аргумента: arr (массив элементов), low (индекс начала массива) и high (индекс конца массива). Эта функция выполняет следующие действия:

1. Выбирает случайный индекс pivot\_index в диапазоне от low до high с помощью функции random.randint.
2. Меняет местами элементы arr[high] и arr[pivot\_index], чтобы поставить случайный элемент на последнее место в массиве.
3. Затем выполняются действия как в partition.

Шаг 3:



Эта функция c\_partition является частью алгоритма быстрой сортировки (QuickSort) и используется для разделения массива на три части: элементы меньше опорного, элементы равные опорному и элементы больше опорного.

**Вычисление опорного элемента**

Функция c\_partition начинает с выбора случайного элемента из массива в качестве опорного элемента pivot. Это делается с помощью функции random.randint, которая возвращает случайный индекс в диапазоне от low до high. Затем элементы arr[low] и arr[pivot\_index] меняются местами, чтобы поставить случайный элемент на первое место в массиве.

**Инициализация переменных**

Далее функция инициализирует три переменные:

* lt (less than): индекс элементов меньше опорного. Начинается с low.
* gt (greater than): индекс элементов больше опорного. Начинается с high.
* i: индекс текущего элемента. Начинается с low + 1.

**Цикл**

Функция затем запускает цикл, который продолжается до тех пор, пока i не превысит gt. В каждой итерации цикла функция сравнивает текущий элемент arr[i] с опорным элементом pivot.

* Если arr[i] меньше pivot, то функция меняет местами arr[lt] и arr[i], а затем увеличивает lt и i на 1.
* Если arr[i] больше pivot, то функция меняет местами arr[gt] и arr[i], а затем уменьшает gt на 1.
* Если arr[i] равен pivot, то функция просто увеличивает i на 1.

**Возвращение результатов**

После завершения цикла функция возвращает два индекса: lt и gt. lt указывает на последний элемент, который меньше опорного, а gt указывает на первый элемент, который больше опорного.

функция randomized\_quick\_sort\_c\_partition является частью алгоритма быстрой сортировки (QuickSort) и используется для сортировки массива элементов с помощью функции c\_partition для разделения массива на три части: элементы меньше опорного, элементы равные опорному и элементы больше опорного.

Затем вызвала randomized\_quick\_sort\_c\_partition.

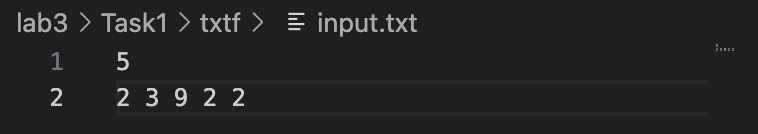
Шаг 4:

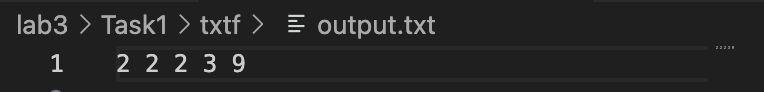
**Произвожу вычисление памяти и времени:**

****

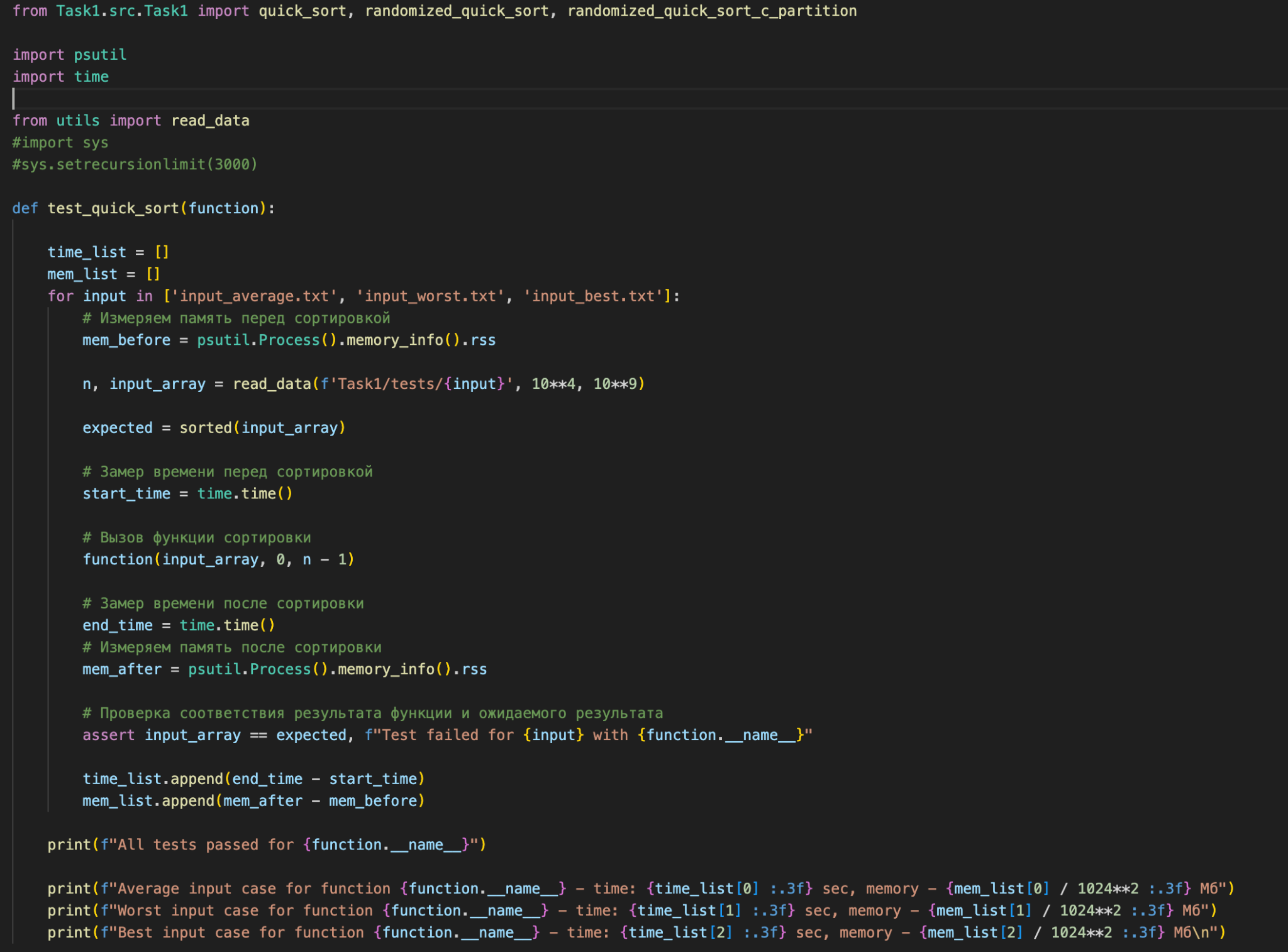
Шаг 5:

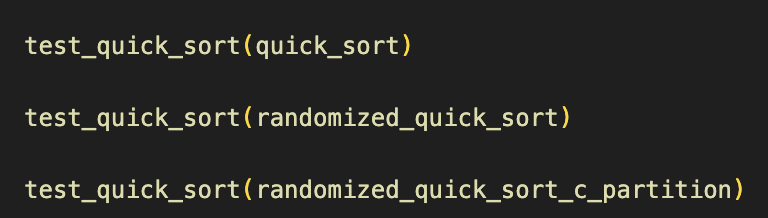
С помощью utils.py считываю данные и записываю результат в output.txt



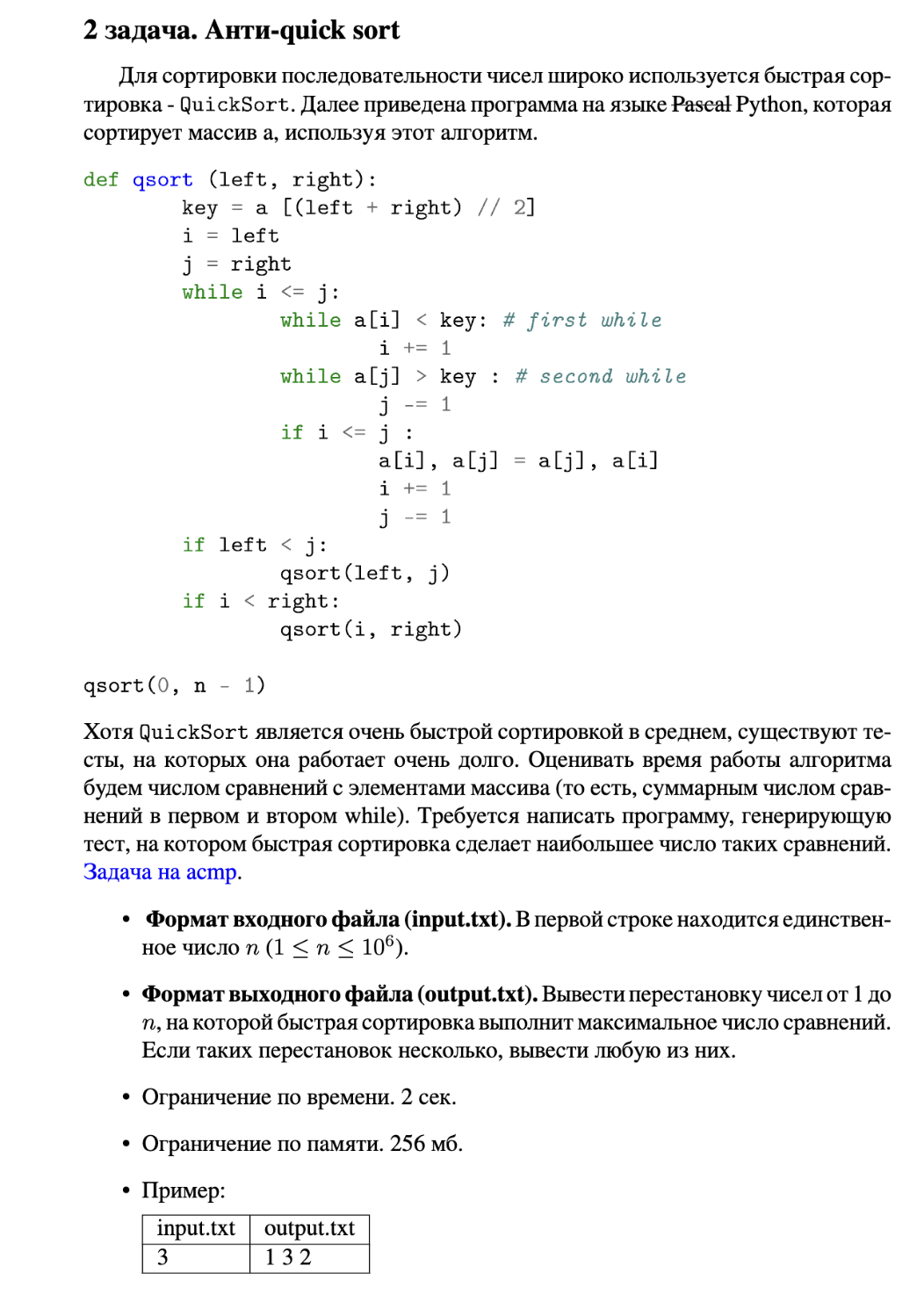


Шаг 6:Тест

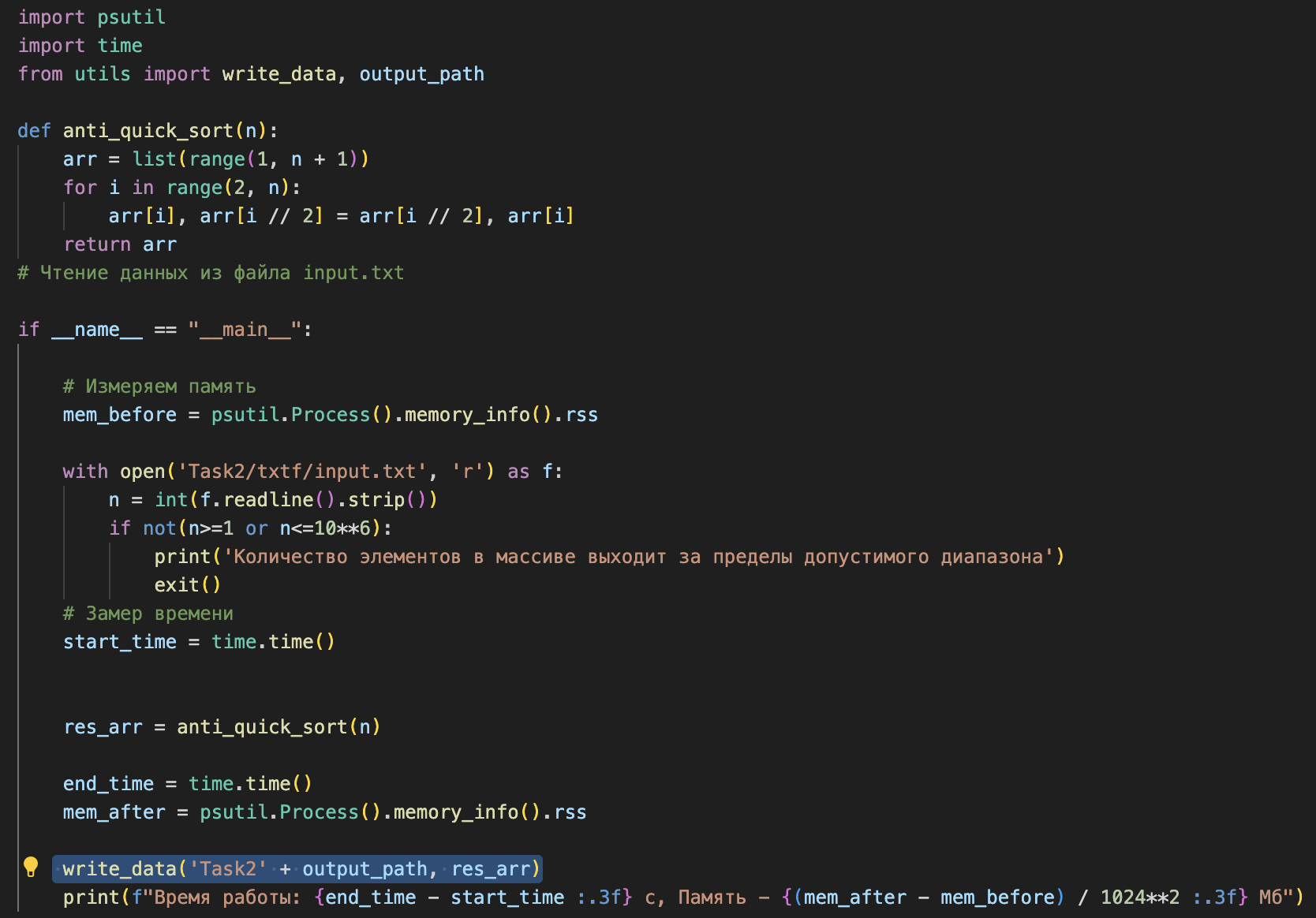




Задача №2:



Решение задачи :



Этот код реализует алгоритм, который я бы назвал "анти-быстрая сортировка" (anti-quick sort). Он принимает на вход целое число n и возвращает массив из n элементов, который является... не отсортированным.

**Шаг 1: Создание массива**

arr = list(range(1, n + 1))

* Функция создает массив arr из n элементов, начиная с 1 и заканчивая n.

**Шаг 2: Цикл**

for i in range(2, n):

* Начинается цикл от 2 до n-1 (поскольку range не включает последнее значение).
* В каждой итерации цикла будет выполняться следующий шаг.

**Шаг 3: Обмен элементов**

arr[i], arr[i // 2] = arr[i // 2], arr[i]

* В каждой итерации цикла выполняется обмен элементов в массиве arr.
* i // 2 - это целочисленное деление i на 2, т.е. результат будет целым числом.
* Элементы arr[i] и arr[i // 2] меняются местами.

**Шаг 4: Возврат массива**

return arr

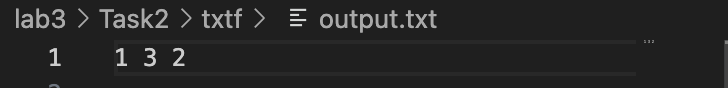
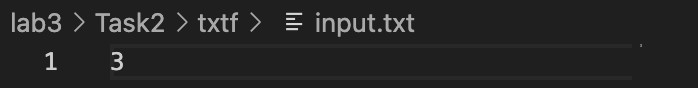
* После завершения цикла функция возвращает массив arr.

