

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 3

Выполнила:
Блинова П. В.
К3139 (номер группы)

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 №6. Пузырьковая сортировка	10
Дополнительные задачи	14
Задача №3. Сортировка вставкой по убыванию	14
Задача №5. Сортировка выбором	17
Задача №10. Палиндром	20
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.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_1/tests/output.txt", 'w')

def insertion_sort(n, list_arr):
    for i in range(1, n):
        key = list_arr[i]
        j = i - 1
        while j >= 0 and key < list_arr[j]:
            list_arr[j + 1] = list_arr[j]
            j -= 1
        list_arr[j + 1] = key
    return list_arr

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_1/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 10 ** 3:
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            file_output.write(' '.join(map(str, insertion_sort(n, nums))))
        else:
            print('Введите другое число')
    else:
```

```
print('Неверный ввод данных')  
file_output.close()
```

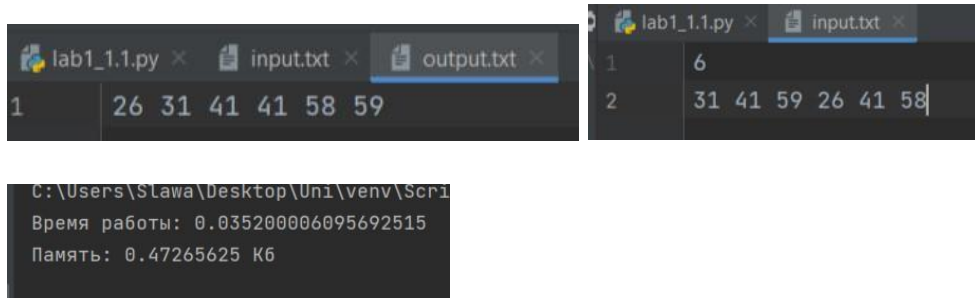
Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file_output результат функции insertion_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция insertion_sort:

Параметрами функции являются n, число элементов массива, и list_arr, сам массив. С помощью цикла for прохожусь по элементам массива, начиная с 1. Записываю элемент массива в переменную key. В переменную j записываю индекс предыдущего элемента. С помощью цикла while, пока j больше или равно 0 и элемент массива меньше элемента массива под индексом j, определяю позицию элемента в отсортированном списке. Эта функция работает поэтапно, разбивая список на отсортированную и неотсортированную части. Путём последовательно сравнения с элементами отсортированного списка первый элемент из неотсортированного вставит на своё место.

Результат работы кода на примерах из текста задачи:



```
lab1_1.1.py x input.txt x output.txt x  
1 26 31 41 41 58 59  
6 31 41 59 26 41 58  
C:\Users\Slawa\Desktop\Un1\venv\Scr1  
Время работы: 0.035200006095692515  
Память: 0.47265625 K6
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab1.task_1.src.lab1_1 import insertion_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
Время работы: 0.0153000000086426735
Память: 0.38671875 Кб
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab1.task_1.src.lab1_1 import insertion_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 3)]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Script
Время работы: 0.2644021000014618
Память: 38.66796875 Кб
```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений	0.0153	0.386

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.00352	0.4726
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	0.264402	38.6679

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки вставкой. Была изучена эффективность этого метода.

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.

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]

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_2/tests/output.txt", 'w')

def insertion_sort(n, list_arr):
    index_result = [1]
    for i in range(1, n):
        for j in range(i - 1, -1, -1):
            if list_arr[i] < list_arr[j]:
                list_arr[i], list_arr[j] = list_arr[j], list_arr[i]
                i, j = j, i
        index_result.append(i + 1)
    return index_result, list_arr

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_2/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 10 ** 3:
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            index_list, res = insertion_sort(n, nums)
            file_output.write(f"{' '.join(map(str, index_list))}\n")
            file_output.write(' '.join(map(str, res)))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()
```

Текстовое объяснение решения.

