САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Сортировка вставками,выбором,пузырьковая

Выполнил:

Просветова В.Д.

К3141

Проверила:

Афанасьев.А.В.

Санкт-Петербург

2024 г.

# Содержание отчета

Содержание отчета:

Задача №1: Сортировка вставкой

Задача №2: Сортировка вставкой +

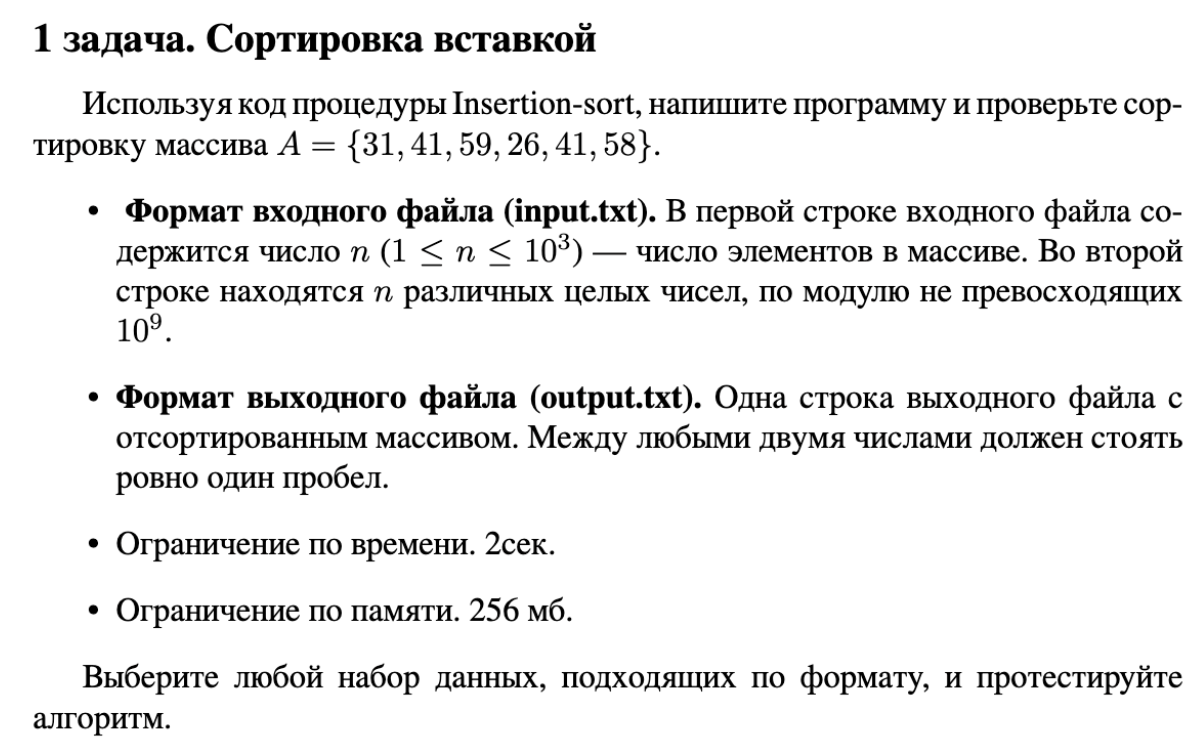
Задача №3:Сортировка вставкой по убыванию

Задача №4:Линейный поиск

Задача №5:Сортировка выбором

Задача №6:Пузырьковая сортировка

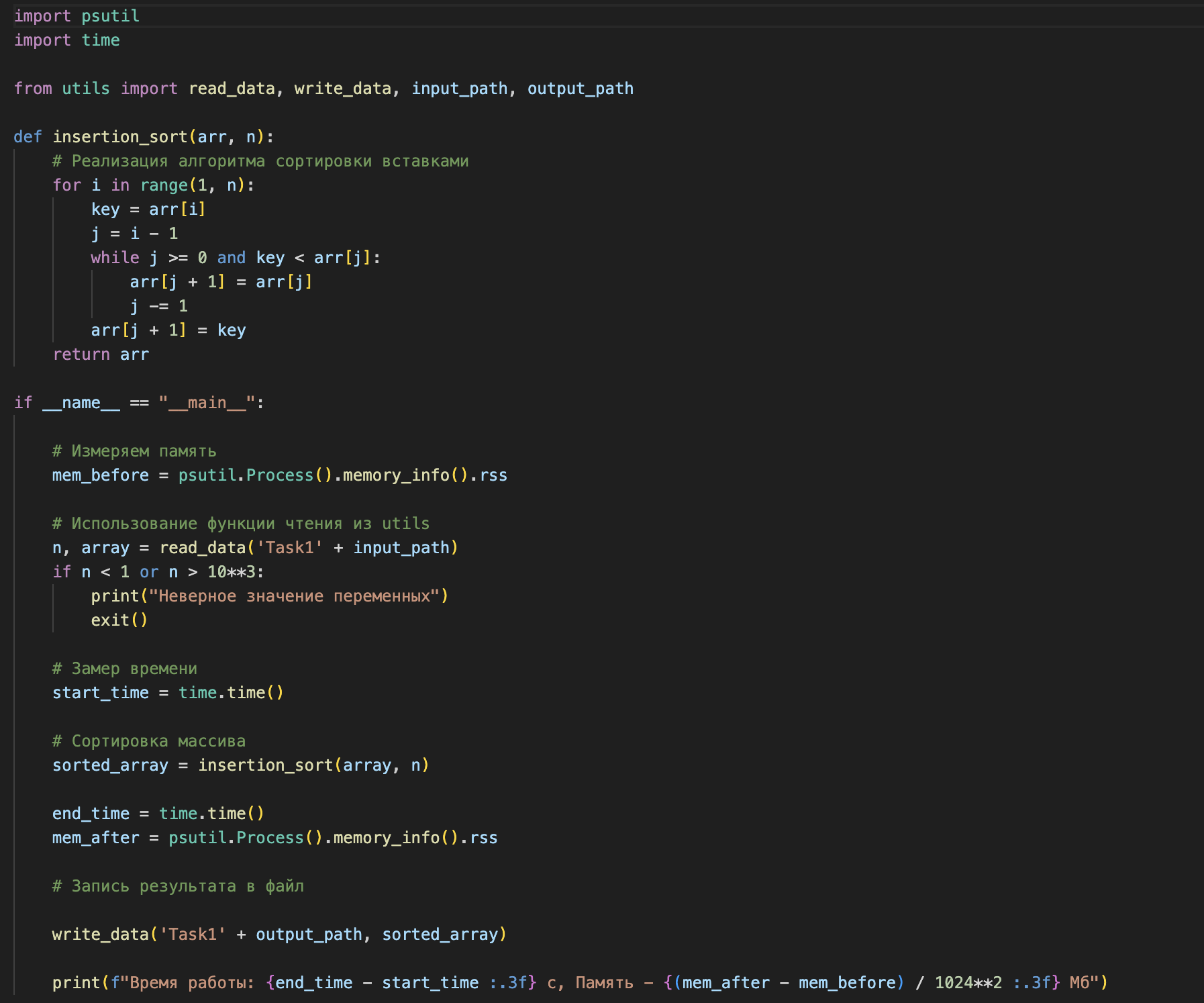
Задание №1.Сортировка вставкой.



Для автоматизации кода использовала utils.py



Решение задачи №1:



Этот код реализует алгоритм сортировки вставками (insertion\_sort) для сортировки массива целых чисел, который читается из файла input.txt. Результат сортировки записывается в файл output.txt. Кроме того, код измеряет время работы алгоритма и потребление памяти.



* time - модуль, предоставляющий функции для работы с временем.
* psutil - модуль, предоставляющий функции для работы с системными ресурсами, такими как память.
* Вычисление времени :  
   start\_time = time.time()

end\_time = time.time()

end\_time - start\_time

Вычисление памяти:

mem\_before = psutil.Process().memory\_info().rss

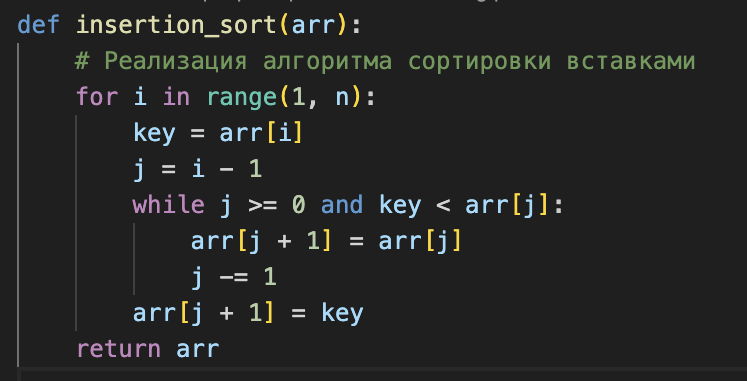
mem\_after = psutil.Process().memory\_info().rss

mem\_after - mem\_before



Считываем данные из файлов с помощью utils.py

**Определение функции сортировки:**

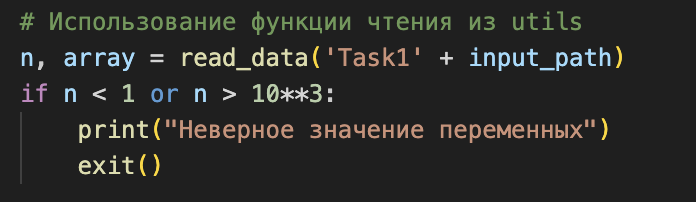


* Функция insertion\_sort принимает массив arr и сортирует его в порядке возрастания.

