

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 1

Выполнила:

Гайдук А. С.

K3241

Проверила:

Ромакина О. М.

Санкт-Петербург

2024 г.

Содержание отчета

Содержание отчета.....	2
Задачи по варианту	3
Задача №1. Сортировка вставкой	3
Задача №2. Сортировка вставкой +.....	5
Задача №4. Линейный поиск	7
Дополнительные задачи	10
Задача №5. Сортировка выбором	10
Задача №3. Сортировка вставкой по убыванию	12
Задача №6. Пузырьковая сортировка.....	14
Задача №7. Знакомство с жителями Сортлэнда.....	17
Задача №8. Секретарь Своп	19
Вывод	22

Задачи по варианту

Задача №1. Сортировка вставкой

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

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def insertion_sort(array):
    for i in range(1, n):
        x = array[i]
        j = i

        while j > 0 and array[j - 1] > x:
            array[j] = array[j - 1]
            j -= 1

        array[j] = x

    return array

start_time = time.perf_counter()
with open('input_1.txt') as input_f:
    n = int(input_f.readline())
    a = [int(x) for x in input_f.readline().split()]

sorted_array = insertion_sort(a)
with open('output_1.txt', 'w') as output_f:
    output_f.write(" ".join(map(str, sorted_array)))

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()
print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения данной задачи я описала функцию сортировки вставками, которая перебирает элементы массива от второго до последнего и перемещает их через цикл while. Для того, чтобы отследить объем используемой памяти я использовала tracemalloc (запуск через tracemalloc.start()), чтобы рассчитать время выполнения – функцию time.perf_counter().

Результат работы кода на примерах из текста задачи:

текста задачи		
---------------	--	--

Вывод по задаче: в ходе решения данной задачи я научилась сортировать массивы с помощью сортировки вставкой.

Задача №2. Сортировка вставкой +

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.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def insertion_sort(array, n):
    indices = list(range(1, n + 1))
    for i in range(1, n):
        key = array[i]
        j = i - 1
        while j >= 0 and array[j] > key:
            array[j + 1] = array[j]
            j -= 1
        array[j + 1] = key
        indices[i] = j + 2

    return array, indices

start_time = time.perf_counter()
with open('input_2.txt') as input_f:
    n = int(input_f.readline())
    a = [int(x) for x in input_f.readline().split()]

sorted_array, indices = insertion_sort(a, n)

with open('output_2.txt', 'w') as output_f:
    output_f.write(" ".join(map(str, indices)) + "\n")
    output_f.write(" ".join(map(str, sorted_array)) + "\n")

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()
print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Решение аналогично задаче №1, за исключением создания массива indices, который представляет исходные индексы элементов массива. Когда элемент перемещается на новую позицию в массиве, его исходный индекс также помещается на эту позицию в indices.

Результат работы кода на примерах из текста задачи:

task2.py	input_2.txt	output_2.txt
1	10	
2	1 8 4 2 3 7 5 6 9 0	

task2.py	input_2.txt	output_2.txt
1	1 2 2 2 3 5 5 6 9 1	
2	0 1 2 3 4 5 6 7 8 9	

Результат работы кода на максимальных и минимальных значениях:

task2.py	input_2.txt	output_2.txt
1	1	
2	0	

task2.py	input_2.txt	output_2.txt
1	1	
2	0	

task2.py	input_2.txt	output_2.txt
1	100000	
2	-1000000000 -999999999 -999999998 -999999997 -999999996 -999999995 -999999994 -999999993 -999999992 -999999991	

task2.py	input_2.txt	output_2.txt
1	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65	

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.002177 сек	0.037685 MB
Пример из задачи	0.013629 сек	0.037764 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.616203 сек	14.628603 MB

Вывод по задаче:

В ходе решения данной задачи я преобразовала алгоритм вставками таким образом, чтобы при каждом шаге отслеживались новые индексы элементов исходного массива.

Задача №4. Линейный поиск

Рассмотрим задачу поиска.

- Формат входного файла. Последовательность из n чисел $A = a_1, a_2, \dots, a_n$ в первой строке, числа разделены пробелом, и значение V во второй строке. Ограничения: $0 \leq n \leq 10^3$, $-10^3 \leq a_i, V \leq 10^3$
- Формат выходного файла. Одно число - индекс i , такой, что $V = A[i]$, или значение -1 , если V отсутствует.
- Напишите код линейного поиска, при работе которого выполняется сканирование последовательности в поисках значения V .
- Если число встречается несколько раз, то выведите, сколько раз встречается число и все индексы i через запятую.

