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

Вариант 5

Выполнил:

Артемов И. В.

К3141

Проверил:

Афанасьев А. В.

Санкт-Петербург

2024 г.

# Содержание отчета

[**Содержание отчета**](#_10udeakjagvs) **2**

[**Задачи по варианту**](#_gf7yxvsg0zb) **3**

Задача №1. Сортировка вставкой 3

Задача №2. Сортировка вставкой + 6

Задача №8. Секретарь Своп 9

[**Дополнительные задачи**](#_inlxofidblo4) **13**

Задача №3. Сортировка вставкой по убыванию 13

Задача №5. Сортировка выбором 16

Задача №6. Пузырьковая сортировка 19

[**Вывод**](#_fu90fuyk873) **23**

# Задачи по варианту

## Задача №1. Сортировка вставкой

Текст задачи.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Листинг кода.

import tracemalloc  
import time  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
if (1 <= n <= 10\*\*3) and (all(abs(i) <= 10\*\*9 for i in m)):  
 for i in range(1, len(m)):  
 key = m[i]  
 j = i - 1  
 while j >= 0 and m[j] > key:  
 m[j + 1] = m[j]  
 j -= 1  
 m[j + 1] = key  
 m\_sorted = ' '.join(map(str, m))  
 f\_output = open('output.txt', 'w')  
 f\_output.write(m\_sorted)  
else:  
 print('Введите корректные данные')  
print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Далее сам алгоритм сортировки вставкой: для каждого элемента (начиная со второго, так как в алгоритме предполагается, что первый элемент уже отсортирован) ищется его правильное место среди предыдущих элементов массива; в процессе сравнения элементы “сдвигаются” пока не найдётся подходящая позиция для текущего элемента (key)
4. Отсортированный массив преобразуем в строку через пробелы и записываем результат в output.txt

Результат работы кода на примере из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

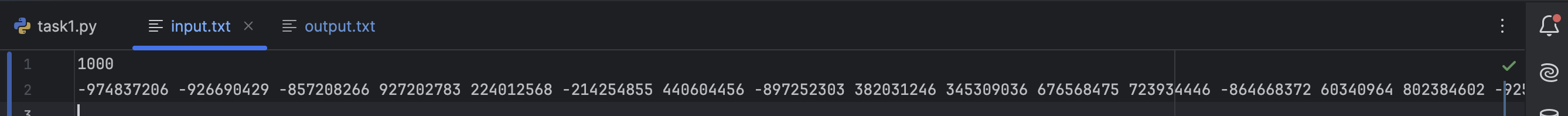
Результат работы кода на максимальных и минимальных значениях:

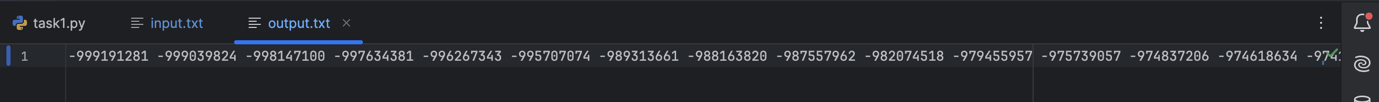
Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

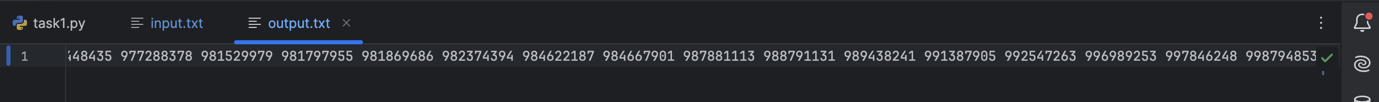
Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание