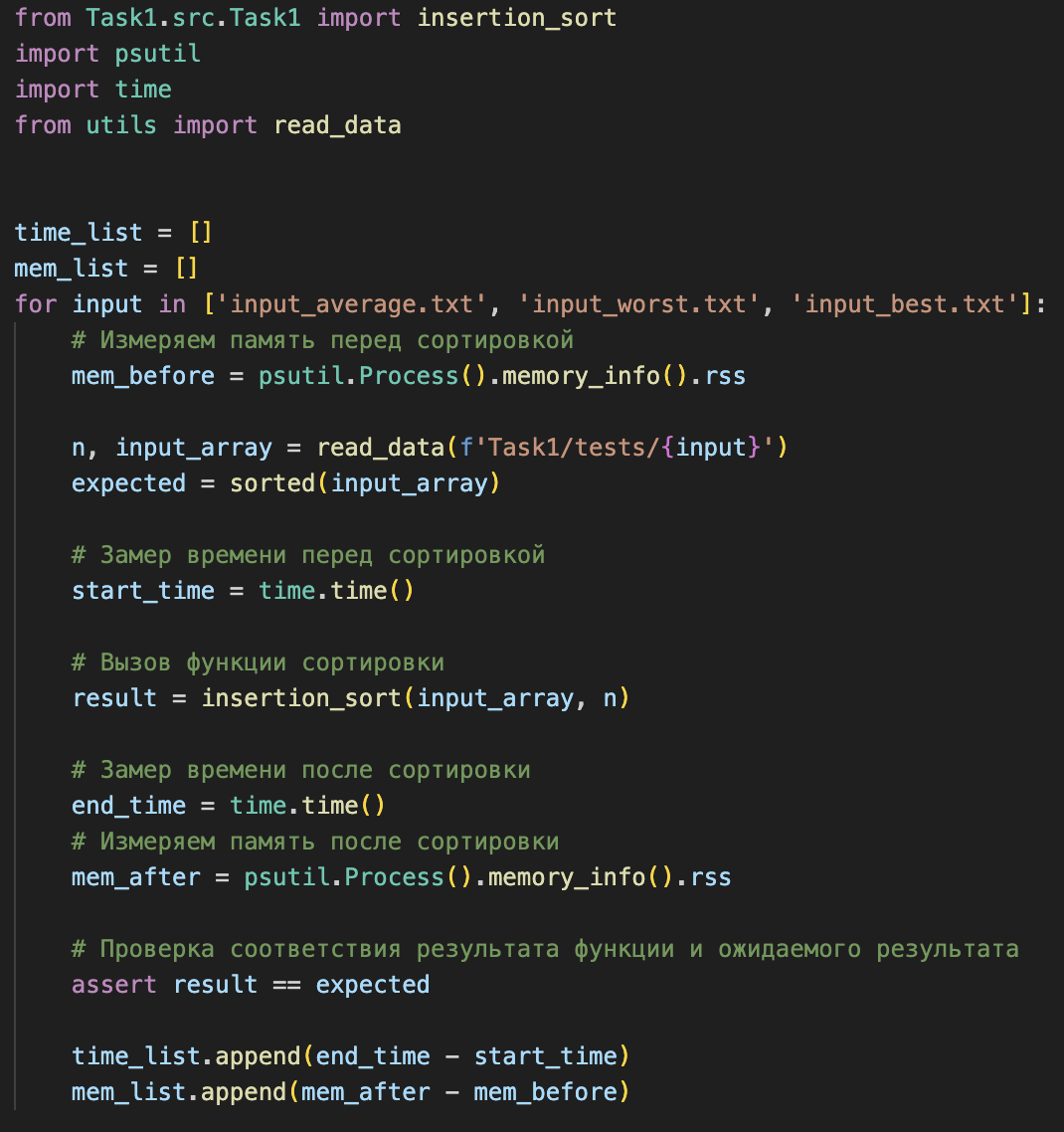
Алгоритм работает следующим образом:

* Цикл начинается со второго элемента массива (i = 1), так как первый элемент уже отсортирован (он единственный элемент в массиве).
* Цикл продолжается до n-1 элемента, где n - длина массива.
* На каждой итерации цикла выбирается текущий элемент массива (arr[i]) как ключевой элемент (key).
* Внутри цикла while мы начинаем с предыдущего элемента (j = i - 1) и движемся влево, пока не найдем правильную позицию для ключевого элемента.
* Если текущий элемент (arr[j]) больше ключевого элемента (key), мы сдвигаем его вправо, копируя его значение в следующую ячейку (arr[j + 1] = arr[j]).
* Мы продолжаем это процесс, пока не найдем элемент, который меньше или равен ключевому элементу, или пока не достигнем начала массива (j >= 0).
* Как только мы нашли правильную позицию для ключевого элемента, мы вставляем его в массив, присваивая его значение ячейке arr[j + 1].
* После того, как цикл завершен, функция возвращает отсортированный массив.

В целом, алгоритм сортировки вставками работает, последовательно просматривая элементы массива и вставляя каждый элемент в правильную позицию в уже отсортированной части массива.

****

Написала тест для проверки (учитывала наилучший, средний, наихудший случаи):

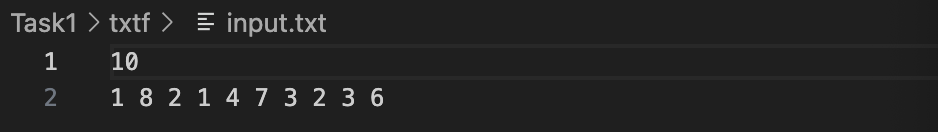
****

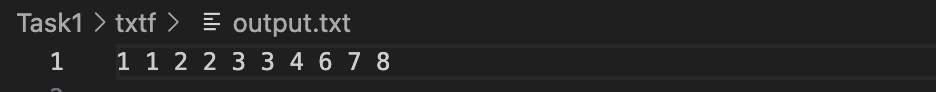
**Основные этапы работы теста:**

1. Загрузка входных данных из файлов.
2. Измерение времени и памяти перед сортировкой.
3. Вызов функции сортировки вставками.
4. Измерение времени и памяти после сортировки.
5. Проверка корректности результата сортировки.
6. Сохранение результатов измерений времени и памяти.
7. Вывод результатов измерений времени и памяти для каждого файла.

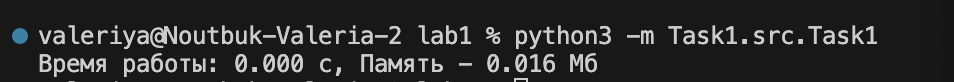
Затем записываем результат в файл.

**Файлы input.txt и output.txt:**

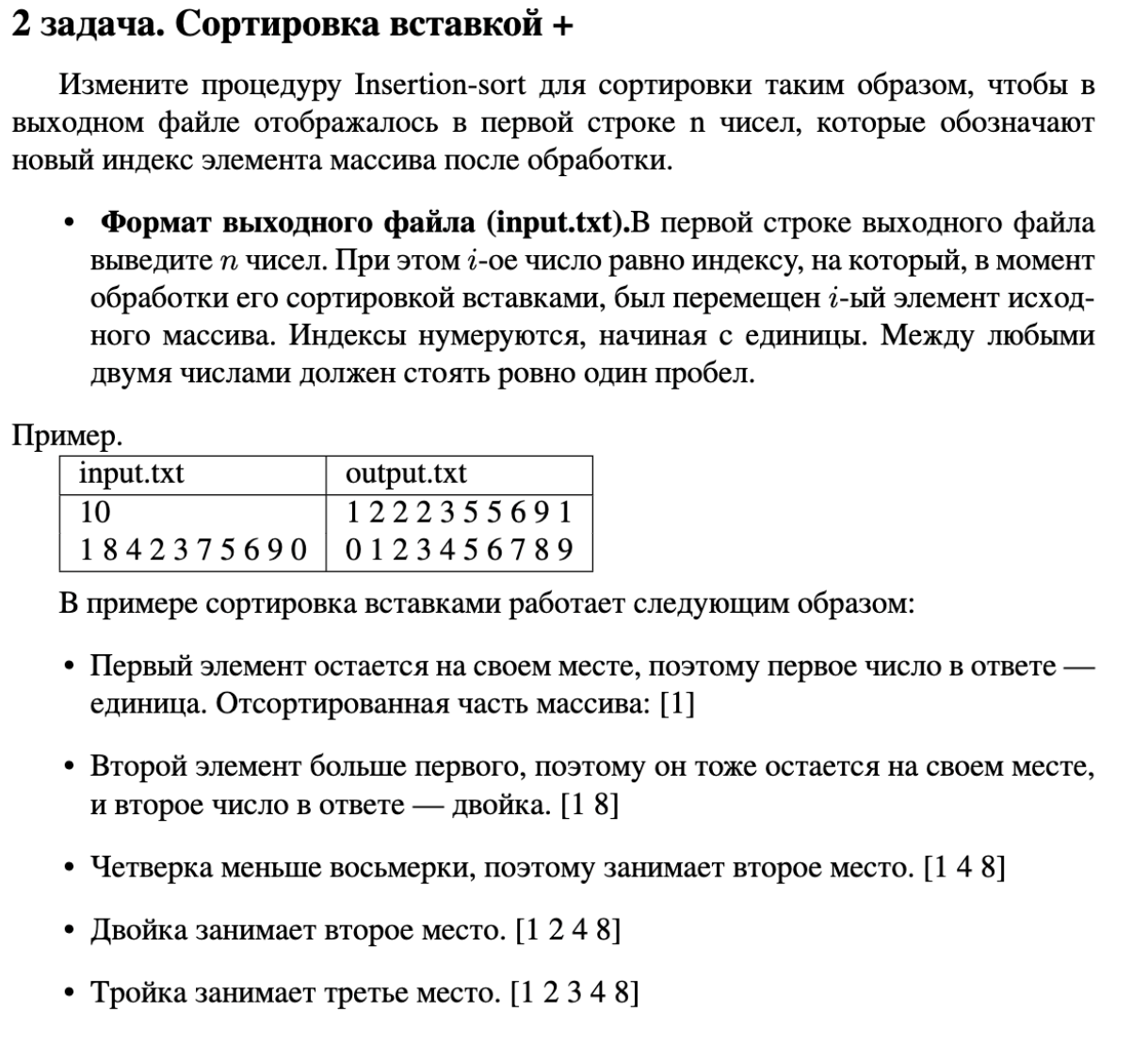




**Время и количество памяти:**



Задача №2: Сортировка вставкой +



Решение задачи №2:

  
Этот код реализует алгоритм сортировки вставками с отслеживанием позиций перемещения элементов



Модуль time используется для измерения времени выполнения программы.

