

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	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №2. Сортировка вставкой +	6
Задача №8. Секретарь Своп	9
Дополнительные задачи	13
Задача №3. Сортировка вставкой по убыванию	13
Задача №5. Сортировка выбором	16
Задача №6. Пузырьковая сортировка	19
Вывод	23

Задачи по варианту

Задача №1. Сортировка вставкой

Текст задачи.

Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива $A = \{31, 41, 59, 26, 41, 58\}$.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($1 \leq n \leq 10^3$) — 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.

Выберите любой набор данных, подходящих по формату, и протестируйте алгоритм.

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:

The first screenshot shows the initial state of the program. The 'input.txt' file contains two lines of data: '6' and '31 41 59 26 41 58'. The 'output.txt' file is empty. The second screenshot shows the result after the first iteration of the sorting algorithm. The 'output.txt' file now contains the sorted sequence '26 31 41 41 58 59'.

Результат работы кода на максимальных и минимальных значениях:

The first screenshot shows the program handling a dataset with extreme values: '1' and '-1000000000'. The second screenshot shows the result for the same dataset: '-1000000000'. The third screenshot shows the program handling a large dataset of 20 numbers, including '1000' and various negative values. The output shows the first few numbers of the sorted sequence: '-974837206 -926690429 -857208266 927202783 224012568 -214254855 440604456 -897252303 382031246 345309036 676568475 723934446 -864668372 60340964 802384602 -921'.

	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.

Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

- **Формат выходного файла (input.txt).** В первой строке выходного файла выведите n чисел. При этом i -ое число равно индексу, на который, в момент обработки его сортировкой вставками, был перемещен i -ый элемент исходного массива. Индексы нумеруются, начиная с единицы. Между любыми двумя числами должен стоять ровно один пробел.

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)

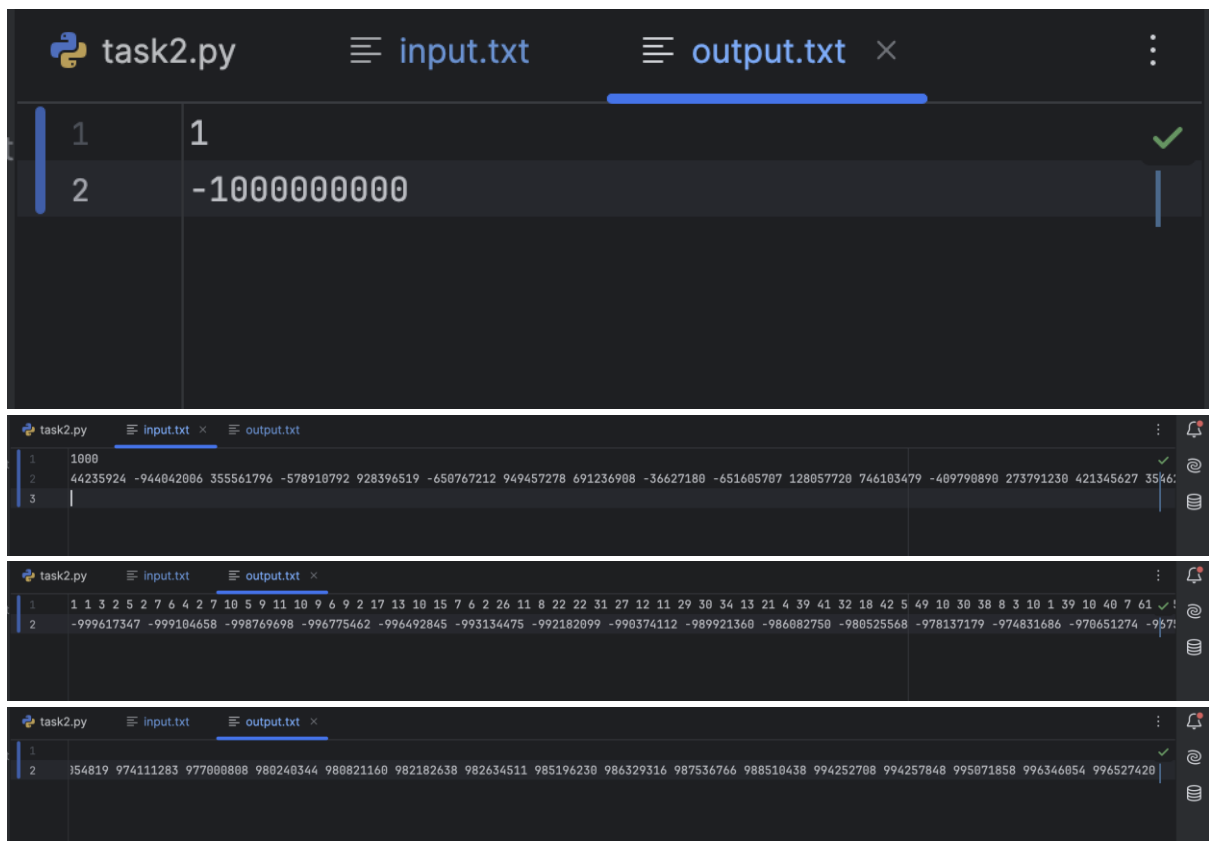
- 4) После того как элемент вставлен, в список `indices` добавляется индекс его новой позиции (в исходной нумерации, начиная с 1). Это значение вычисляется как $j + 2$, так как j отсчитывается с нуля, и $+1$ для учёта сдвига
- 5) Индексы вставки и отсортированный массив преобразуются в строковый формат
- 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:

The first screenshot shows the `input.txt` file with two lines of input: `10` and `1 8 4 2 3 7 5 6 9 0`. The second screenshot shows the `output.txt` file with two lines of output: `1 2 2 2 3 5 5 6 9 1` and `0 1 2 3 4 5 6 7 8 9`. Both screenshots show a green checkmark in the right margin, indicating successful execution.

Результат работы кода на максимальных и минимальных значениях:

The screenshot shows the `input.txt` file with two lines of input: `1` and `-10000000000`. The `output.txt` file is empty. A blue vertical bar is visible on the left side of the editor, and a green checkmark is in the right margin.



	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 Своя

Текст задачи.

Дан массив, состоящий из n целых чисел. Вам необходимо его отсортировать по неубыванию. Но делать это нужно так же, как это делает мистер Своя — то есть, каждое действие должно быть взаимной перестановкой пары элементов. Вам также придется записать все, что Вы делали, в файл, чтобы мистер Своя смог проверить Вашу работу.

- **Формат входного файла (input.txt).** В первой строке входного файла содержится число n ($3 \leq n \leq 5000$) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 10^9 . Числа могут совпадать друг с другом.
- **Формат выходного файла (output.txt).** В первых нескольких строках выведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:

Swap elements at indices X and Y .

Здесь X и Y — различные индексы массива, элементы на которых нужно переставить ($1 \leq X, Y \leq n$). Мистер Своя любит порядок, поэтому сделайте так, чтобы $X < Y$.

После того, как все нужные перестановки выведены, выведите следующую фразу:

No more swaps needed.

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[i]:
                m[j], m[i] = m[i], m[j]
                f_output.write(f'Swap elements at indices {i} 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 $m == \text{sorted}(m)$), 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:

The image shows two screenshots of a code editor interface. The top screenshot shows the 'input.txt' file with two lines of input: '5' on line 1 and '3 1 4 2 2' on line 2. A blue 'run' button is visible below the input. The bottom screenshot shows the 'output.txt' file with six lines of output: 'Swap elements at indices 1 and 2.', 'Swap elements at indices 3 and 4.', 'Swap elements at indices 4 and 5.', 'Swap elements at indices 2 and 3.', 'Swap elements at indices 3 and 4.', and 'No more swaps needed.' on lines 1 through 6 respectively. Both screenshots show a green checkmark in the right margin, indicating successful execution.

File	Line	Content
input.txt	1	5
	2	3 1 4 2 2
output.txt	1	Swap elements at indices 1 and 2.
	2	Swap elements at indices 3 and 4.
	3	Swap elements at indices 4 and 5.
	4	Swap elements at indices 2 and 3.
	5	Swap elements at indices 3 and 4.
	6	No more swaps needed.

Результат работы кода на максимальных и минимальных значениях:

The image displays three screenshots of a code editor interface, likely VS Code, showing the results of running a Python script named `task8.py` against test cases in `input.txt` and `output.txt`.

Top Screenshot: Shows the first two test cases. The first case has an input of `3` and a green checkmark. The second case has an input of `-9999999998 -999999999 -10000000000` and a blue vertical bar on the right.

Index	Input	Status
1	3	✓
2	-9999999998 -999999999 -10000000000	

Middle Screenshot: Shows the next three test cases. The first two have green checkmarks, and the third has a blue vertical bar on the right.

Index	Input	Status
1	Swap elements at indices 1 and 2.	✓
2	Swap elements at indices 2 and 3.	✓
3	No more swaps needed.	

Bottom Screenshot: Shows a larger set of test cases. The first two have green checkmarks, and the third has a blue vertical bar on the right.

