САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ ИНФОРМАТИКИ

Отчет по лабораторной работе №1
по курсу «Алгоритмы и структуры данных»
Тема: Insertion Sort
Вариант 1

Выполнила:
Анисимова. В. А.
К3162

Проверил:
Царьков Г. И.

Санкт-Петербург
2025 г.

Содержание отчета

Задачи по варианту

Задача №1. Сортировка вставкой

Задача №2. Сортировка вставкой +

Задача №3. Сортировка вставкой по убыванию

Вывод.

1 задача. Сортировка вставкой

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 ≤ n ≤ 103) — 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.

```
def insertion_sort(arr):
 for i in range(1, len(arr)):
 current = arr[i]
 j = i - 1
 while j >= 0 and arr[j] > current:
 arr[j + 1] = arr[j]
 j -= 1
 arr[j + 1] = current
 return arr
 with open('input.txt', 'r') as file:
 n = int(file.readline().strip())
 numbers = list(map(int, file.readline().strip().split()))
 sorted_numbers = insertion_sort(numbers)
 with open('output.txt', 'w') as file:
 file.write(' '.join(map(str, sorted_numbers)))
 print("Готово! Результат в output.txt")
```

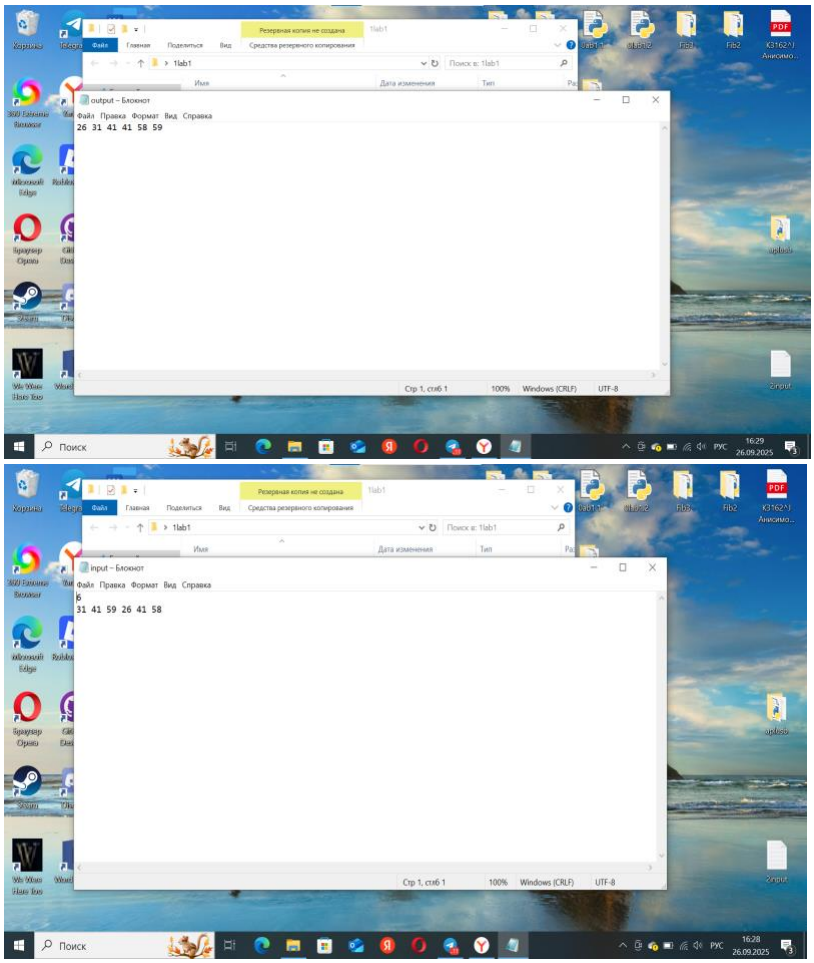
Объяснение:

Нужно отсортировать массив по порядку возрастания с помощью сортировки вставками

Для этого нам нужно создать входной и выходной файл

Во входном файле будут находится числа массива, где первая строка состоит из общего числа количества чисел массива, а вторая строка из чисел, вписанных через пробел

Данный алгоритм делит массив на 2 части, отсортированная (слева) и не отсортированная (справа) части.



	Время выполнения	Затраты памяти
Нижний диапазон: n=1	0.000007 сек	0.000027 MB
Верхний диапазон: n=1000	0.629 сек	0.026703 MB
Пример из задачи: n=6	0.000023 сек	0.000160 MB

Вывод:

Сортировка вставкой - это эффективный алгоритм для решения малых массивов таких как n<=1000. И данная сортировка успешно справляется с задачей. Алгоритм корректно отсортировал входные данные массива.

2 задача. 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.

• 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.