Модуль psutil используется для измерения потребления памяти программой.

* Вычисление времени :  
   start\_time = time.time()

end\_time = time.time()

end\_time - start\_time

Вычисление памяти:

mem\_before = psutil.Process().memory\_info().rss

mem\_after = psutil.Process().memory\_info().rss

mem\_after - mem\_before



**Определение функции сортировки:**

def insertion\_sort\_with\_position\_tracking(arr):- определяется функция insertion\_sort\_with\_position\_tracking, которая принимает массив arr в качестве аргумента.

**Инициализация массива для отслеживания позиций:**

positions = [1] \* n

* Создается массив positions длиной n, где n - количество элементов в массиве arr.
* Массив positions используется для отслеживания новых индексов для каждого элемента после сортировки.

**Основной алгоритм сортировки вставками:**



Этот код реализует алгоритм сортировки вставками.

* На каждой итерации цикла for выбирается текущий элемент key из массива arr.
* Внутри цикла while элементы массива arr сдвигаются вправо, если они больше ключевого элемента key.
* После сдвига элементов ключевой элемент key вставляется в правильную позицию в массиве arr.

**Отслеживание позиций:**

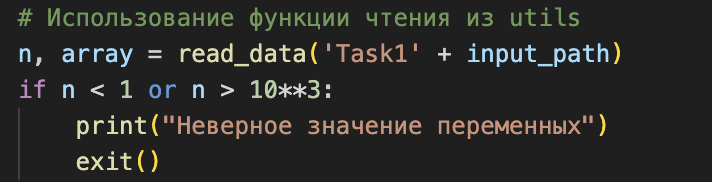
positions[i] = j + 2

* После вставки ключевого элемента в правильную позицию его новый индекс записывается в массив positions.
* Индекс увеличивается на 1, чтобы соответствовать задаче.

return positions, arr - Функция возвращает два массива: positions и arr.

* Массив positions содержит новые индексы для каждого элемента после сортировки.
* Массив arr содержит отсортированный массив.

**Чтение данных из utils:**



Первая строка файла содержит количество элементов n.

Вторая строка файла содержит массив чисел, который конвертируется в список arr.

Вызов функции сортировки:

positions, sorted\_arr = insertion\_sort\_with\_position\_tracking(arr)

* Функция insertion\_sort\_with\_position\_tracking вызывается с массивом arr в качестве аргумента.
* Результаты функции записываются в переменные positions и sorted\_arr.

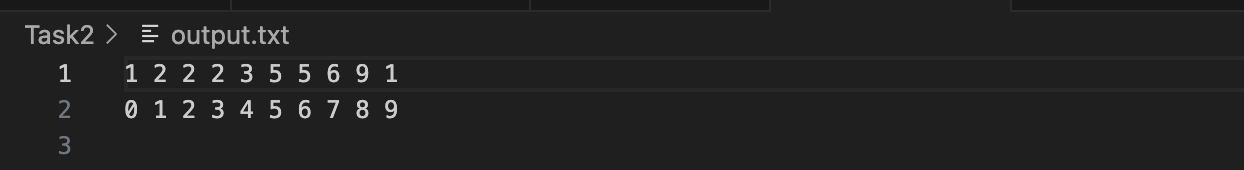
**Запись результатов в файл:**

****

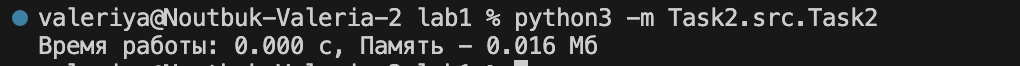
* Результаты записываются в файл output.txt.
* Первая строка файла содержит массив индексов positions.
* Вторая строка файла содержит отсортированный массив sorted\_arr.

**Файлы input.txt и output.txt:**

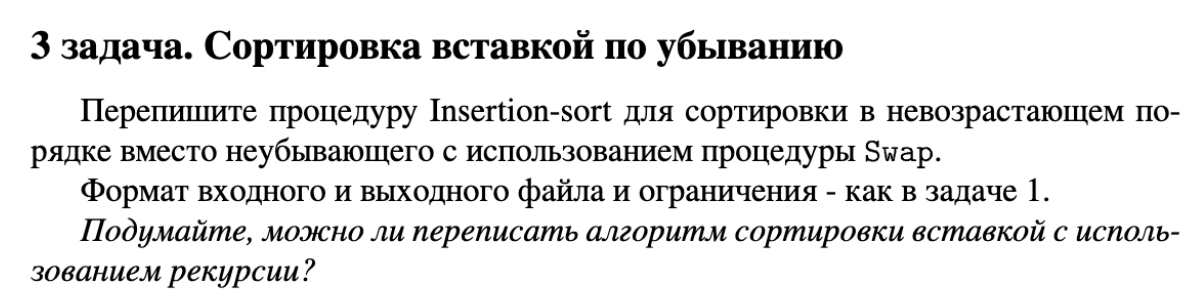
****

****

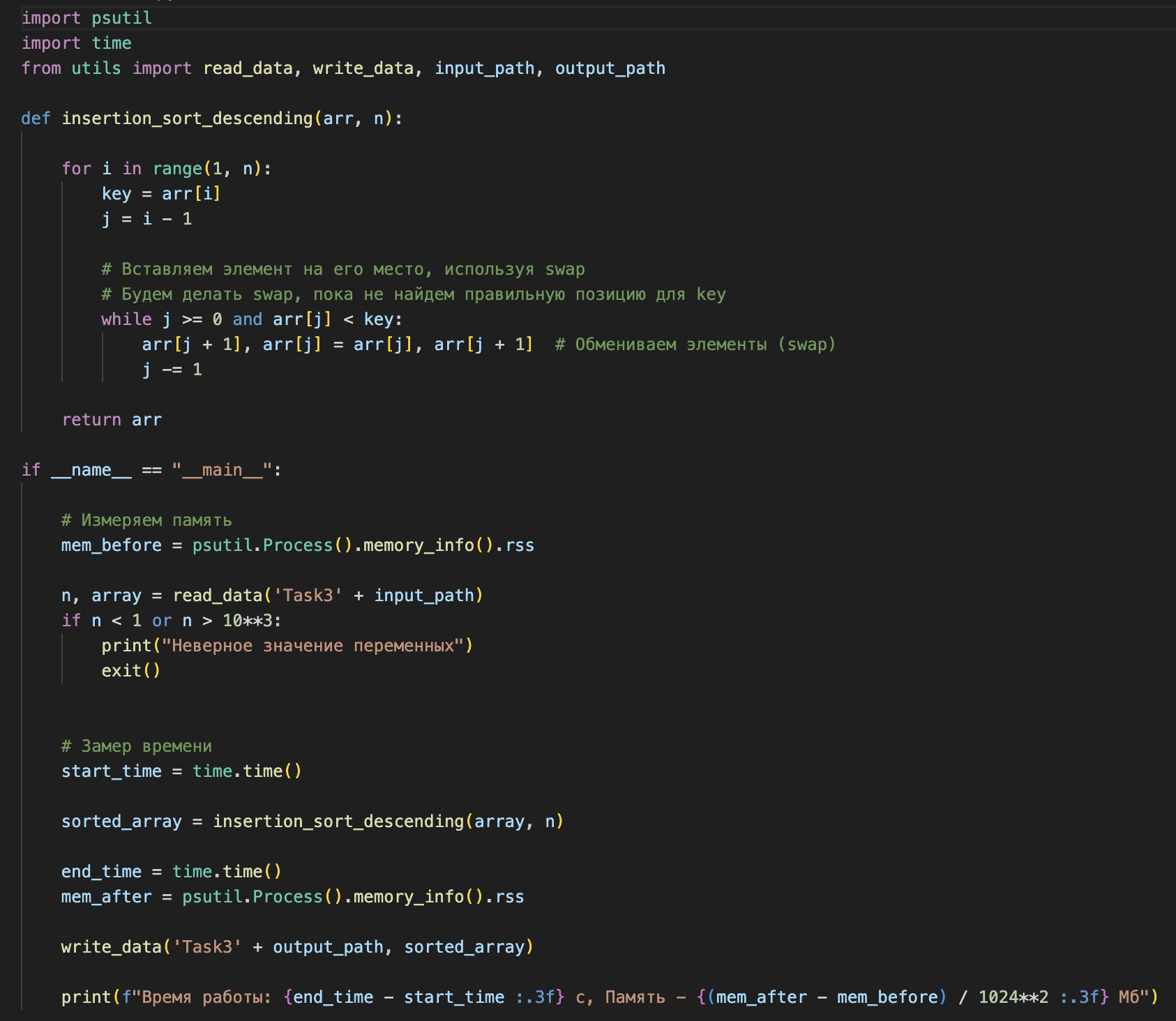
**Время и количество памяти:**

****

Задача №3:Сортировка вставкой по убыванию



Решение задачи №3:



Этот код реализует алгоритм сортировки вставками в порядке убывания

****

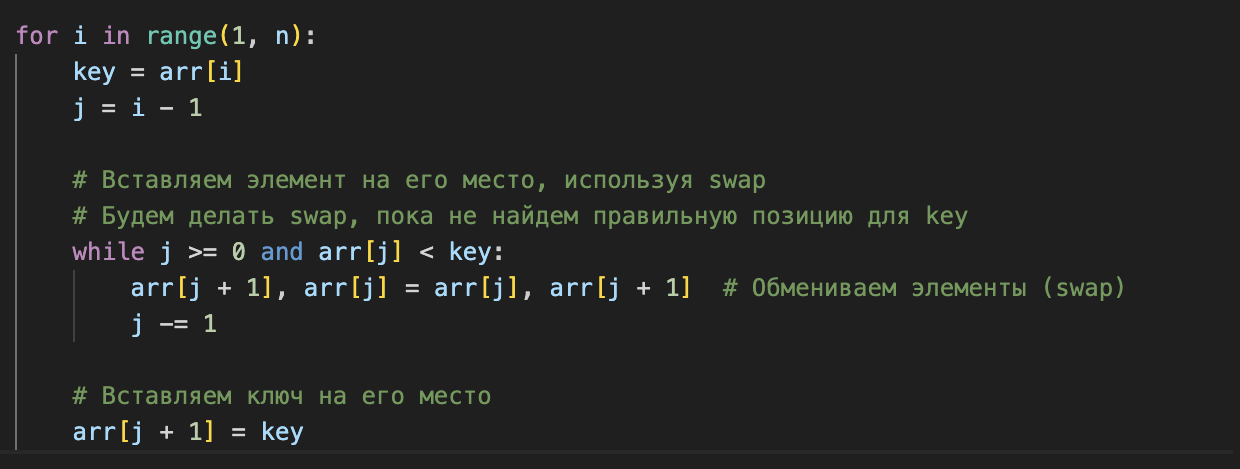
Модуль time используется для измерения времени выполнения программы.