**Шаг 5: Произвожу вычисление памяти и времени**

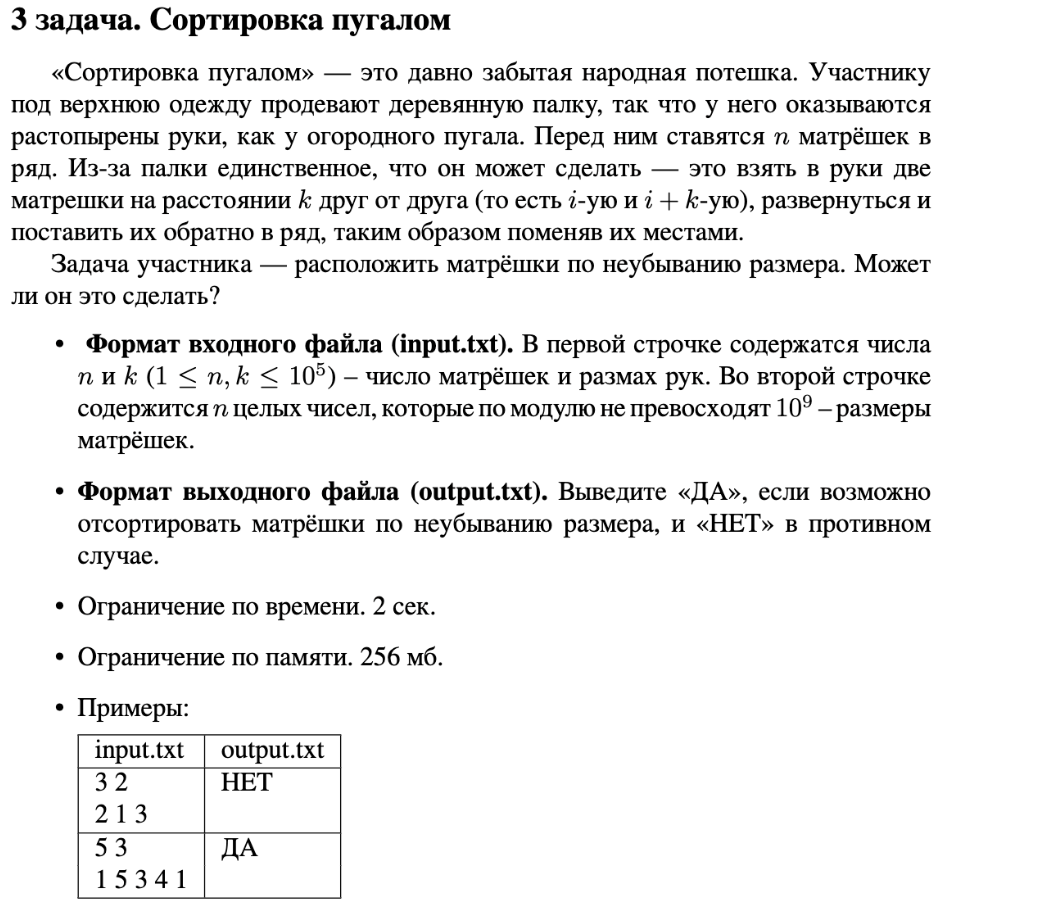


**Шаг 6:**

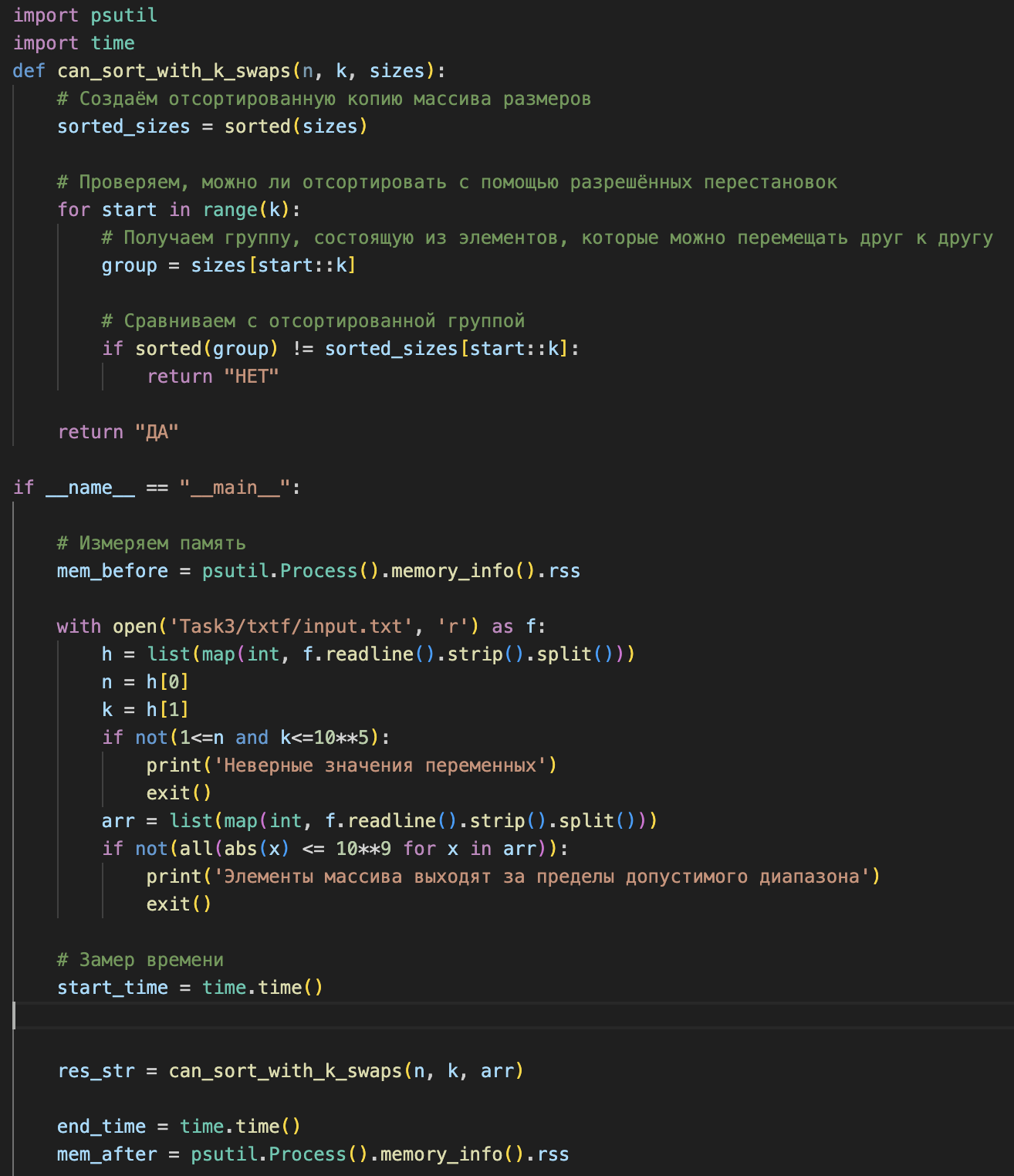
С помощью utils.py записываю результат в output.txt

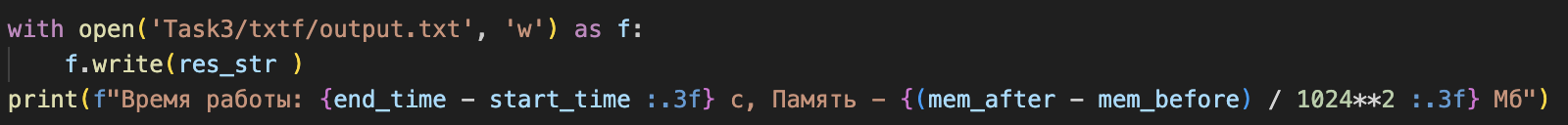


Задача №3:



Решение :





Решение задачи:

Эта функция проверяет, можно ли отсортировать массив sizes с помощью не более чем k перестановок (swap) элементов. Функция возвращает "ДА", если это возможно, и "НЕТ" в противном случае.

Давайте разберем код функции шаг за шагом:

**Шаг 1: Создание отсортированной копии массива**

sorted\_sizes = sorted(sizes)

* Функция создает отсортированную копию массива sizes с помощью встроенной функции sorted.
* Эта копия будет использоваться для сравнения с исходным массивом.

**Шаг 2: Проверка возможности сортировки**

for start in range(k):

* Начинается цикл, который будет проверять возможность сортировки для каждой группы элементов, которые можно перемещать друг к другу.
* Группы образуются с помощью шага k, т.е. элементы с индексами start, start + k, start + 2k, ...

**Шаг 3: Получение группы элементов**

group = sizes[start::k]

* В каждой итерации цикла получается группа элементов, которые можно перемещать друг к другу.
* Группа образуется с помощью среза sizes[start::k], который включает элементы с индексами start, start + k, start + 2k, ...

**Шаг 4: Сравнение с отсортированной группой**

if sorted(group) != sorted\_sizes[start::k]:

* В каждой итерации цикла сравнивается отсортированная группа элементов с соответствующей группой в отсортированной копии массива.
* Если группы не совпадают, функция возвращает "НЕТ", т.е. сортировка не возможна с помощью не более чем k перестановок.

**Шаг 5: Возврат результата**

return "ДА"

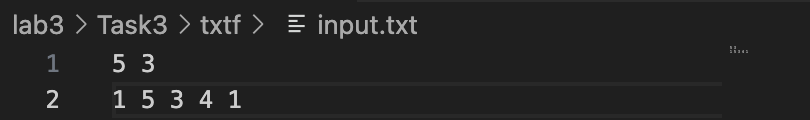
* Если функция не вернула "НЕТ" после проверки всех групп, она возвращает "ДА", т.е. сортировка возможна с помощью не более чем k перестановок.