Листинг кода:

```
import tracemalloc
import time

tracemalloc.start()

def find_v(array, v):
    v_nums = []
    for i in range(0, len(array)):
        if array[i] == v:
            v_nums.append(i)
    if not v_nums:
```

```

        return -1
    return f'{len(v_nums)}\n{','.join(map(str, v_nums))}'

start_time = time.perf_counter()

with open ('input_3.txt', 'r') as input_f:
    array = [int(x) for x in input_f.readline().split()]
    v = int(input_f.readline())

with open('output_3.txt', 'w') as output_f:
    output_f.write(str(find_v(array, v)))

end_time = time.perf_counter()

current, peak = tracemalloc.get_traced_memory()
print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения этой задачи я создала метод `find_v`, который перебирает исходный массив в поисках `V`, и добавляет найденные элементы из исходного массива в массив `v_nums`.

Результат работы кода на максимальных и минимальных значениях:

The image shows four screenshots of a code editor with the following content:

- First screenshot:** Shows the input file `input_3.txt` with two lines: `-1000` and `1`. The output file `output_3.txt` is empty.
- Second screenshot:** Shows the input file `input_3.txt` with one line: `-1`. The output file `output_3.txt` is empty.
- Third screenshot:** Shows the input file `input_3.txt` with one line: `1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000 1000000000`. The output file `output_3.txt` is empty.
- Fourth screenshot:** Shows the input file `input_3.txt` with two lines: `100000` and `0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,`. The output file `output_3.txt` is empty.

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.001614 сек	0.037749 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.246668 сек	14.007273 МВ

Вывод по задаче:

В ходе выполнения данной задачи я научилась реализовывать алгоритм линейного поиска через цикл for.

Дополнительные задачи

Задача №5. Сортировка выбором

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 time
import tracemalloc

tracemalloc.start()

def selection_sort(array, n):
    for i in range(n-1):
        x = array[i]
        position = i
        for j in range(i + 1, n):
            if array[j] < x:
                x = array[j]
                position = j
        if position != i:
            array[i], array[position] = swap(array[i], array[position])
    return array

def swap(x,y):
    temp = x
    x = y
    y = temp
    return x,y

with open('input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(x) for x in f.readline().split()]

start_time = time.perf_counter()

sorted_array = selection_sort(a, n)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()
```

```

with open('output.txt', 'w') as f:
    f.write(" ".join(map(str, sorted_array)))

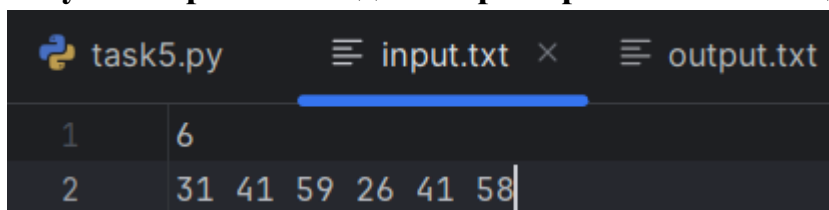
print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения этой задачи я написала метод, в котором при переборе всех элементов массива инициализируется текущий минимальный элемент. Далее минимальный элемент ищется в оставшейся части массива через вложенный цикл, и если он будет найден, то мы его меняем местами с изначальным минимальным элементом.

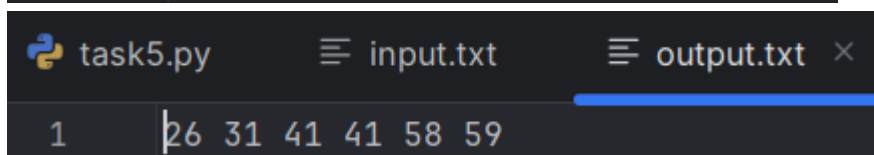
Результат работы кода на примерах из текста задачи:



```

task5.py  input.txt  output.txt
1 6
2 31 41 59 26 41 58

```

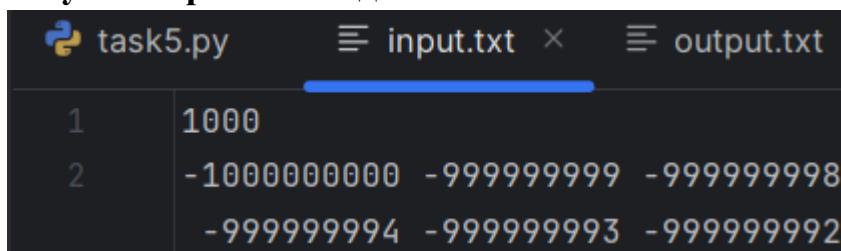


```

task5.py  input.txt  output.txt
1 26 31 41 41 58 59

```

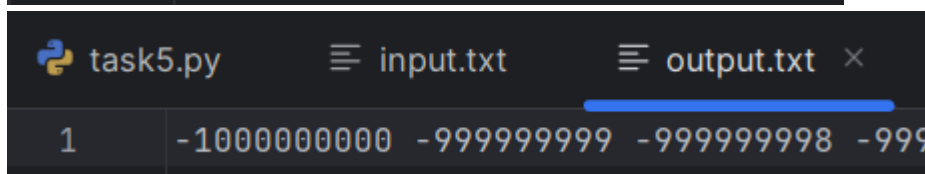
Результат работы кода на максимальных и минимальных значениях:



```

task5.py  input.txt  output.txt
1 1000
2 -1000000000 -999999999 -999999998
  -999999994 -999999993 -999999992