Модуль psutil используется для измерения потребления памяти программой.

**Определение функции сортировки:**

def insertion\_sort\_descending(arr): - Определяется функция insertion\_sort\_descending, которая принимает массив arr в качестве аргумента.

**Основной алгоритм сортировки вставками:**

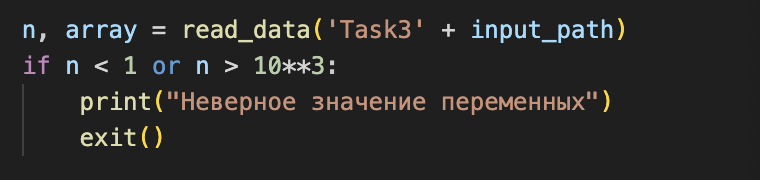
****

* Этот код реализует алгоритм сортировки вставками в порядке убывания.
* На каждой итерации цикла for выбирается текущий элемент key из массива arr.
* Внутри цикла while элементы массива arr обмениваются местами (swap), если они меньше ключевого элемента key.
* После обмена элементами ключевой элемент key вставляется в правильную позицию в массиве arr.

**Возвращение результата:**

return arr - Функция возвращает отсортированный массив arr.

**Чтение данных файла из utils:**

****

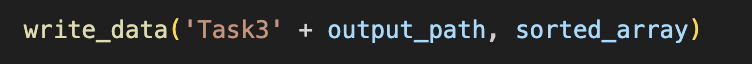
* Данные читаются из файла input.txt.
* Первая строка файла содержит количество элементов n.
* Если количество элементов не удовлетворяет условиям (n < 1 или n > 10\*\*3), выводится сообщение об ошибке и программа завершается.
* Вторая строка файла содержит массив чисел, который конвертируется в список array.

**Вызов функции сортировки:**

sorted\_array = insertion\_sort\_descending(array, n)

* Функция insertion\_sort\_descending вызывается с массивом array в качестве аргумента.
* Результат функции записывается в переменную sorted\_array.

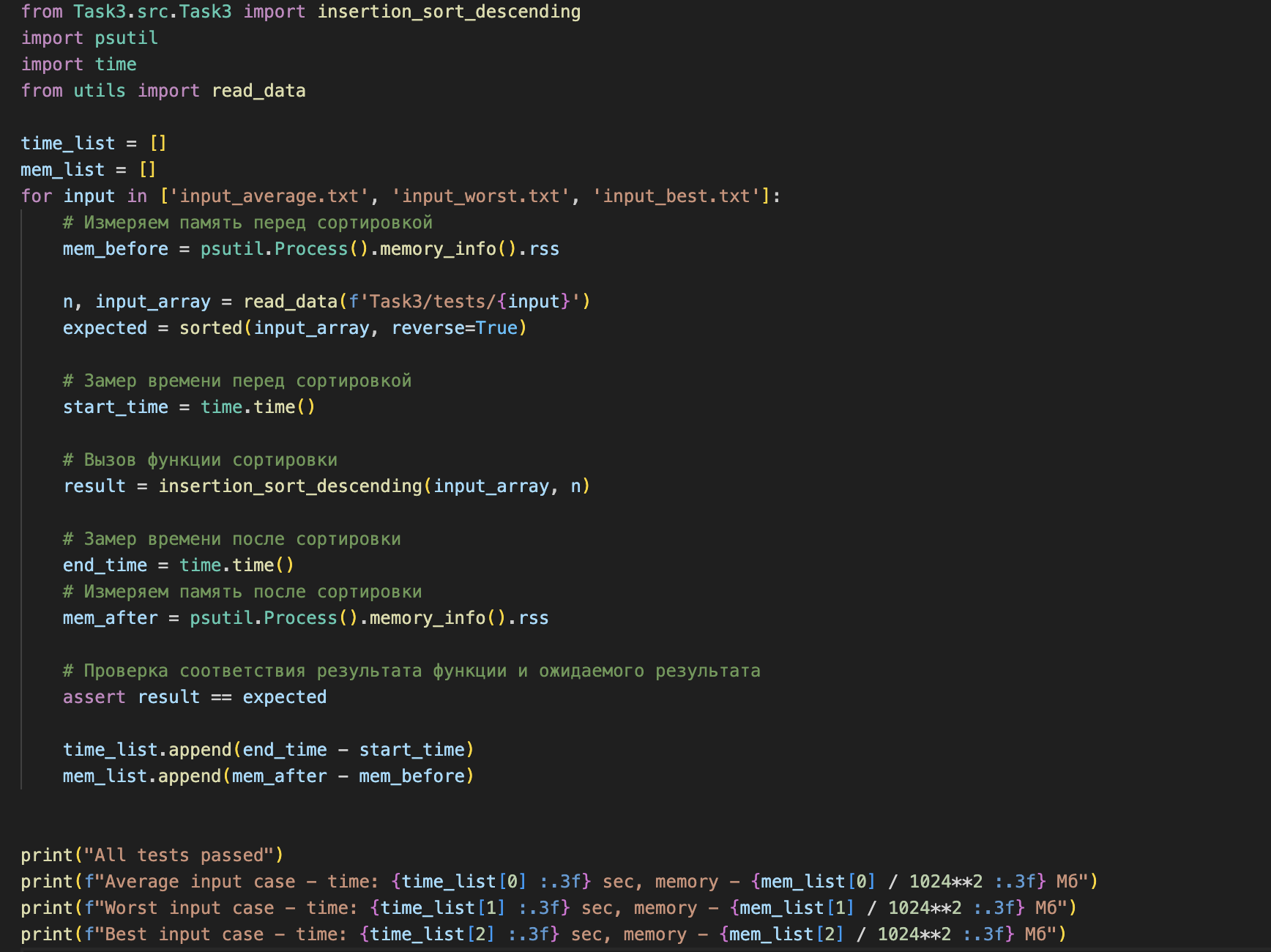
**Запись результата в файл и Вывод времени выполнения и потребления памяти:**

****

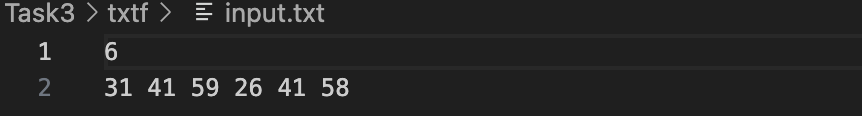
****

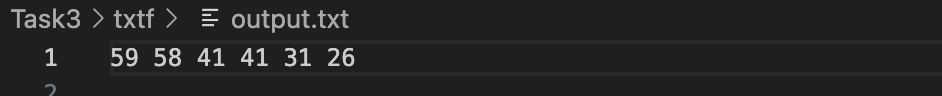
* Результат записывается в файл output.txt.
* Массив sorted\_array конвертируется в строку и записывается в файл.
* Время выполнения программы рассчитывается как разница между текущим временем и временем запуска таймера.
* Потребление памяти программой рассчитывается с помощью модуля psutil.
* Результаты выводятся на экран.

Написала тест для проверки (учитывала наилучший, средний, наихудший случаи):



**Файлы input.txt и output.txt:**

****

****

**Время и количество памяти:**

****

Можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Базовый случай: Если количество элементов n меньше или равно 1, массив уже отсортирован, и функция просто возвращается.

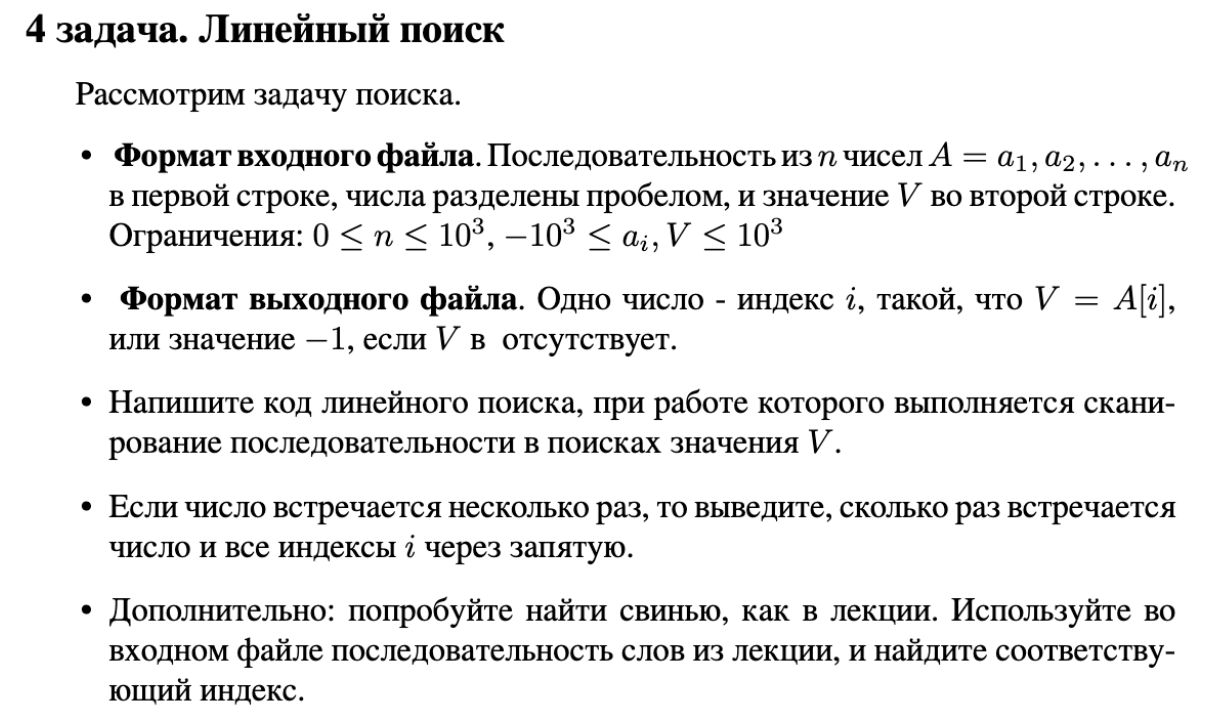
Рекурсивный вызов: Функция вызывает саму себя для сортировки первых n-1 элементов.