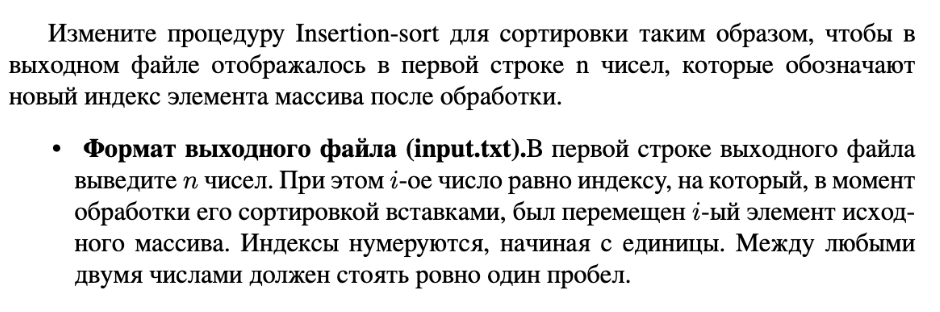


|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00030233399593271315 секунд | 13863 байт |
| Пример из задачи | 0.0002968749904539436 секунд | 13875 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.367432708007982 секунд | 116128 байт |

Вывод по задаче: мною был изучен алгоритм сортировки массива вставкой, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти.

## Задача №2. Сортировка вставкой +

Текст задачи.



Листинг кода.

import tracemalloc  
import time  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
indices = [1]  
if (1 <= n <= 10\*\*3) and (all(abs(i) <= 10\*\*9 for i in m)):  
 for i in range(1, len(m)):  
 key = m[i]  
 j = i - 1  
 while j >= 0 and m[j] > key:  
 m[j + 1] = m[j]  
 j -= 1  
 m[j + 1] = key  
 indices.append(j + 2)  
 indices = ' '.join(map(str, indices))  
 m\_sorted = ' '.join(map(str, m))  
 f\_output = open('output.txt', 'w')  
 f\_output.write(f'{indices}\n')  
 f\_output.write(m\_sorted)  
else:  
 print('Введите корректные данные')  
print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Далее сам алгоритм сортировки вставкой: для каждого элемента (начиная со второго, так как в алгоритме предполагается, что первый элемент уже отсортирован) ищется его правильное место среди предыдущих элементов массива; в процессе сравнения элементы “сдвигаются” пока не найдётся подходящая позиция для текущего элемента (key)
4. После того как элемент вставлен, в список indices добавляется индекс его новой позиции (в исходной нумерации, начиная с 1). Это значение вычисляется как j + 2, так как j отсчитывается с нуля, и +1 для учёта сдвига
5. Индексы вставки и отсортированный массив преобразуются в строковый формат
6. Открывается файл output.txt, и в него записываются индексы вставки на первой строке и отсортированный массив на второй строке

Результат работы кода на примере из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

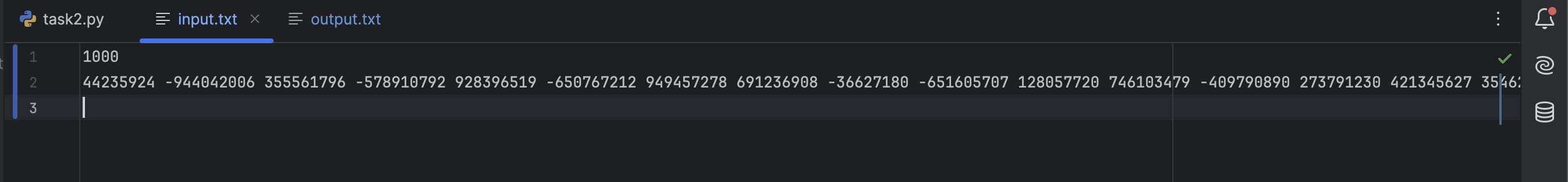
Результат работы кода на максимальных и минимальных значениях:

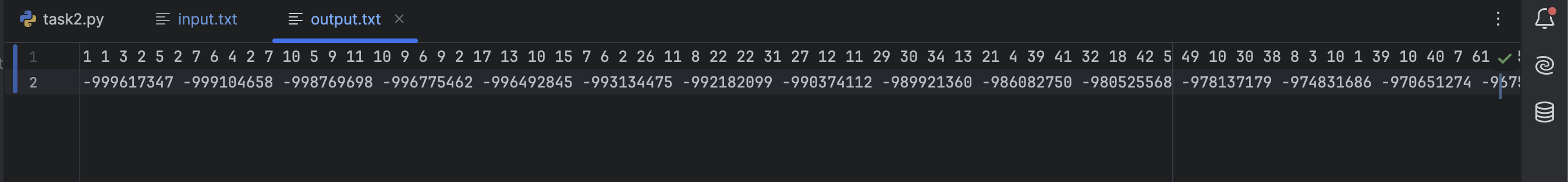
Изображение выглядит как снимок экрана, текст, Шрифт, Мультимедийное программное обеспечение