Index	Input	Status
1	5000	✓
2	432818741 -806968886 642432311 -885017714 -69885879 -90576026 447536022 542517113 829452688 -714395791 842216948 822216540 -167853544 -61771588 817475514 1676	✓
3		

```
task8.py  input.txt  output.txt x
⚠ The file size (188,51 MB) exceeds the configured limit (2,56 MB). Code insight features are not available.

1  Swap elements at indices 1 and 2.
2  Swap elements at indices 3 and 4.
3  Swap elements at indices 4 and 5.
4  Swap elements at indices 5 and 6.
5  Swap elements at indices 6 and 7.
6  Swap elements at indices 7 and 8.
7  Swap elements at indices 9 and 10.
8  Swap elements at indices 11 and 12.
9  Swap elements at indices 12 and 13.
10 Swap elements at indices 13 and 14.
11 Swap elements at indices 14 and 15.
12 Swap elements at indices 15 and 16.
13 Swap elements at indices 16 and 17.
14 Swap elements at indices 17 and 18.
15 Swap elements at indices 19 and 20.
16 Swap elements at indices 20 and 21.
17 Swap elements at indices 21 and 22.
18 Swap elements at indices 22 and 23.
19 Swap elements at indices 23 and 24.
20 Swap elements at indices 24 and 25.
21 Swap elements at indices 25 and 26.
22 Swap elements at indices 26 and 27.
23 Swap elements at indices 28 and 29.
24 Swap elements at indices 29 and 30.
25 Swap elements at indices 30 and 31.
26 Swap elements at indices 31 and 32.
27 Swap elements at indices 32 and 33.
28 Swap elements at indices 34 and 35.
29 Swap elements at indices 35 and 36.
30 Swap elements at indices 36 and 37.
31 Swap elements at indices 37 and 38.
32 Swap elements at indices 38 and 39.
33 Swap elements at indices 39 and 40.
34 Swap elements at indices 40 and 41.
35 Swap elements at indices 41 and 42.
```

	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.

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swap.

Формат входного и выходного файла и ограничения - как в задаче 1.

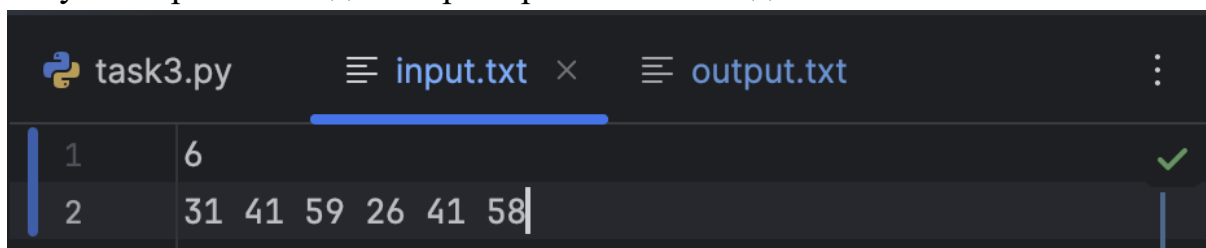
Листинг кода.

```
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:



task3.py	input.txt	output.txt
1	6	
2	31 41 59 26 41 58	

Результат работы кода на максимальных и минимальных значениях:

	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.

Рассмотрим сортировку элементов массива, которая выполняется следующим образом. Сначала определяется наименьший элемент массива, который ставится на место элемента $A[1]$. Затем производится поиск второго наименьшего элемента массива A , который ставится на место элемента $A[2]$. Этот процесс продолжается для первых $n - 1$ элементов массива A .

Напишите код этого алгоритма, также известного как сортировка выбором (selection sort). Определите время сортировки выбором в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

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) Далее 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 $m[\text{min_elem}]$, индекс минимального элемента обновляется ($\text{min_elem} = j$); после

завершения внутреннего цикла происходит обмен текущего элемента $m[i]$ с найденным минимальным элементом $m[\text{min_elem}]$

- 4) Отсортированный массив преобразуется в строку, где элементы разделены пробелом и записывается в output.txt

Результат работы кода на примере из текста задачи:

The first screenshot shows the input file 'input.txt' with two lines of data. The first line contains '1' and '6'. The second line contains '2' followed by the numbers '31 41 59 26 41 58'. A green checkmark is visible on the right side of the first line. The second screenshot shows the output file 'output.txt' with one line of data: '1 26 31 41 41 58 59'. A green checkmark is visible on the right side of the first line.

Результат работы кода на максимальных и минимальных значениях:

The first screenshot shows the input file 'input.txt' with two lines of data. The first line contains '1' and '1'. The second line contains '2' followed by '-1000000000'. A green checkmark is visible on the right side of the first line. The second screenshot shows the output file 'output.txt' with one line of data: '1 -1000000000'. A green checkmark is visible on the right side of the first line.

	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.

Пузырьковая сортировка представляет собой популярный, но не очень эффективный алгоритм сортировки. В его основе лежит многократная перестановка соседних элементов, нарушающих порядок сортировки. Вот псевдокод этой сортировки:

```
Bubble_Sort(A):  
  for i = 1 to A.length - 1  
    for j = A.length downto i+1  
      if A[j] < A[j-1]  
        поменять A[j] и A[j-1] местами
```

Напишите код на Python и докажите корректность пузырьковой сортировки. Для доказательства корректности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq \dots \leq A'[n]$, где A' - выход процедуры Bubble_Sort, а n - длина массива A .

Определите время пузырьковой сортировки в наихудшем случае и в среднем случае и сравните его со временем сортировки вставкой.

Формат входного и выходного файла и ограничения - как в задаче 1.

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.

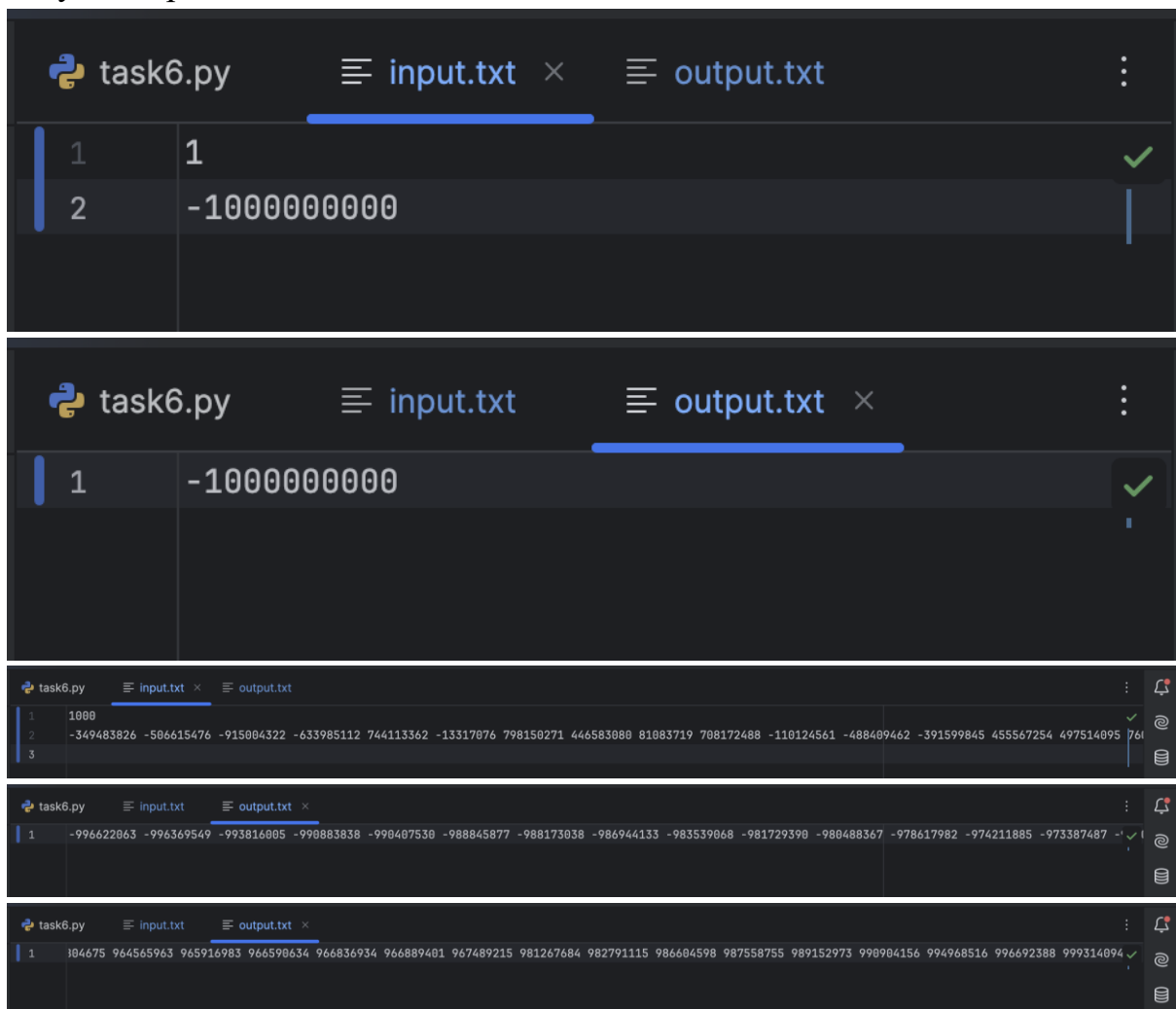
- 1) Считываем число элементов в массиве и сам массив из input.txt
- 2) Проверяем удовлетворяют ли полученные данные условию задачи. Если нет, то просим пользователя ввести корректные данные
- 3) Дальше идёт сам алгоритм пузырьковой сортировки: внешний цикл проходится по каждому элементу массива от начала 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:

task6.py	input.txt	output.txt
1	6	✓
2	59 58 41 41 31 26	

task6.py	input.txt	output.txt
1	26 31 41 41 58 59	✓

Результат работы кода на максимальных и минимальных значениях:



	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.