Вставка элемента: После сортировки первых n-1 элементов последний элемент вставляется на нужную позицию в отсортированной части массива:

Мы сравниваем последний элемент с предыдущими элементами, и если они меньше, то сдвигаем их вправо.

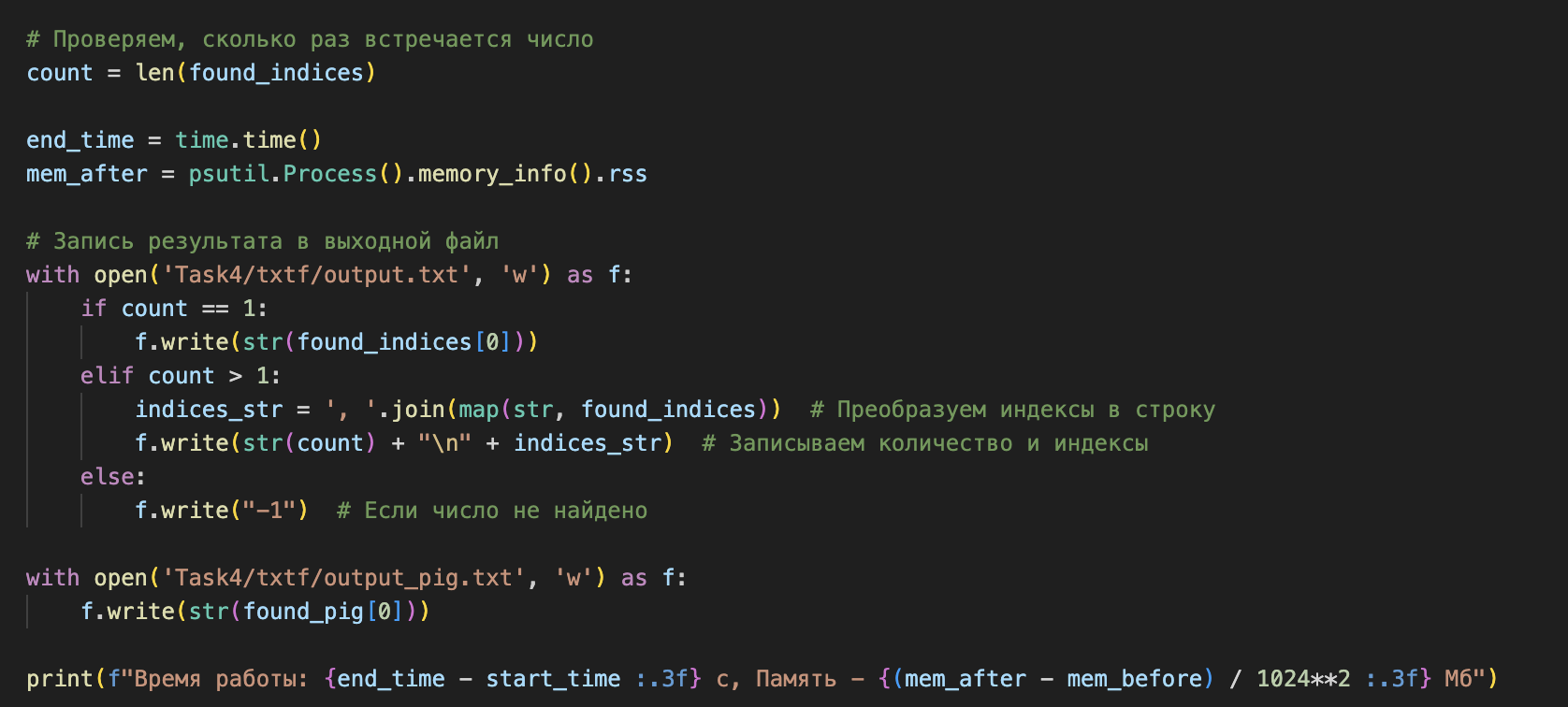
Затем вставляем последний элемент в правильную позицию.

Задача №4:Линейный поиск



Решение задачи №4:





Этот код реализует линейный поиск целевого числа в массиве чисел.

Здесь, как в предыдущих трех задачах, использую модули import time и import psutil.

**Определение функции линейного поиска:**

def linear\_search(arr, target):

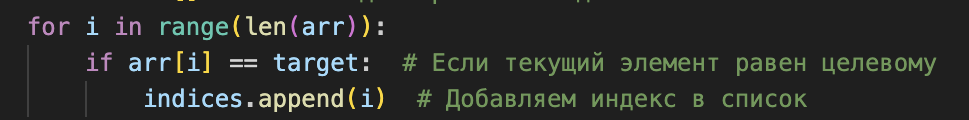
* Определяется функция linear\_search, которая принимает массив arr и целевое число target в качестве аргументов.

**Инициализация списка для хранения индексов:**

indices = [] # Список для хранения индексов

* Создается пустой список indices для хранения индексов, где находится целевое число в массиве.

**Линейный поиск:**

****

* Производится линейный поиск целевого числа в массиве.
* Если текущий элемент равен целевому числу, его индекс добавляется в список indices.

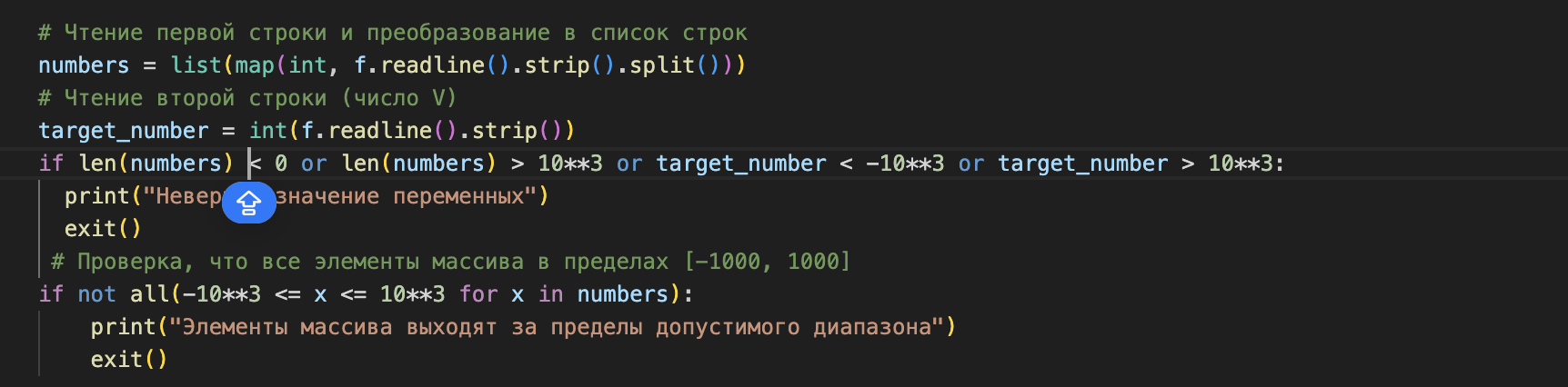
**Возвращение списка индексов:**

return indices # Возвращаем список индексов

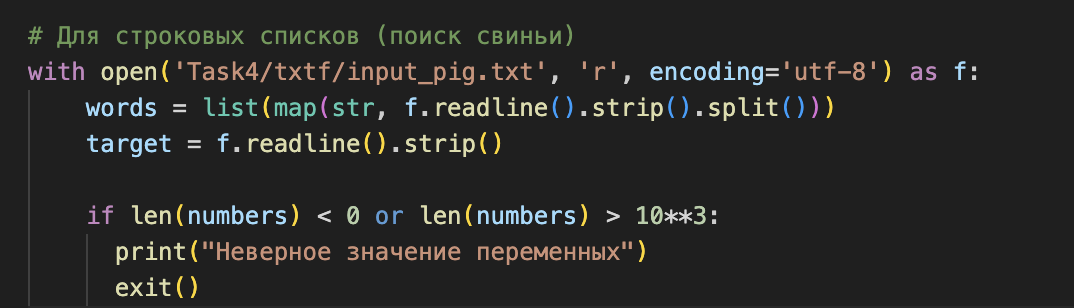
* Функция возвращает список индексов, где находится целевое число в массиве.

**Чтение данных из входного файла и проверка корректности :**

* Данные читаются из файла input.txt.
* Первая строка файла содержит массив чисел, который преобразуется в список numbers.
* Вторая строка файла содержит целевое число target\_number.
* Проверяется корректность входных данных.
* Если длина массива или целевое число выходят за пределы допустимого диапазона, выводится сообщение об ошибке и программа завершается.