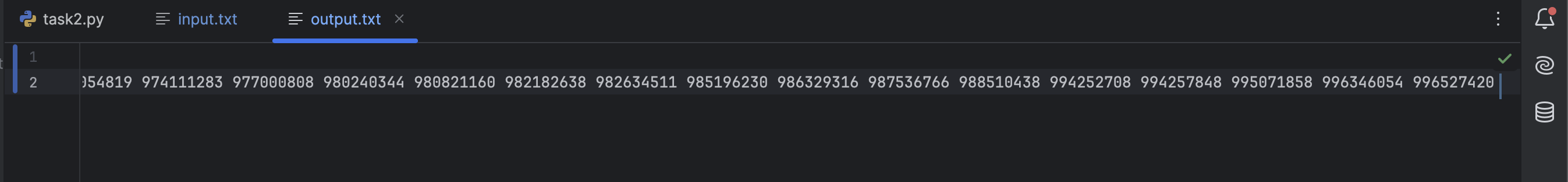
Автоматически созданное описание

Изображение выглядит как снимок экрана, текст, Шрифт, программное обеспечение

Автоматически созданное описание







|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00032733401167206466 секунд | 13863 байт |
| Пример из задачи | 0.00035366599331609905 секунд | 13880 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.33740270900307223 секунд | 124288 байт |

Вывод по задаче: мною был изучен алгоритм сортировки массива вставкой, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти. Был также получен способ записи новых индексов элементов при сортировке.

## Задача №8. Секретарь Своп

Текст задачи.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

Листинг кода.

import tracemalloc  
import time  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
if (3 <= n <= 5000) and (all(abs(i) <= 10\*\*9 for i in m)):  
 f\_output = open('output.txt', 'w')  
 for i in range(len(m)):  
 for j in range(i + 1, len(m)):  
 if m[j] < m[j - 1]:  
 m[j], m[j - 1] = m[j - 1], m[j]  
 f\_output.write(f'Swap elements at indices {j} and {j + 1}.\n')  
 if m == sorted(m):  
 break  
 f\_output.write('No more swaps needed.')  
else:  
 print('Введите корректные данные')  
print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Далее сам алгоритм сортировки с записью обменов: идём по индексам массива m; на каждой итерации внешнего цикла сортируются элементы от текущего индекса до конца массива; для каждого элемента m[j] проверяется, меньше ли он предыдущего элемента m[j-1]; если это так, элементы меняются местами, и в файл output.txt записывается информация о том, что произошла перестановка (индексы указываются в формате, начиная с 1, то есть j + 1)
4. Если на какой-то итерации внешнего цикла массив уже оказывается отсортированным (проверка m == sorted(m)), программа завершает сортировку досрочно, выходя из цикла; это позволяет ускорить выполнение, если дальнейшие проходы уже не нужны
5. Когда сортировка завершена, записывается строка “No more swaps needed”, сигнализирующая, что больше обменов не требуется

Результат работы кода на примере из текста задачи:

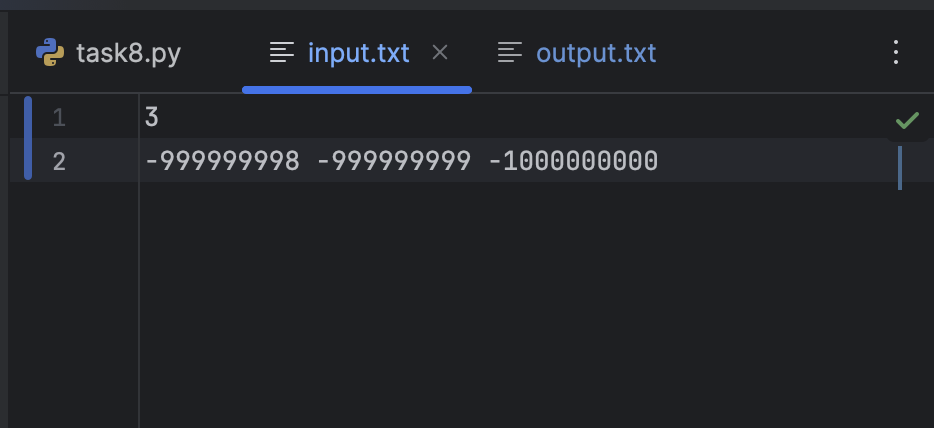
Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Результат работы кода на максимальных и минимальных значениях:



Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Изображение выглядит как снимок экрана, Мультимедийное программное обеспечение, программное обеспечение, компьютер

Автоматически созданное описание

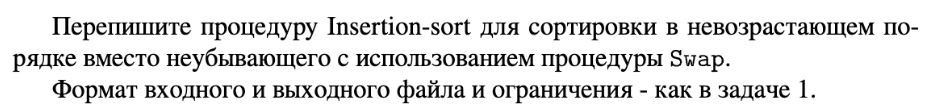
|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0007170420140028 секунд | 13907 байт |
| Пример из задачи | 0.00032629098859615624 секунд | 13859 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 57.775730666995514 секунд | 486492 байт |

Вывод по задаче: мною был изучен алгоритм пузырьковой сортировки, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти. Был также получен способ записи операции взаимной перестановки пары элементов.

# Дополнительные задачи

## Задача №3. Сортировка вставкой по убыванию

Текст задачи.



Листинг кода.

import tracemalloc  
import time  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
if (1 <= n <= 10\*\*3) and (all(abs(i) <= 10\*\*9 for i in m)):  
 for i in range(1, len(m)):  
 key = m[i]  
 j = i - 1  
 while j >= 0 and m[j] < key:  
 m[j + 1] = m[j]  
 j -= 1  
 m[j + 1] = key  
 m\_sorted = ' '.join(map(str, m))  
 f\_output = open('output.txt', 'w')  
 f\_output.write(m\_sorted)  
else:  
 print('Введите корректные данные')  
print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Для того, чтобы отсортировать массив в порядке убывания, пользуемся алгоритмом первой задачи, но меняем знак между m[j] и key с > на < в цикле while
4. После завершения сортировки массив преобразуется в строку с пробелами между элементами и записывается в output.txt

Результат работы кода на примере из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

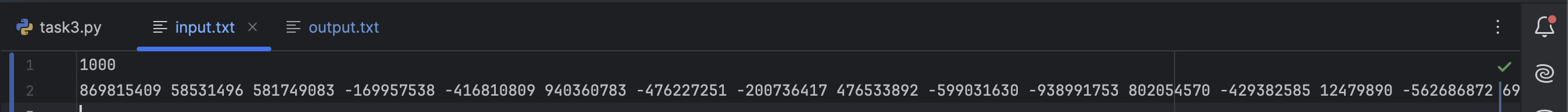
Результат работы кода на максимальных и минимальных значениях:

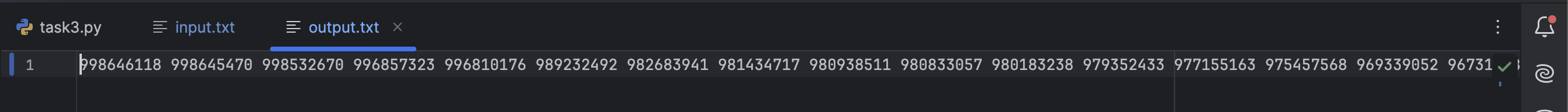
Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