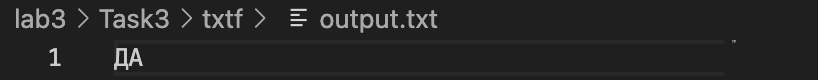
**Шаг 6: Произвожу вычисление памяти и времени**

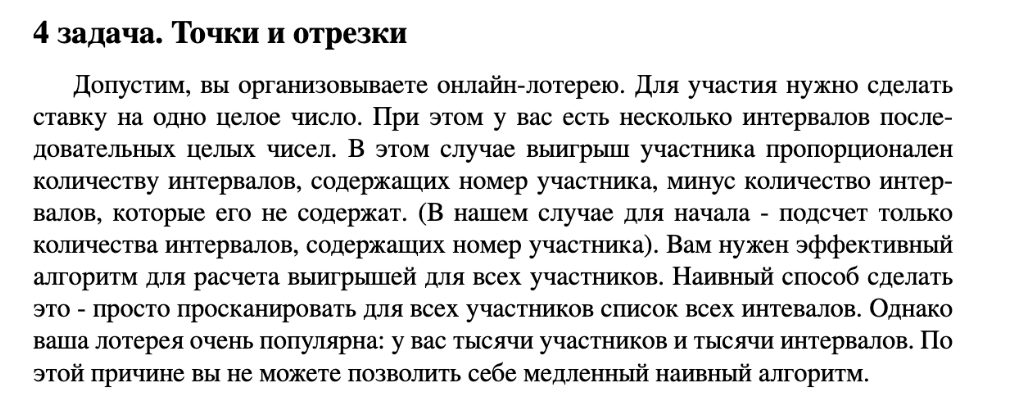
****

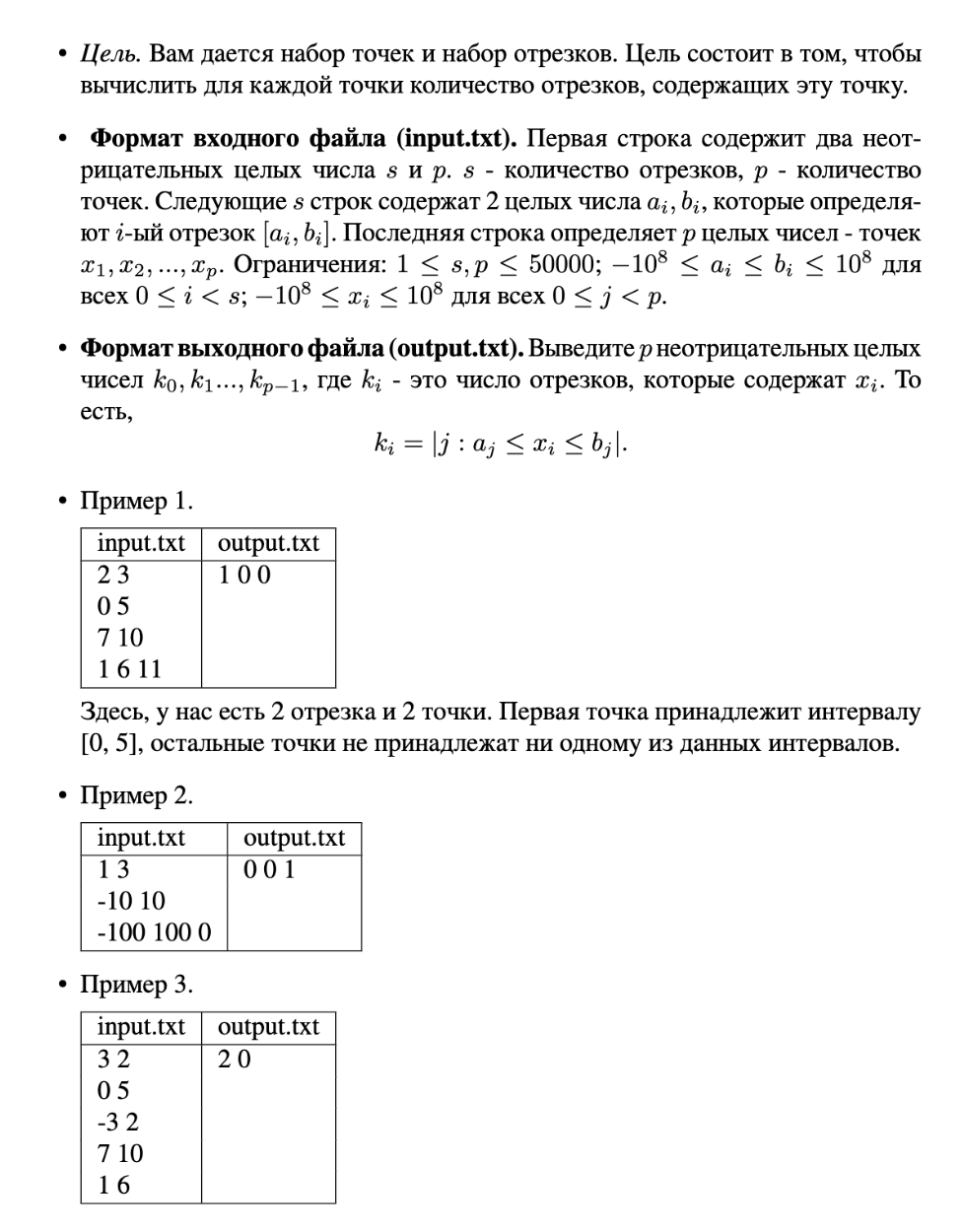
**Шаг 7:**

С помощью utils.py записываю результат в output.txt



Задача №4:





Решение задачи:

Эта функция count\_segments\_containing\_points принимает два списка: segments и points. segments - это список отрезков, каждый из которых представлен как кортеж из двух целых чисел, обозначающих начало и конец отрезка. points - это список целых чисел, представляющих точки на числовой прямой.

Функция возвращает список, в котором каждый элемент представляет количество отрезков, которые содержат соответствующую точку.

**Шаг 1: Разделение отрезков на начала и концы**

starts = sorted([segment[0] for segment in segments]) ends = sorted([segment[1] for segment in segments])

* Эти две строки кода разделяют отрезки на начала и концы и сортируют их по возрастанию.
* starts - это список начал отрезков, а ends - это список концов отрезков.

**Шаг 2: Сортировка точек и сохранение их исходного индекса**

points = sorted((point, idx) for idx, point in enumerate(points))

* Эта строка кода сортирует точки по возрастанию и сохраняет их исходный индекс.
* enumerate - это функция, которая возвращает итератор, который на каждой итерации возвращает кортеж из двух элементов: индекса и значения элемента в коллекции.
* (point, idx) - это кортеж, который содержит точку и ее исходный индекс.
* points сортируется, но каждому значению добавляется индекс изначального положения, чтобы результаты были правильно упорядочены в конце.
* result инициализируется как список нулей, который будет хранить количество отрезков для каждой точки.

**Шаг 3: Инициализация указателей и активных отрезков**

* **s\_ptr и e\_ptr —** указатели, которые будут перемещаться по спискамstarts **и** ends **соответственно.**
* **active\_segments —** переменная для отслеживания количества "активных" отрезков (тех, которые покрывают текущую точку).

**Шаг 4: Обход точек и подсчет активных отрезков**

for point, idx in points:

* Эта строка кода начинает обход точек в отсортированном порядке.

**Шаг 5: Увеличение активных отрезков**

while s\_ptr < len(starts) and starts[s\_ptr] <= point: active\_segments += 1 s\_ptr += 1

* Эта часть кода увеличивает количество активных отрезков, если начало отрезка меньше или равно текущей точке.
* s\_ptr - это указатель на начала отрезков, который увеличивается на 1 после каждого увеличения активных отрезков.