**Потом считывания данных для поиска свиньи:**



**Поиск индексов целевого числа:**

found\_indices = linear\_search(numbers, target\_number)

* Функция linear\_search вызывается с массивом numbers и целевым числом target\_number в качестве аргументов.
* Результат функции записывается в переменную found\_indices.

found\_pig = linear\_search(words, target)

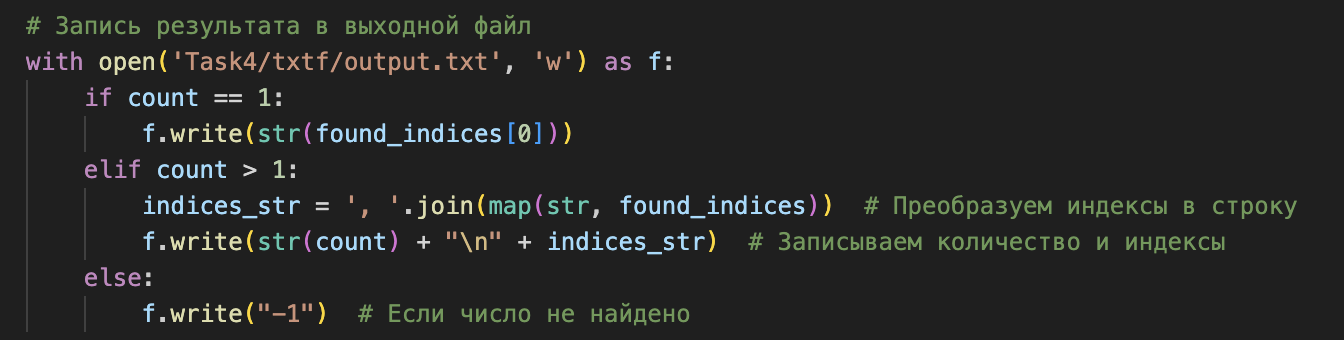
Функция выполняет аналогичную работу, но только для слов.

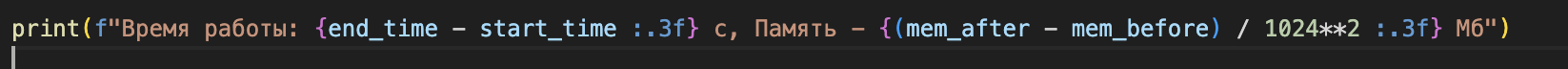
**Проверка количества найденных индексов:**

count = len(found\_indices)

* Проверяется количество найденных индексов.
* Если количество найденных индексов равно 1, записывается только индекс.
* Если количество найденных индексов больше 1, записывается количество и индексы.
* Если количество найденных индексов равно 0, записывается "-1".

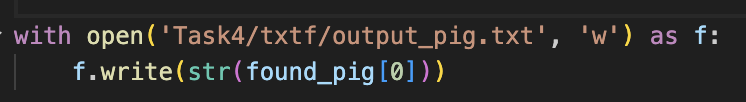
**Запись результата в выходной файл и вывод времени выполнения и потребления памяти:**



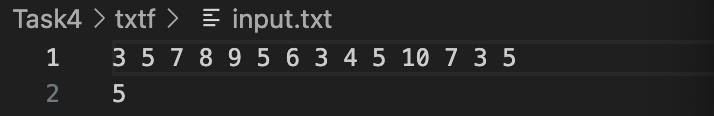


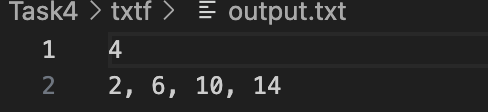
* Результат записывается в файл output.txt.
* Время выполнения программы рассчитывается как разница между текущим временем и временем запуска таймера.
* Потребление памяти программой рассчитывается с помощью модуля psutil.
* Результаты выводятся на экран.

Запись вывода программы в output:



**Файлы input.txt и output.txt:**

****

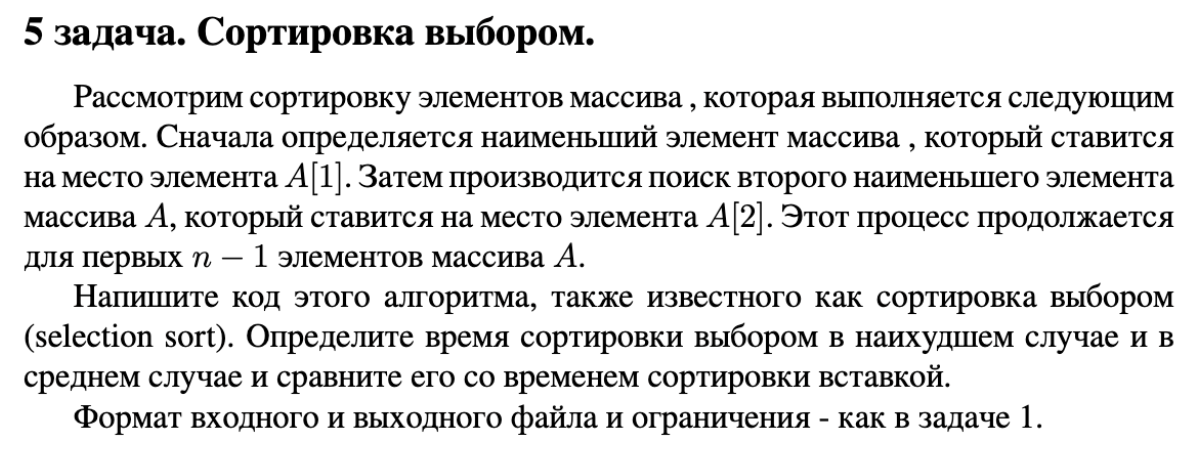
****

**Для свиньи не могу приложить скрин, так input.txt очень большой и не помещается .**

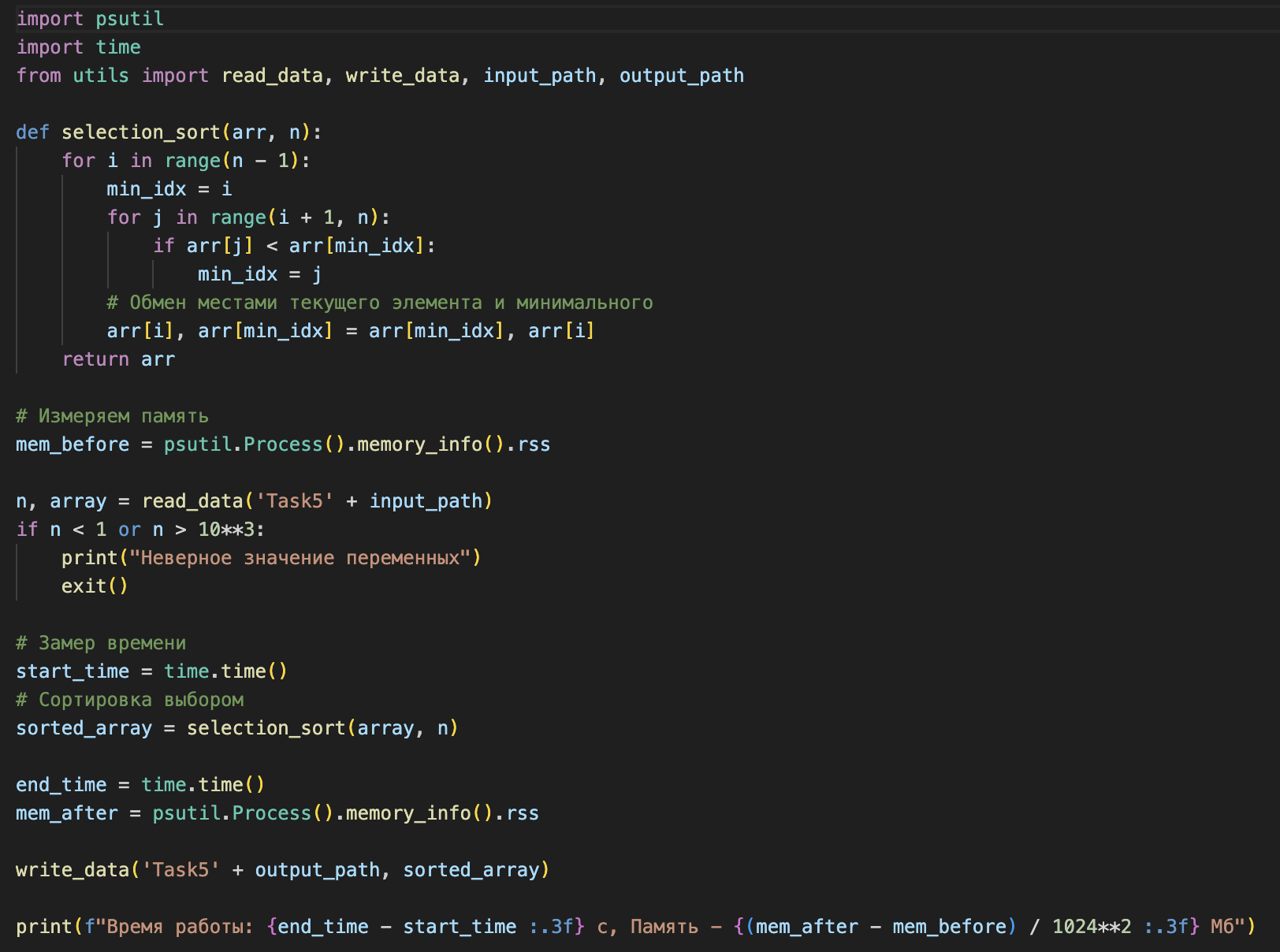
**Время и количество памяти:**

****

Задача №5:Сортировка выбором



Решение задачи №5:



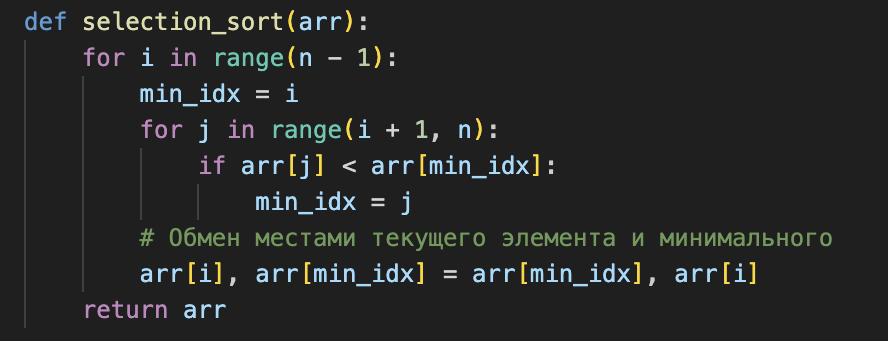
Данный код на Python реализует сортировку выбором (selection sort) для массива целых чисел, который читается из файла input.txt. После сортировки результат записывается в файл output.txt. Кроме того, код измеряет время выполнения программы и количество используемой памяти.