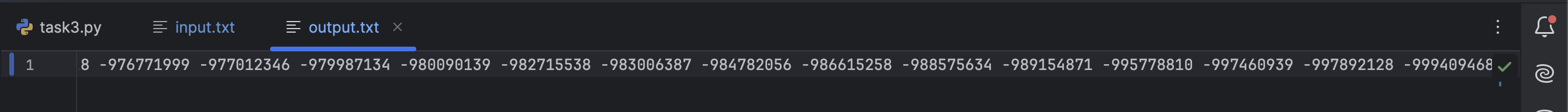
Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, Шрифт

Автоматически созданное описание







|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.00027833401691168547 секунд | 13863 байт |
| Пример из задачи | 0.000306583009660244 секунд | 13875 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.40635350000229664 секунд | 116148 байт |

Вывод по задаче: мною был изучен алгоритм сортировки массива вставкой по убыванию, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти.

## Задача №5. Сортировка выбором

Текст задачи.

Изображение выглядит как текст, Шрифт, снимок экрана, документ

Автоматически созданное описание

Листинг кода.

import tracemalloc  
import time  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
if (1 <= n <= 10\*\*3) and (all(abs(i) <= 10\*\*9 for i in m)):  
 for i in range(len(m)):  
 min\_elem = i  
 for j in range(i + 1, len(m)):  
 if m[j] < m[min\_elem]:  
 min\_elem = j  
 m[i], m[min\_elem] = m[min\_elem], m[i]  
 m\_sorted = ' '.join(map(str, m))  
 f\_output = open('output.txt', 'w')  
 f\_output.write(m\_sorted)  
else:  
 print('Введите корректные данные')  
print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Дальше идёт сам алгоритм сортировки выбором: внешний цикл идёт по каждому элементу массива, начиная с первого; каждый элемент на текущей позиции i будет заменён минимальным элементом из оставшейся несортированной части массива; переменная min\_elem инициализируется текущим индексом i и будет хранить индекс минимального элемента в оставшейся части массива; внутренний цикл проходит по всем элементам массива от i + 1 до конца; если элемент m[j] меньше текущего минимального элемента m[min\_elem], индекс минимального элемента обновляется (min\_elem = j); после завершения внутреннего цикла происходит обмен текущего элемента m[i] с найденным минимальным элементом m[min\_elem]
4. Отсортированный массив преобразуется в строку, где элементы разделены пробелом и записывается в output.txt

Результат работы кода на примере из текста задачи:

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Мультимедийное программное обеспечение, программное обеспечение

Автоматически созданное описание

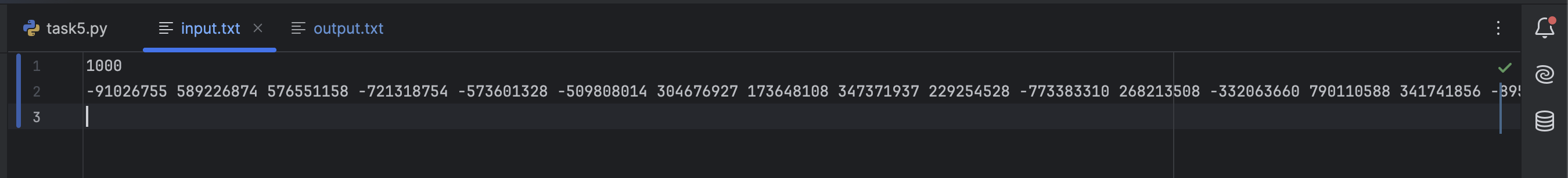
Результат работы кода на максимальных и минимальных значениях:

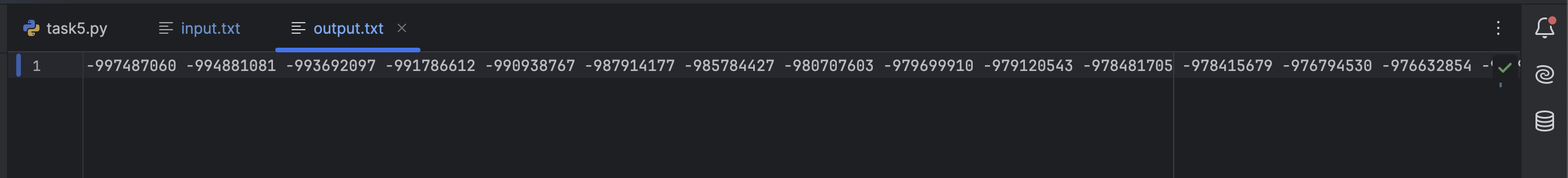
Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