**Шаг 6: Уменьшение активных отрезков**

while e\_ptr < len(ends) and ends[e\_ptr] < point: active\_segments -= 1 e\_ptr += 1

* Эта часть кода уменьшает количество активных отрезков, если конец отрезка меньше текущей точки.
* e\_ptr - это указатель на концы отрезков, который увеличивается на 1 после каждого уменьшения активных отрезков.

**Шаг 7: Сохранение результата для текущей точки**

result[idx] = active\_segments

* Эта строка кода сохраняет количество активных отрезков для текущей точки в списке result.

**Шаг 8: Возврат результата**

return result

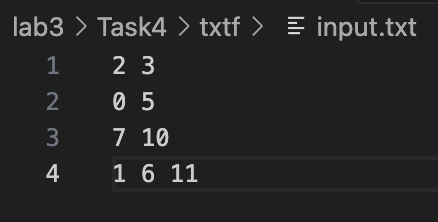
* Эта строка кода возвращает список result, который содержит количество отрезков, которые содержат каждую точку.

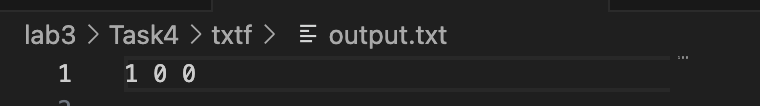
**Шаг 9: Произвожу вычисление памяти и времени**



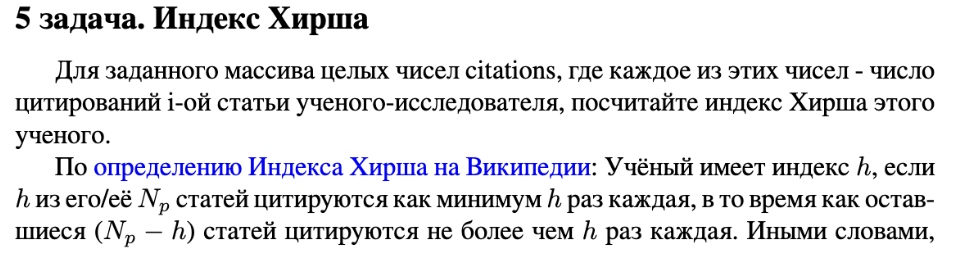
**Шаг 10:**

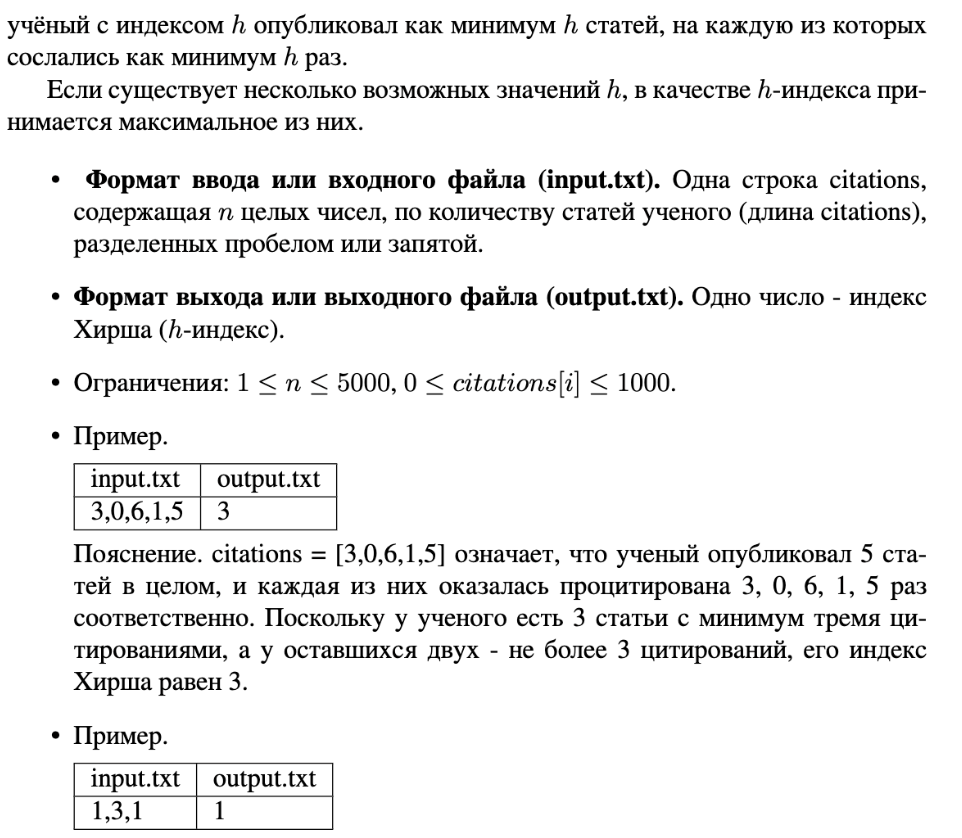
С помощью utils.py записываю результат в output.txt