**Импорт модулей**

В начале кода импортируются два модуля:

* time: для измерения времени выполнения программы
* psutil: для измерения количества используемой памяти

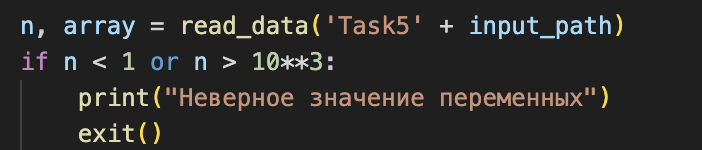
**Функция сортировки выбором:**



Эта функция принимает на вход массив arr и выполняет сортировку выбором. Алгоритм работает следующим образом:

1. Внешний цикл for проходит по массиву от первого элемента до предпоследнего (n - 1).
2. Внутренний цикл for проходит по массиву от текущего элемента (i + 1) до последнего элемента (n).
3. Если элемент arr[j] меньше текущего минимального элемента arr[min\_idx], то индекс минимального элемента обновляется (min\_idx = j).
4. После внутреннего цикла текущий элемент и минимальный элемент меняются местами (arr[i], arr[min\_idx] = arr[min\_idx], arr[i]).
5. Функция возвращает отсортированный массив.

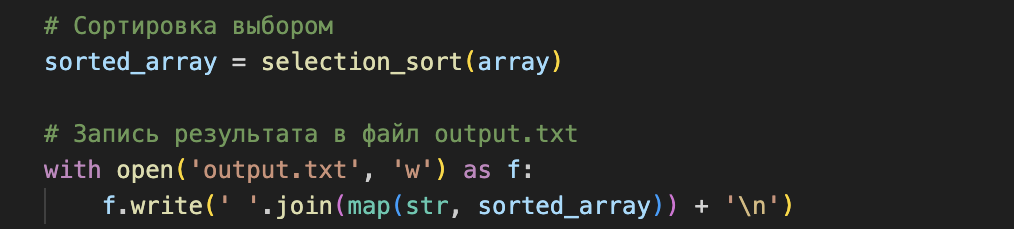
**Чтение данных файла из utils:**



Этот код читает данные из файла input.txt:

1. Первая строка файла содержит количество элементов в массиве (n). Если это значение не удовлетворяет условиям (n < 1 или n > 10\*\*3), то программа выводит ошибку и завершается.
2. Вторая строка файла содержит массив целых чисел, который читается и преобразуется в список (array).

**Сортировка и запись результата:**

****

****

Этот код сортирует массив array с помощью функции selection\_sort и записывает результат в файл output.txt.

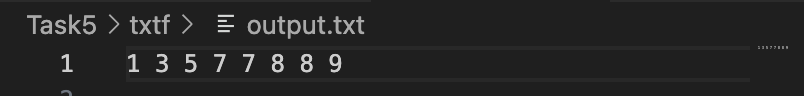
**Измерение времени и памяти:**

****

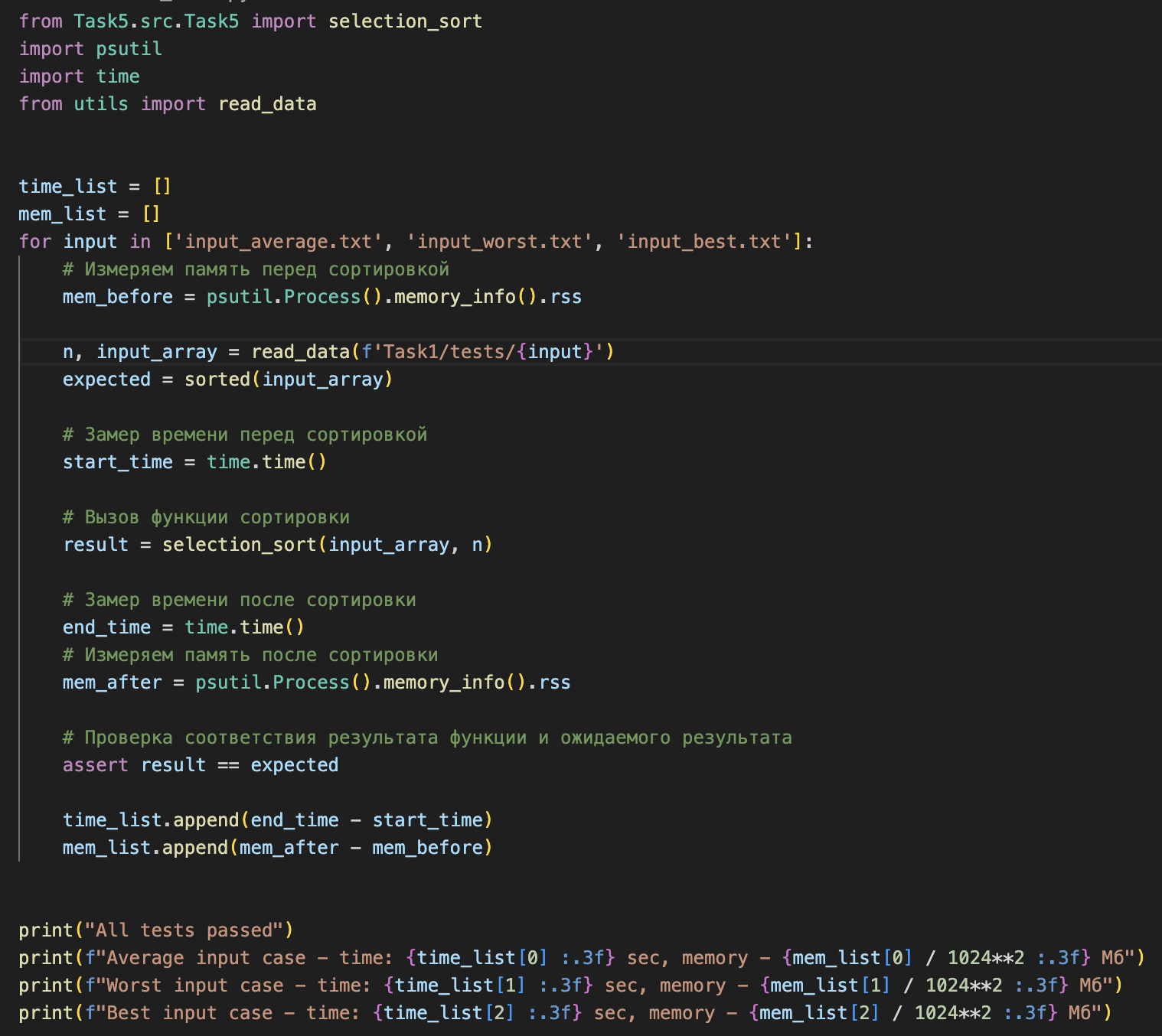
Этот код измеряет время выполнения программы с помощью модуля psutil. Результаты выводятся на экран.

**Файлы input.txt и output.txt:**

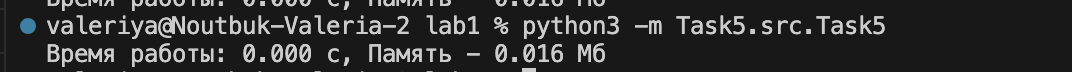
****

****

Написала тест для проверки (учитывала наилучший, средний, наихудший случаи):