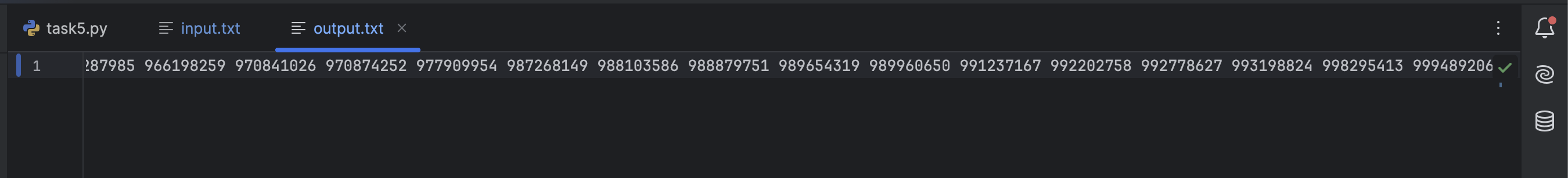
Автоматически созданное описание

Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

Автоматически созданное описание







|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0002698749885894358 секунд | 13863 байт |
| Пример из задачи | 0.00030341598903760314 секунд | 13875 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 0.4534935000119731 секунд | 116256 байт |

Вывод по задаче: мною был изучен алгоритм сортировки массива выбором, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти. В среднем случае время выполнения алгоритма 0.00030341598903760314 секунд. В наихудшем - 0.0003274159971624613 секунд. Сортировка вставками на практике обычно быстрее, особенно если массив частично отсортирован или небольшого размера. Сортировка выбором же медленнее, так как выполняет одинаковое количество сравнений независимо от структуры данных.

## Задача №6. Пузырьковая сортировка

Текст задачи.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Листинг кода.

import tracemalloc  
import time  
  
t\_start = time.perf\_counter()  
tracemalloc.start()  
f\_input = open('input.txt', 'r')  
n = int(f\_input.readline())  
m = [int(x) for x in f\_input.readline().split()]  
if (1 <= n <= 10 \*\* 3) and (all(abs(i) <= 10 \*\* 9 for i in m)):  
  
 #algorithm itself  
 for i in range(len(m)):  
 for j in range(len(m) - 1, i, -1):  
 if m[j] < m[j - 1]:  
 m[j], m[j - 1] = m[j - 1], m[j]  
 m\_sorted = ' '.join(map(str, m))  
 f\_output = open('output.txt', 'w')  
 f\_output.write(m\_sorted)  
 print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
 print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
 tracemalloc.stop()  
  
 #The proof that m'[1] <= m'[2] <= ... <= m'[n], where m' is the output of the bubble sort procedure  
 # and n is the length of the array m  
 condition = 0  
 for i in range(len(m) - 1):  
 if m[i] < m[i + 1]:  
 condition += 1  
 if condition == len(m) - 1:  
 print("m'[1] <= m'[2] <= ... <= m'[n]")  
else:  
 print('Введите корректные данные')  
 print("Время работы: %s секунд" % (time.perf\_counter() - t\_start))  
 print("Затрачено памяти:", tracemalloc.get\_traced\_memory()[1], "байт")  
 tracemalloc.stop()

Текстовое объяснение решения.