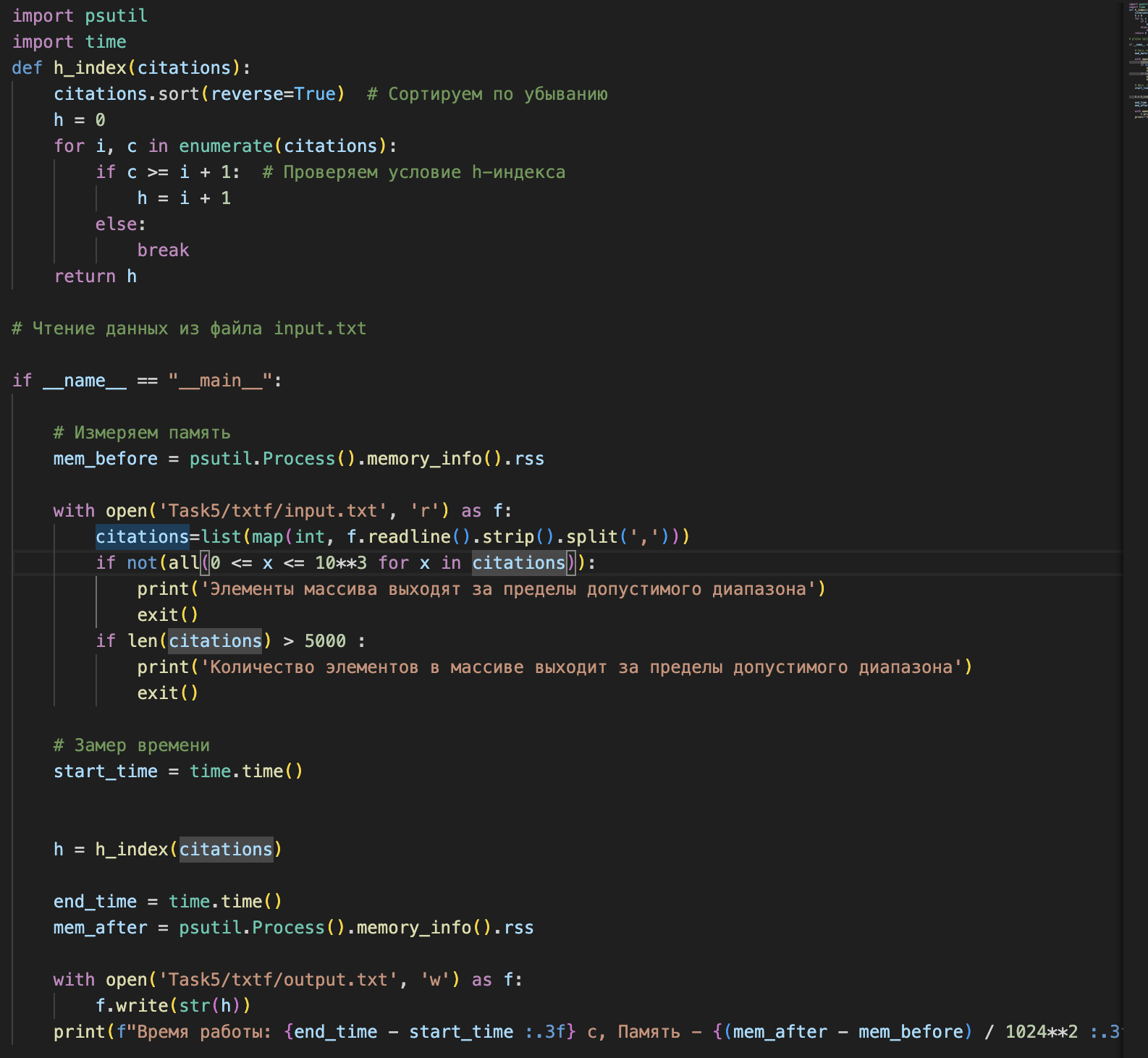


Задача №5:





Решение задачи:



Эта функция h\_index принимает список citations, который содержит количество цитирований научных работ. Функция возвращает h-индекс, который представляет собой наибольшее значение h, при котором существует не менее h работ, каждая из которых цитирована не менее h раз.

Давайте разберем код функции шаг за шагом:

**Шаг 1: Сортировка списков цитирований**

citations.sort(reverse=True)

* Эта строка кода сортирует список citations по убыванию.

**Шаг 2: Инициализация переменных**

h = 0

* Эта строка кода инициализирует переменную h нулевым значением.

**Шаг 3: Обход списка цитирований**

for i, c in enumerate(citations):

* Эта строка кода начинает обход списка citations с помощью цикла for.
* В цикле for эти кортежи будут разбиваться на две переменные: i и c, что позволяет доступиться к индексу и значению элемента в списке.

**Шаг 4: Проверка условия h-индекса**

if c >= i + 1:

* Эта часть кода проверяет условие h-индекса.
* Условие h-индекса утверждает, что количество работ, каждая из которых цитирована не менее h раз, не меньше h.
* c - это количество цитирований текущей работы, а i - это индекс текущей работы.

**Шаг 5: Обновление значения h**

h = i + 1

* Эта строка кода обновляет значение переменной h на текущий индекс i плюс 1, если условие h-индекса выполняется.

**Шаг 6: Выход из цикла**

else: break

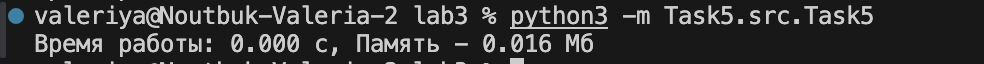
* Эта часть кода прерывает цикл, если условие h-индекса не выполняется для текущей работы.

**Шаг 7: Возврат значения h**

return h

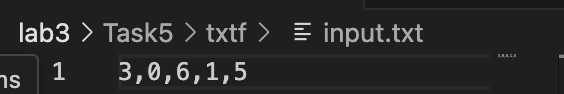
* Эта строка кода возвращает значение переменной h, которое представляет собой h-индекс.

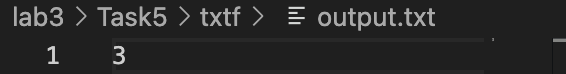
**Шаг 8:** **Произвожу вычисление памяти и времени**

****

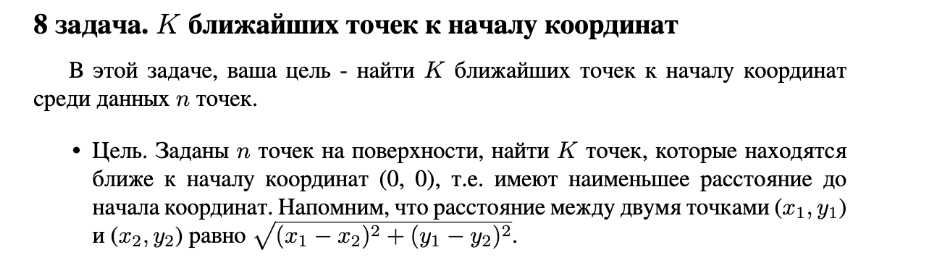
**Шаг 9:**

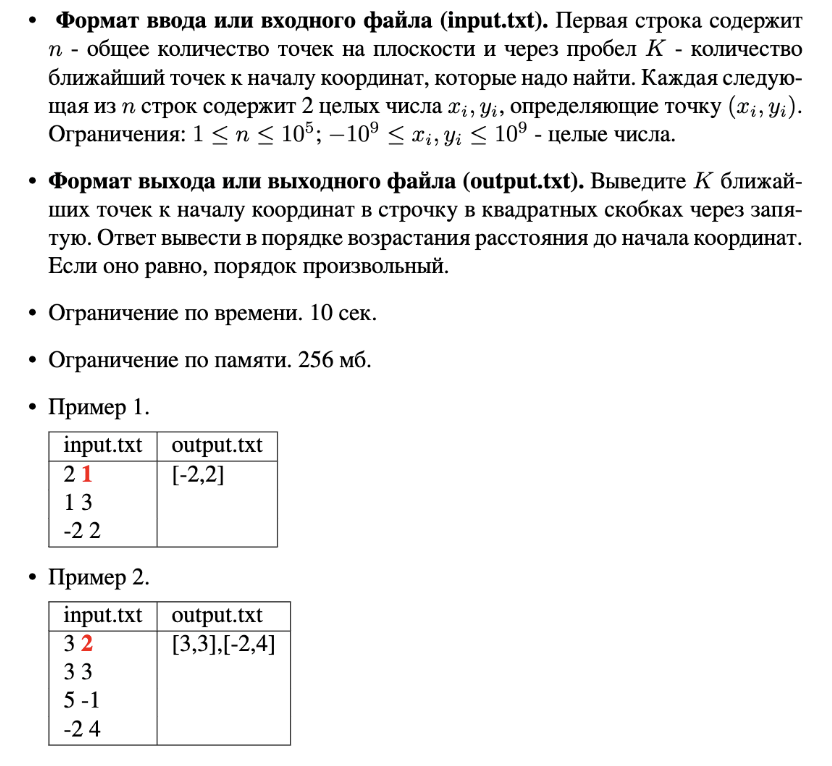
С помощью utils.py записываю результат в output.txt