****

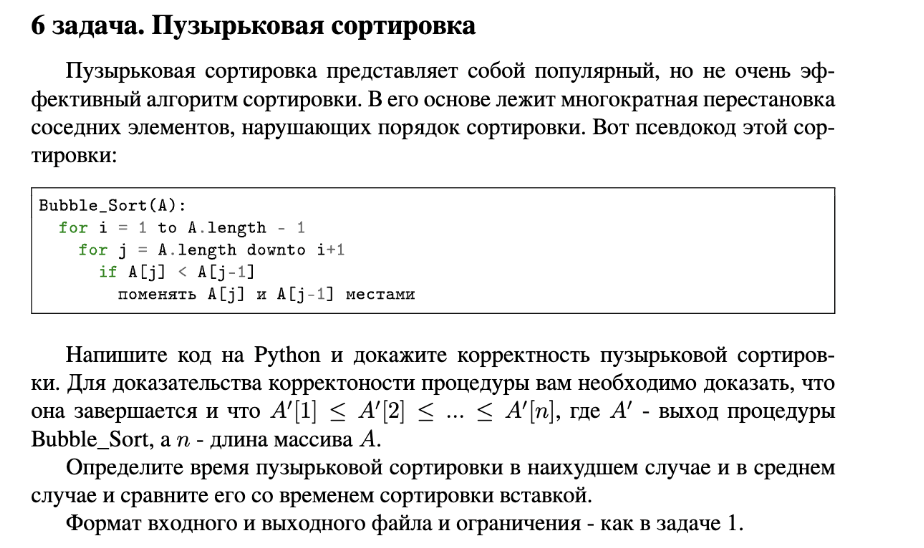
**Время и количество памяти:**

****

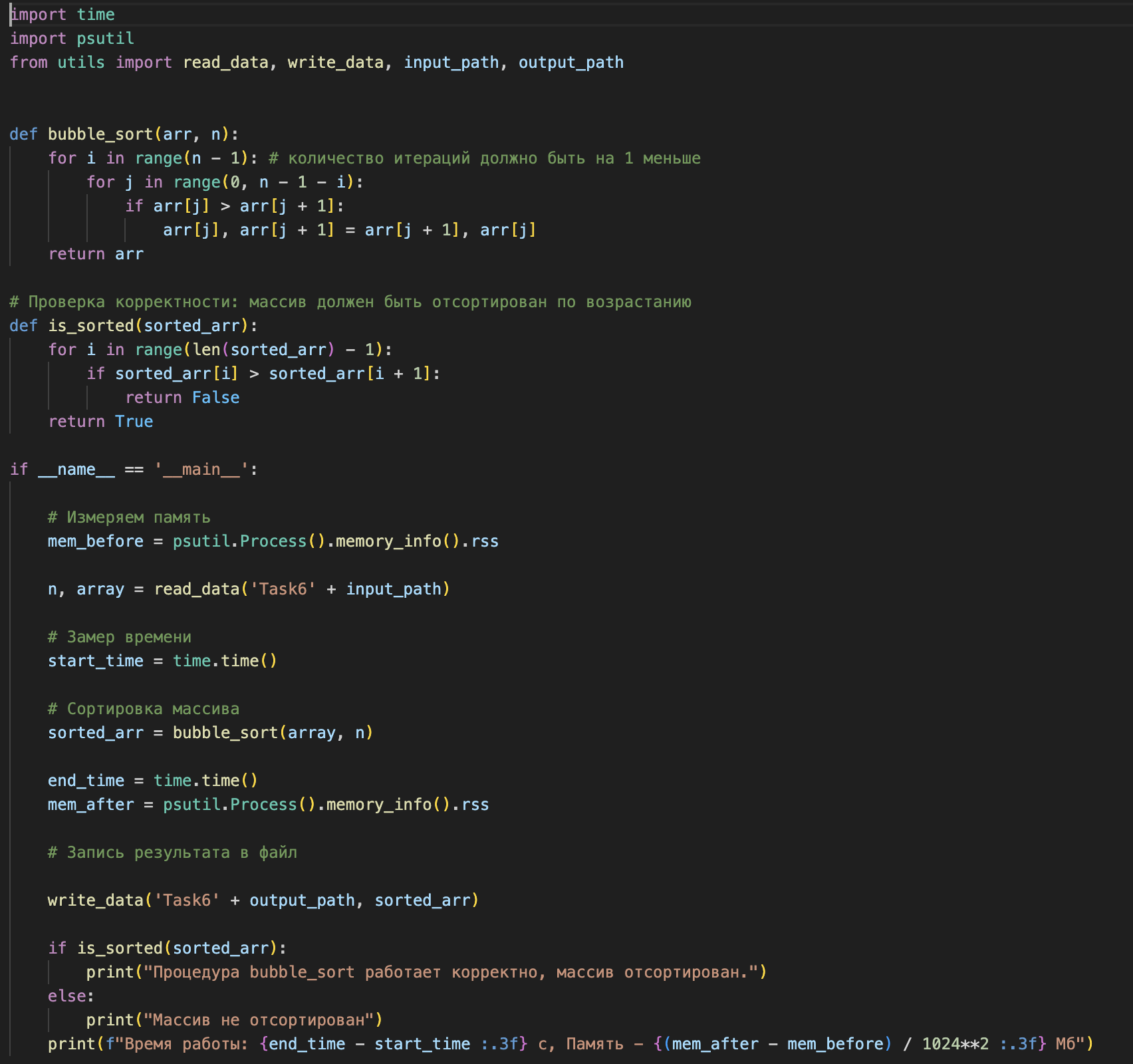
**Сравнение времени сортировкой вставкой и сортировкой выбором:**

O(n^2) - квадратичная функция от n, где n - размер массива (для наихудшего случая сортировки вставкой и сортировки выбором).В наилучшем случае O(n) - для сортировки вставкой, сортировка выбором - не меняется, остается O(n^2).

Задача №6:Пузырьковая сортировка



Решение задачи №6:



Этот код представляет собой реализацию алгоритма сортировки пузырьком (bubble sort) и функцию проверки корректности отсортированного массива.

****

Эта функция принимает на вход массив arr и его длину n. Функция сортирует массив по возрастанию, используя алгоритм пузырьком.

Алгоритм работает следующим образом:

1. Внешний цикл for i in range(n - 1): выполняется n-1 раз, где n - длина массива.
2. Внутренний цикл for j in range(0, n - 1 - i): выполняется n-1-i раз, где i - текущая итерация внешнего цикла.
3. Внутри внутреннего цикла проверяется, если текущий элемент arr[j] больше следующего элемента arr[j + 1]. Если это так, то элементы меняются местами.
4. После окончания внутреннего цикла, наибольший элемент в массиве "всплывает" в конец массива.
5. Внешний цикл повторяется, пока не будут отсортированы все элементы массива.

****

Эта функция принимает на вход отсортированный массив sorted\_arr. Функция проверяет, если массив действительно отсортирован по возрастанию.

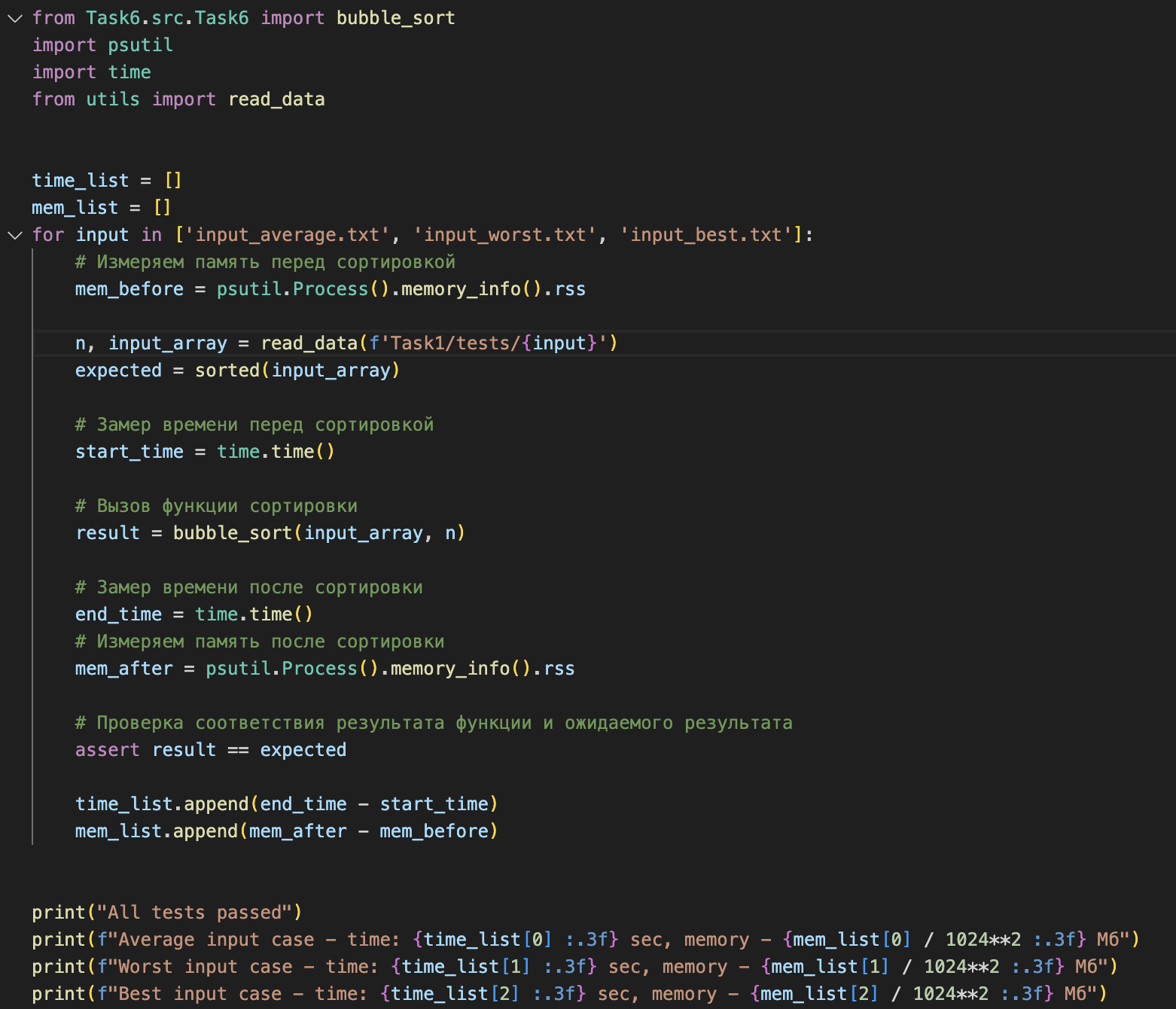
Алгоритм работает следующим образом:

1. Цикл for i in range(len(sorted\_arr) - 1): выполняется len(sorted\_arr) - 1 раз, где len(sorted\_arr) - длина массива.
2. Внутри цикла проверяется, если текущий элемент sorted\_arr[i] больше следующего элемента sorted\_arr[i + 1]. Если это так, то функция возвращает False, указывая на то, что массив не отсортирован.
3. Если цикл завершается без обнаружения несортированных элементов, то функция возвращает True, указывая на то, что массив отсортирован.

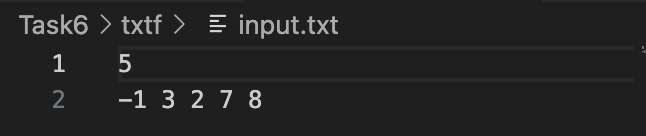
****

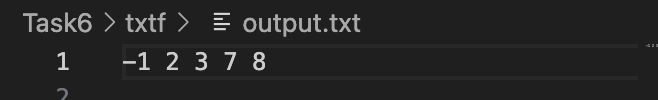
Здесь я считываю данные и записываю их в output.Также произвожу подсчет времени и памяти.

Написала тест для проверки (учитывала наилучший, средний, наихудший случаи):

****

**Файлы input и output:**

****

****

**Вывод:** Цель лабораторной работы достигнута. А именно, я познакомилась и отработала различные виды сортировки : сортировку вставкой, сортировку вставкой +, сортировку вставкой по убыванию, линейный поиск, сортировку выбором. При выполнении работы столкнулась с трудностями, но к счастью, успешно их решила.