Пример.

input.txt output.txt

10 1 2 2 2 3 5 5 6 9 1

1 8 4 2 3 7 5 6 9 0 0 1 2 3 4 5 6 7 8 9

В примере сортировка вставками работает следующим образом:

• Первый элемент остается на своем месте, поэтому первое число в ответе — единица. Отсортированная часть массива: [1]

• Второй элемент больше первого, поэтому он тоже остается на своем месте, и второе число в ответе — двойка. [1 8]

• Четверка меньше восьмерки, поэтому занимает второе место. [1 4 8]

• Двойка занимает второе место. [1 2 4 8]

• Тройка занимает третье место. [1 2 3 4 8]

• Семерка занимает пятое место. [1 2 3 4 7 8]

• Пятерка занимает пятое место. [1 2 3 4 5 7 8]

• Шестерка занимает шестое место. [1 2 3 4 5 6 7 8]

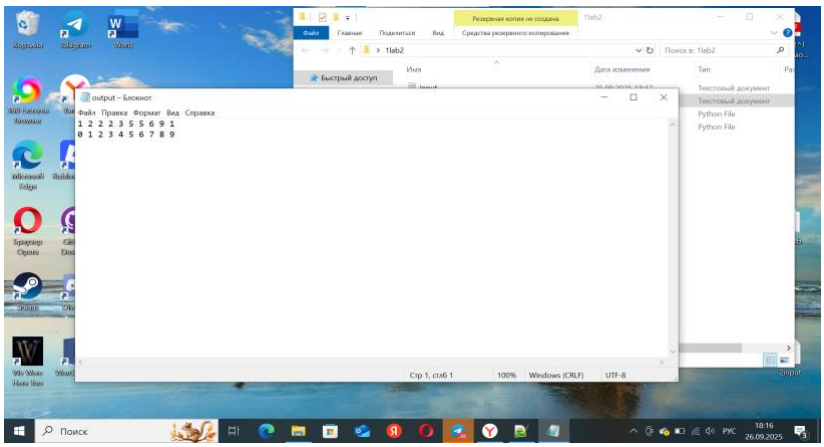
• Девятка занимает девятое место. [1 2 3 4 5 6 7 8 9]

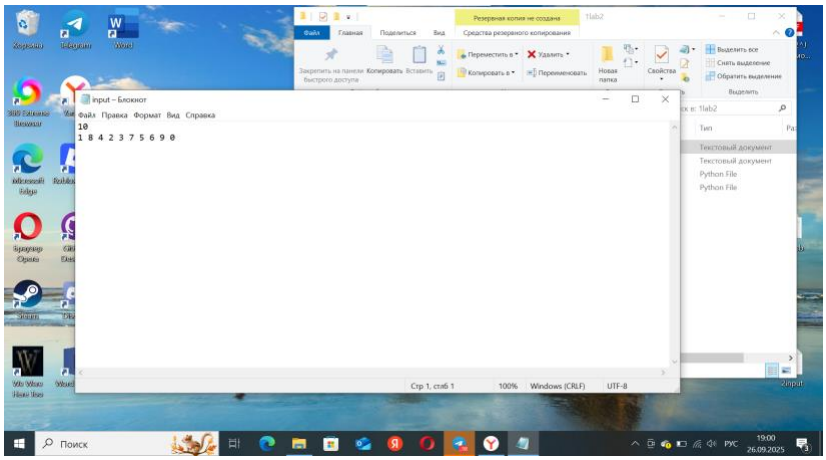
• Ноль занимает первое место. [0 1 2 3 4 5 6 7 8 9]

```
with open('input.txt', 'r') as f:
 n = int(f.readline())
 arr = list(map(int, f.readline().split()))
 sorted_arr = arr.copy()
 positions = [0] * n
 positions[0] = 1
 for i in range(1, n):
 key = sorted_arr[i]
 j = i - 1
 while j >= 0 and sorted_arr[j] > key:
 sorted_arr[j + 1] = sorted_arr[j]
 j = j - 1
 sorted_arr[j + 1] = key
 positions[i] = j + 2
 with open('output.txt', 'w') as f:
 f.write(' '.join(map(str, positions)) + '\n')
 f.write(' '.join(map(str, sorted_arr)))
 print(" ")
```

Объяснение:

В данной задачи нужно взять элемент из неотсортированной части, найти ему правильное место в отсортированной части и сдвинуть другие элементы, чтобы вставить элемент на найденную позицию. Задача стоит в том, что программе нужно запомнить для каждого элемента на какое место по счету его поставили в момент его обработки. И вывести результат в выходном файле.





	Время выполнения	Затраты памяти
Нижний диапазон: n=1	0.000005 сек	0.000027 MB
Верхний диапазон: n=1000	0.264 сек	0.026703 MB
Пример: n=10	0.000025 сек	0.000267 MB

Вывод:

Задача научила меня отслеживать промежуточные результаты работы алгоритма, а не только конечный результат отсортированного массива.