1. Считываем число элементов в массиве и сам массив из input.txt
2. Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
3. Дальше идёт сам алгоритм пузырьковой сортировки: внешний цикл проходится по каждому элементу массива от начала до конца; во внутреннем цикле, начиная с конца массива, сравниваются соседние элементы m[j] и m[j-1]; если элемент слева больше, они меняются местами; таким образом, в каждом проходе наибольший элемент “всплывает” к своему месту в конце массива; повторяется до тех пор, пока все элементы не окажутся на своих местах
4. Отсортированный массив преобразуется в строку и записывается в файл output.txt
5. Далее идёт проверка корректности сортировки: алгоритм проходит по отсортированному массиву и проверяет, что каждый элемент меньше или равен следующему; если это выполняется для всех элементов, выводится сообщение, подтверждающее правильность сортировки

Результат работы кода на примере из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

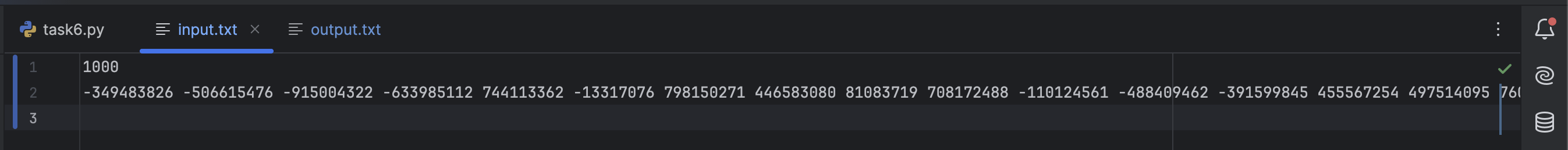
Результат работы кода на максимальных и минимальных значениях:

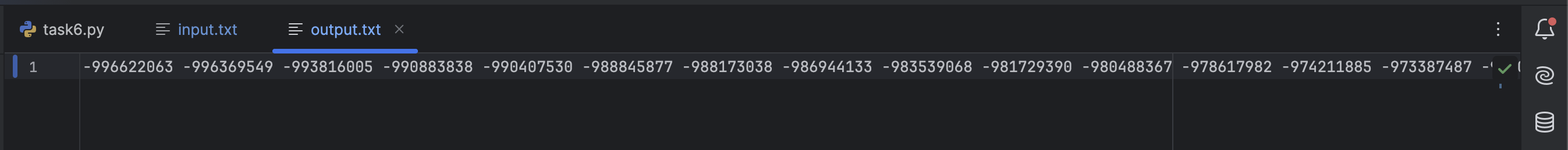
Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

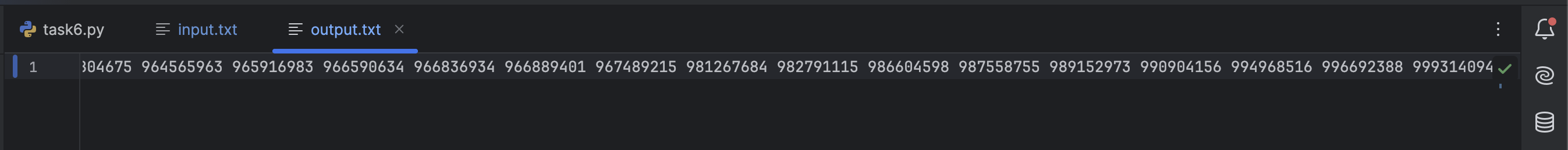
Автоматически созданное описание

Изображение выглядит как текст, Мультимедийное программное обеспечение, программное обеспечение, Шрифт

Автоматически созданное описание







|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0002955830132123083 секунд | 13863 байт |
| Пример из задачи | 0.00031250002211891115 секунд | 13875 байт |
| Верхняя граница диапазона значений входных данных из текста задачи | 1.1362923750129994 секунд | 116352 байт |

Вывод по задаче: мною был изучен алгоритм пузырьковой сортировки массива, а также выяснилось, что чем больше массив, тем дольше будет время выполнения и больше затраты памяти. В среднем случае время выполнения алгоритма 0.00031250002211891115 секунд. В наихудшем - 0.000582042004680261 секунд. Сортировка вставками на практике обычно быстрее, особенно если массив частично отсортирован или небольшого размера. Пузырьковая сортировка же медленнее, так как выполняет больше сравнений и обменов, что делает её менее эффективной на практике.

# Вывод

 В ходе лабораторной работы были изучены различные алгоритмы сортировки: вставкой, пузырьковая и выбором. Был проведён анализ работы алгоритмов на максимальных и минимальных значениях.