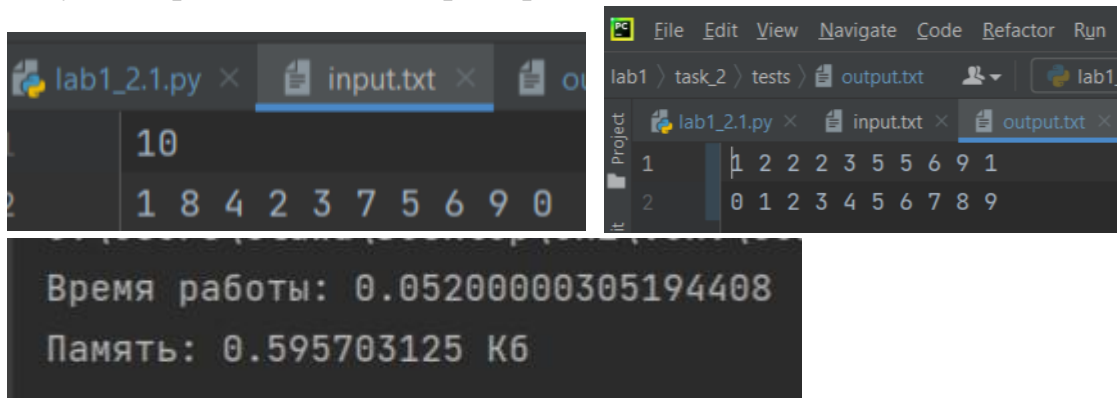
Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file_output

результат функции `insertion_sort`, делая с помощью `map()` все элементы строками, и объединяю результат в строку через `join`. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция `insertion_sort`:

Параметрами функции являются `n`, число элементов массива, и `list_arr`, сам массив. С помощью цикла `for` прохожусь по элементам массива, начиная с 1. Записываю элемент массива в переменную `key`. В переменную `j` записываю индекс предыдущего элемента. С помощью цикла `while`, пока `j` больше или равно 0 и элемент массива меньше элемента массива под индексом `j`, определяю позицию элемента в отсортированном списке. Эта функция работает поэтапно, разбивая список на отсортированную и неотсортированную части. Путём последовательно сравнения с элементами отсортированного списка первый элемент из неотсортированного встаёт на своё место.

Результат работы кода на примерах из текста задачи.



Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab1.task_2.src.lab1_2 import insertion_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
```



```
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Script
Время работы: 0.026000008801929653
Память: 0.39453125 Кб
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab1.task_2.src.lab1_2 import insertion_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 3)]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
Время работы: 0.22343549999641255
Память: 57.982421875 Кб
```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.026	0.3945
Пример из задачи	0.052	0.5957
Верхняя граница диапазона значений входных данных из текста задачи	0.2234	57.9824

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки вставкой. Была изучена эффективность этого метода. Был получен способ записи новых индексов элементов при обработке.

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.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_6/tests/output.txt", 'w')

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

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_6/tests/input.txt",
'r') as f:
```

```

file = f.readlines()
n = int(file[0])
if 1 <= n <= 10 ** 3:
    nums = list(map(int, list(file[1].split(' '))))
    if all([abs(n) <= 10 ** 9 for x in nums]):
        file_output.write(' '.join(map(str, bubble_sort(n, nums))))
    else:
        print('Введите другое число')
else:
    print('Неверный ввод данных')

file_output.close()

```

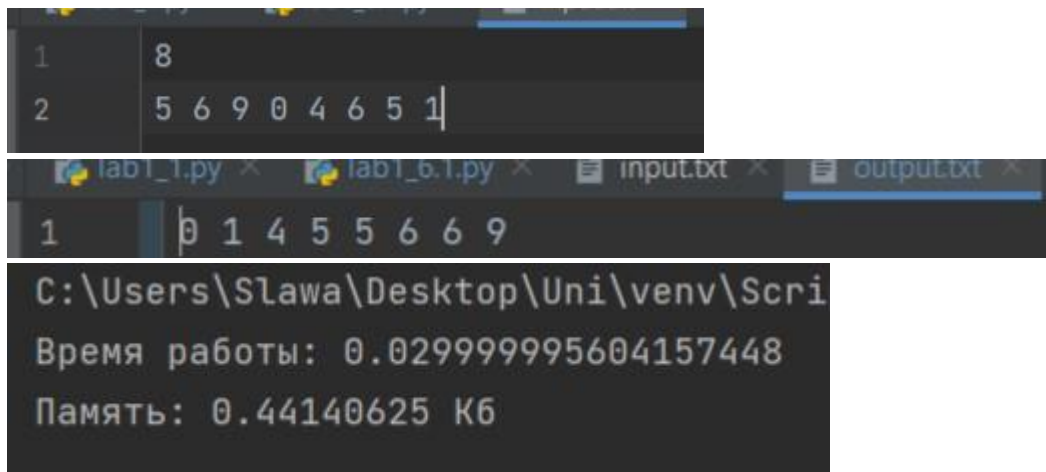
Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file_output результат функции bubble_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция bubble_sort:

Параметрами функции являются n, число элементов массива, и A, сам массив. С помощью двух вложенных циклов for прохожусь по массиву, сравнивая элементы под индексами из цикла. Если порядок в паре неверный, выполняю перестановку элементов. Алгоритм работает так, что на каждом шаге он находит наибольший элемент в текущей части массива и помещает его в конец этой части рядом с предыдущим наибольшим элементом. При этом наименьший элемент текущей части массива сдвигается на одну позицию ближе к началу. Процесс повторяется для оставшейся части массива, пока не будут обработаны все элементы.