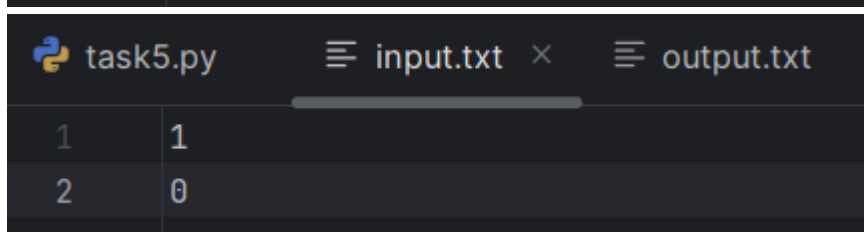
```



```

task5.py  input.txt  output.txt
1 -1000000000 -999999999 -999999998 -999999997

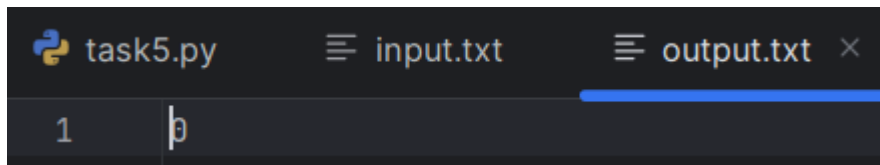
```



```

task5.py  input.txt  output.txt
1 1
2 0

```



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000003 сек	0.037661 МВ
Пример из задачи	0.000011 сек	0.037735 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.139730 сек	0.131445 МВ

Вывод по задаче:

В ходе выполнения данной задачи получилось реализовать алгоритм сортировки выбором минимального значения массива через вложенные циклы.

Задача №3. Сортировка вставкой по убыванию

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.

Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии?

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def insertion_sort_reverse(array):
    for i in range(1, n):
        x = array[i]
        j = i
        while j > 0 and array[j - 1] < x:
```

```

        array[j], array[j-1] = swap(array[j], array[j-1])
        j -= 1
    array[j] = x
    return array

def swap(x, y):
    temp = x
    x = y
    y = temp
    return x, y

with open('input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(x) for x in f.readline().split()]

start_time = time.perf_counter()

sorted_array = insertion_sort_reverse(array=a)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(" ".join(map(str, sorted_array)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

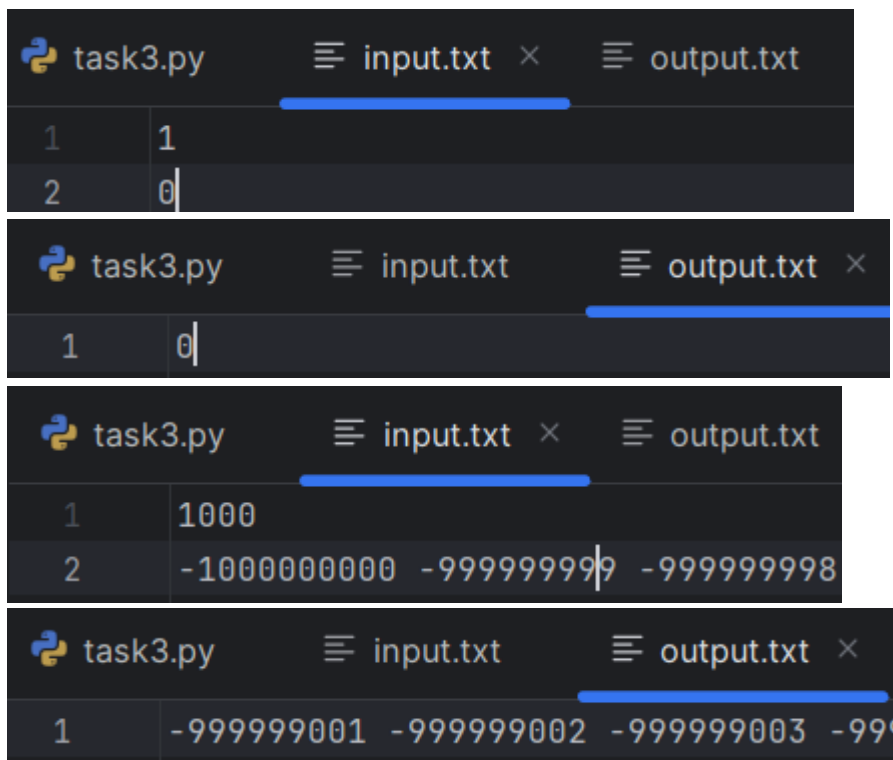
Для решения этой задачи я преобразовала код, написанный для задачи 1, добавив функцию Swap, которая меняет местами значения переменных.

Результат работы кода на примерах из текста задачи:

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

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

Результат работы кода на максимальных и минимальных значениях:



	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.002177 сек	0.037685 MB
Пример из задачи	0.000013 сек	0.037895 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.457798 сек	0.131605 MB

Вывод по задаче:

В ходе выполнения этой задачи я ознакомилась с алгоритмом сортировки вставкой по убыванию при помощи процедуры Swap.

Задача №6. Пузырьковая сортировка

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

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 time
import tracemalloc

tracemalloc.start()

def bubble_sort(array, n):
    for i in range(n-1):
        for j in range(n-1-i):
            if array[j] > array[j+1]:
                temp = array[j+1]
                array[j+1] = array[j]
                array[j] = temp
    return array

with open('input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(x) for x in f.readline().split()]

start_time = time.perf_counter()

sorted_array = bubble_sort(array=a, n=n)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(" ".join(map(str, sorted_array)))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МБ; Пиковое  
использование: {peak / 10 ** 6:.6f} МБ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения этой задачи я написала метод, который содержит вложенный цикл. После каждой итерации внешнего цикла, на свое место ставится следующий наибольший элемент.

Результат работы кода на примерах из текста задачи:

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

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

Результат работы кода на максимальных и минимальных значениях:

```
task6.py  task6\input.txt  task6\output.txt
1 1000
2 -10000000000 -999999999 -999999998 -999999997
3
```

```
task6.py  task6\input.txt  task6\output.txt
1 -10000000000 -999999999 -999999998 -999999997 -
```

```
task6.py  input.txt  output.txt
1 1
2 0
```

```
task6.py  input.txt  output.txt
1 0
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000003 сек	0.037661 МВ
Пример из задачи	0.000010 сек	0.037735 МВ
Верхняя граница диапазона значений входных данных из текста задачи	0.173580 сек	0.131445 МВ

Вывод по задаче:

В ходе выполнения этой задачи я ознакомилась с алгоритмом Bubble Sort и его реализацией.

Задача №7. Знакомство с жителями Сортлэнда

Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет n , где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем.

Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n . Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i , содержится в ячейке $M[i]$. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def bubble_sort(array, n):
    for i in range(0, n-1):
        for j in range(0, n-1-i):
            if array[j] > array[j+1]:
                temp = array[j+1]
                array[j+1] = array[j]
                array[j] = temp
    return array

with open('input.txt', 'r') as f:
    n = int(f.readline())
    a = [float(x) for x in f.readline().split()]

start_time = time.perf_counter()
```

```
sorted_array = bubble_sort(array=a.copy(), n=n)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(f"{a.index(sorted_array[0]) + 1} {a.index(sorted_array[n // 2]) + 1} {a.index(sorted_array[-1]) + 1}")

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")
```

Текстовое объяснение решения:

Для решения этой задачи я преобразовала код, написанный для задачи №6, при этом учла факт работы с вещественными числами и формат выходного файла.

Результат работы кода на примерах из текста задачи:

task7.py	input.txt	output.txt
1	5	
2	10.00 8.70 0.01 5.00 3.00	

task7.py	input.txt	output.txt
1	3 4 1	

Результат работы кода на максимальных и минимальных значениях:

task7.py	input.txt	output.txt
1	3	
2	0.00 0.01 0.02	

task7.py	input.txt	output.txt
1	1 2 3	

task7.py	input.txt	output.txt
1	9999	
2	1000000.0 999999.99 999999.98 999	

```
task7.py  input.txt  output.txt x
1 9999 5000 1
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000011 сек	0.037729 МВ
Пример из задачи	0.000026 сек	0.037751 МВ
Верхняя граница диапазона значений входных данных из текста задачи	1.469023 сек	12.421745 МВ

Вывод по задаче:

В ходе выполнения этой задачи я ознакомилась с реализацией алгоритма Bubble Sort при работе с вещественными числами, а также вывела начальные индексы элементов до сортировки.

Задача №8. Секретарь Своп

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

Листинг кода:

```
import time
import tracemalloc

tracemalloc.start()

def selection_sort(array, n):
    result = ''
    for i in range(0, n-1):
        x = array[i]
        position = i
        for j in range(i + 1, n):
            if array[j] < x:
```

```

        x = array[j]
        position = j
        if position != i:
            array[i], array[position] = swap(array[i], array[position])
            result += f"Swap elements at indices {i + 1} and {position +
1}.\n"
        result += "No more swaps needed."
        return result

def swap(x,y):
    temp = x
    x = y
    y = temp
    return x,y

with open('input.txt', 'r') as f:
    n = int(f.readline())
    a = [int(x) for x in f.readline().split()]

start_time = time.perf_counter()

result = selection_sort(a, n)

end_time = time.perf_counter()
current, peak = tracemalloc.get_traced_memory()

with open('output.txt', 'w') as f:
    f.write(str(result))

print(f"Затраты памяти: {current / 10 ** 6:.6f} МВ; Пиковое
использование: {peak / 10 ** 6:.6f} МВ")
print(f"Время выполнения программы: {end_time - start_time:.6f} секунд")

```

Текстовое объяснение решения:

Для решения данной задачи я преобразовала код, написанный для задачи №5 и использовала процедуру Swap и вложенный цикл. На каждой итерации выбирается минимальный элемент из оставшейся части массива и перемещается в начало.

Результат работы кода на примерах из текста задачи:

task8.py		input.txt	×	output.txt
1	5			
2	3 1 4 2 2			

```
task8.py  input.txt  output.txt x
1  Swap elements at indices 1 and 2.
2  Swap elements at indices 2 and 4.
3  Swap elements at indices 3 and 5.
4  No more swaps needed.
```

Результат работы кода на максимальных и минимальных значениях:

```
task8.py  input.txt x  output.txt
1  3
2  0 2 1
```

```
task8.py  input.txt  output.txt x
1  Swap elements at indices 2 and 3.
2  No more swaps needed.
```

```
task8.py  input.txt x  output.txt
1  5000
2  1000000000 1000000000 1000000000
```

```
task8.py  input.txt  output.txt x
1  No more swaps needed.
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений входных данных из текста задачи	0.000019 сек	0.037871 MB
Пример из задачи	0.000042 сек	0.037879 MB
Верхняя граница диапазона значений входных данных из текста задачи	0.630196 сек	0.539790 MB

Вывод по задаче:

В ходе выполнения этой задачи я научилась выводить информацию о перемещениях элементов в процессе сортировки.

Вывод

В процессе выполнения данной лабораторной работы я ознакомилась с алгоритмами сортировки вставками, выбором и Bubble Sort.