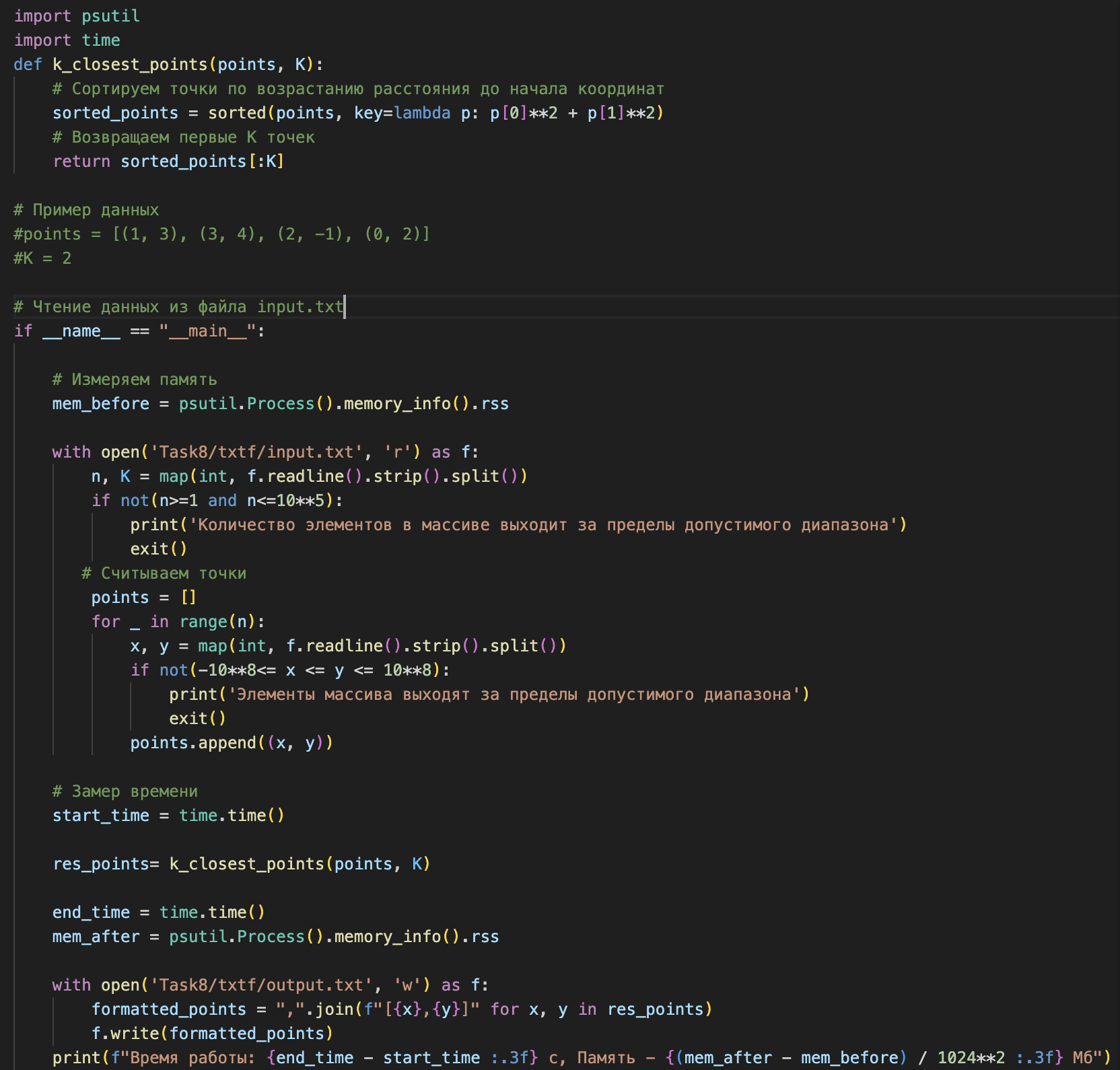


Задача №8:





Решение:



Эта функция k\_closest\_points принимает два аргумента: points и K. points - это список точек, представленных в виде кортежей из двух чисел, обозначающих координаты точки. K - это целое число, представляющее количество ближайших точек, которые нужно вернуть.

Функция возвращает список из K ближайших точек, отсортированных по возрастанию расстояния до начала координат.

Давайте разберем код функции шаг за шагом:

**Шаг 1: Сортировка точек по расстоянию до начала координат**

sorted\_points = sorted(points, key=lambda p: p[0]\*\*2 + p[1]\*\*2)

* Эта строка кода сортирует список points по расстоянию до начала координат с помощью функции sorted.
* key - это параметр функции sorted, который определяет, по какому критерию должны быть отсортированы элементы.
* В данном случае, key - это анонимная функция, которая принимает точку p и возвращает сумму квадратов ее координат p[0]\*\*2 + p[1]\*\*2.
* Эта анонимная функция используется для определения расстояния до начала координат для каждой точки.
* sorted возвращает отсортированный список точек по возрастанию расстояния до начала координат.

**Шаг 2: Возвращение первых K точек**

return sorted\_points[:K]

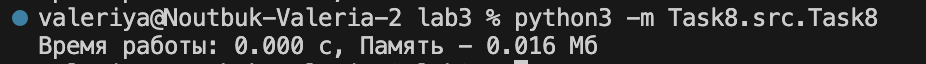
* Эта строка кода возвращает список из первых K элементов отсортированного списка sorted\_points.

\*\*\*|Функция lambda p в коде является анонимной функцией, которая принимает точку p и возвращает сумму квадратов ее координат p[0]\*\*2 + p[1]\*\*2.

Анонимная функция - это функция, которая не имеет имени и определяется прямо в месте ее использования. В этом случае, анонимная функция используется как значение параметра key функции sorted.

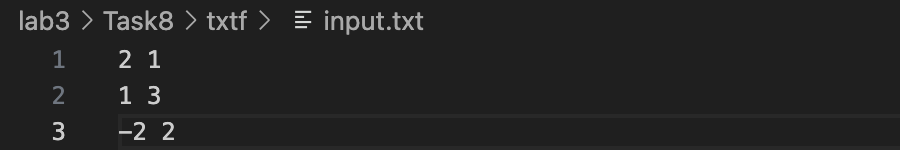
В общем, анонимные функции часто используются в Python для простых вычислений или преобразований данных, когда нет необходимости в отдельном определении функции.|\*\*\*

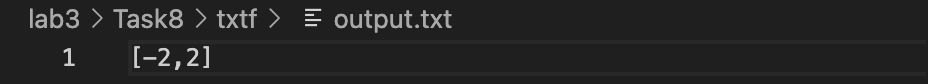
**Шаг 3:** **Произвожу вычисление памяти и времени**

****

**Шаг 4:**

Записываю результат в output.txt обычным способом:



****