3 задача. Сортировка вставкой по убыванию

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

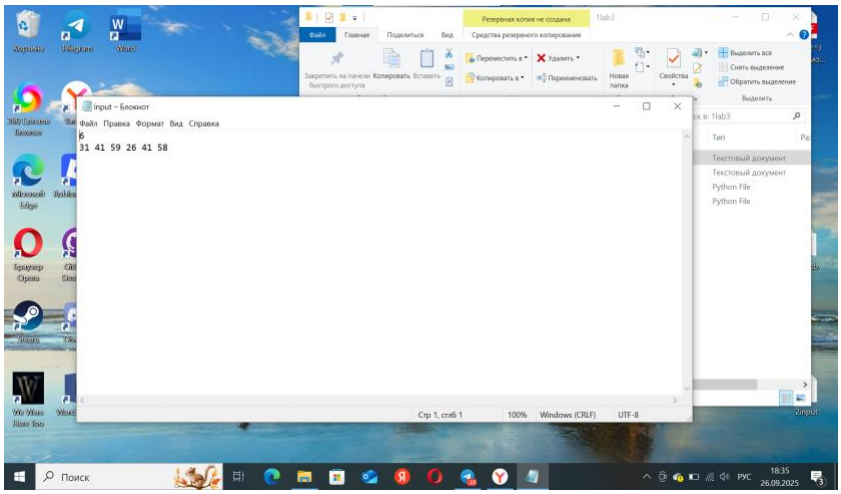
Формат входного и выходного файла и ограничения - как в задаче 1.

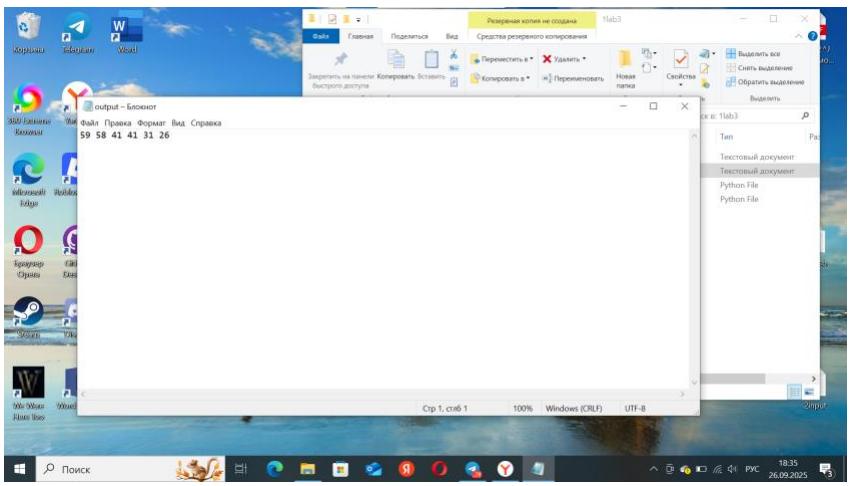
Подумайте, можно ли переписать алгоритм сортировки вставкой с исполь-зованием рекурсии?

```
with open('input.txt', 'r') as f:
 n = int(f.readline())
 arr = list(map(int, f.readline().split()))
 for i in range(1, len(arr)):
 key = arr[i]
 j = i - 1
 while j >= 0 and arr[j] < key:
 arr[j + 1] = arr[j]
 j -= 1 arr[j + 1] = key
 with open('output.txt', 'w') as f:
  f.write(' '.join(map(str, arr)))
 print(" ")
```

Объяснение:

В данной задаче нужно изменить алгоритм сортировки вставкой так, чтобы массив сортировался от большего к меньшему, то есть по убыванию. То есть нужно найти место, где текущий элемент меньше предыдущего.





	Время выполнения	Затраты памяти
Нижний диапазон: n=1	0.000007 сек	0.000027 MB
Верхний диапазон: n=1000	0.437 сек	0.026703 MB
Пример: n=6	0.000016 сек	0.000160 MB

Вывод:

Задача показывает, то как алгоритм сортировки вставкой можно адаптировать для разных видов и порядков сортировки. По поводу алгоритма с использованием рекурсии, я считаю, что это не очень эффективный метод. Так как рекурсия создает большую нагрузку на стек и его вызов. Усложняет понимание кода, работает медленнее циклов. Поэтому я использовала итеративную версию алгоритмов.

Вывод по всей лабораторной работе:

По моему мнению сортировка вставкой- это простой алгоритм для небольших массивов, чтобы быстро и с расчетом последовательности вставить элементы в отсортированную часть массива.

Итеративный алгоритм более экономичен по памяти и времени чем рекурсивный.

Благодаря данным задачам я освоила алгоритм сортировки.