Результат работы кода на примерах из текста задачи:



Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab1.task_6.src.lab1_6 import bubble_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = bubble_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' K6')
tracemalloc.stop()
```

Время работы: 0.019500002963468432
Память: 0.38671875 K6

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab1.task_6.src.lab1_6 import bubble_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 3)]

result = bubble_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' K6')
tracemalloc.stop()
```

C:\Users\Stawa\Desktop\011\veniv\30
Время работы: 0.8634287999884691
Память: 38.546875 Кб

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.0195	0.3867
Пример из задачи	0.02999	0.4414
Верхняя граница диапазона значений входных данных из текста задачи	0.8634	38.5468

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки пузырьком. Была изучена эффективность этого метода.

Дополнительные задачи

Задача №3. Сортировка вставкой по убыванию

Текст задачи.

Перепишите процедуру Insertion-sort для сортировки в невозрастающем порядке вместо неубывающего с использованием процедуры Swar. Формат входного и выходного файла и ограничения - как в задаче 1. Подумайте, можно ли переписать алгоритм сортировки вставкой с использованием рекурсии.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_3/tests/output.txt", 'w')

def insertion_sort(n, list_arr):
    for i in range(1, n):
        key = list_arr[i]
        j = i - 1
        while j >= 0 and key > list_arr[j]:
            list_arr[j + 1] = list_arr[j]
            j -= 1
        list_arr[j + 1] = key
    return list_arr

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_3/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 10 ** 3:
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
            file_output.write(' '.join(map(str, insertion_sort(n, nums))))
        else:
            print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()
```

Текстовое объяснение решения.

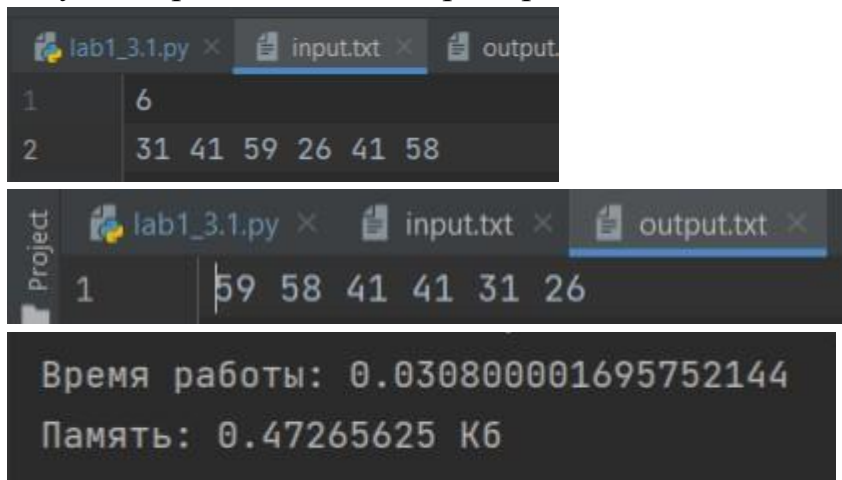
Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод

данных корректен, то с помощью `split()` делю вторую строку в файле по пробелам, делая массив. С помощью `map()` делаю элементы этого массива числами. Записываю результат в переменную `nums`. С помощью `all()` убеждаюсь, что все элементы удовлетворяют условию, записываю в `file_output` результат функции `insertion_sort`, делая с помощью `map()` все элементы строками, и объединяю результат в строку через `join`. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция `insertion_sort`:

Параметрами функции являются `n`, число элементов массива, и `list_arr`, сам массив. С помощью цикла `for` прохожусь по элементам массива, начиная с 1. Записываю элемент массива в переменную `key`. В переменную `j` записываю индекс предыдущего элемента. С помощью цикла `while`, пока `j` больше или равно 0 и элемент массива больше элемента массива под индексом `j`, определяю позицию элемента в отсортированном списке. Эта функция работает поэтапно, разбивая список на отсортированную и неотсортированную части. Путём последовательного сравнения с элементами отсортированного списка первый элемент из неотсортированного встаёт на своё место.

Результат работы кода на примерах из текста задачи:



The screenshot shows a code editor with three tabs: `lab1_3.1.py`, `input.txt`, and `output.txt`. The `input.txt` tab is active, showing the following content:

```
1 6
2 31 41 59 26 41 58
```

Below the editor, the `output.txt` tab is active, showing the following content:

```
1 59 58 41 41 31 26
```

At the bottom, a terminal window displays the execution time and memory usage:

```
Время работы: 0.030800001695752144
Память: 0.47265625 K6
```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab1.task_3.src.lab1_3 import insertion_sort
```

```

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()

```

```

Время работы: 0.01649999467190355
Память: 0.384765625 Кб

```

Максимальные значения

```

import time
import tracemalloc
import random
from asd_itmo.lab1.task_3.src.lab1_3 import insertion_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 3)]

result = insertion_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()

```

```

C:\Users\Slawa\Desktop\Uni\venv\Script
Время работы: 0.35087370000837836
Память: 38.654296875 Кб

```

	Время выполнения, с	Затраты памяти, Кб
Нижняя граница диапазона значений входных данных из текста задачи	0.0164	0.3847
Пример из задачи	0.0308	0.4726
Верхняя граница	0.3508	38.6542

диапазона значений входных данных из текста задачи		
--	--	--

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки вставкой по убыванию.

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.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_5/tests/output.txt", 'w')

def selection_sort(n, A):
    for i in range(n - 1):
        min_local = i
        for j in range(i + 1, n):
            if A[j] < A[min_local]:
                min_local = j
        A[i], A[min_local] = A[min_local], A[i]
    return A

with open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_5/tests/input.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 10 ** 3:
        nums = list(map(int, list(file[1].split(' '))))
        if all([abs(n) <= 10 ** 9 for x in nums]):
```

```

        file_output.write(' '.join(map(str, selection_sort(n, nums))))
    else:
        print('Введите другое число')
    else:
        print('Неверный ввод данных')

file_output.close()

```

Текстовое объяснение решения.

Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то с помощью split() делю вторую строку в файле по пробелам, делая массив. С помощью map() делаю элементы этого массива числами. Записываю результат в переменную nums. С помощью all() убеждаюсь, что все элементы удовлетворяют условию, записываю в file_output результат функции selection_sort, делая с помощью map() все элементы строками, и объединяю результат в строку через join. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция selection_sort:

Параметрами функции являются n, число элементов массива, и A, сам массив. С помощью цикла for прохожусь по элементам массива. Эта функция работает поэтапно, разбивая список на отсортированную и неотсортированную части. В неотсортированной части находится локальный минимум и переставляется в конец отсортированного списка.

Результат работы кода на примерах из текста задачи:

```

lab1_5.1.py x input.txt x
1 6
2 31 41 59 26 41 58

lab1_5.1.py x input.txt x output.txt x
1 26 31 41 41 58 59

C:\Users\Slawa\Desktop\Uni\venv\Script
Время работы: 0.034100026823580265
Память: 0.47265625 K6

```

Результат работы кода на максимальных и минимальных значениях:

Минимальные значения

```
import time
import tracemalloc
from asd_itmo.lab1.task_5.src.lab1_5 import selection_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [0]

result = selection_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time) * 1000))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Sc
Время работы: 0.01500002690590918
Память: 0.384765625 Кб
```

Максимальные значения

```
import time
import tracemalloc
import random
from asd_itmo.lab1.task_5.src.lab1_5 import selection_sort

start_time = time.perf_counter()
tracemalloc.start()
input_arr = [random.randint(-10 ** 9, 10 ** 9 + 1) for _ in range(10 ** 3)]

result = selection_sort(len(input_arr), input_arr)
print('Время работы: ' + str((time.perf_counter() - start_time)))
print('Память: ' + str(tracemalloc.get_traced_memory()[1]/1024) + ' Кб')
tracemalloc.stop()
```

```
C:\Users\Slawa\Desktop\Uni\venv\Scri
Время работы: 0.35512049999670126
Память: 38.529296875 Кб
```

	Время выполнения	Затраты памяти
Нижняя граница диапазона значений	0.015	0.3847

ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ		
Пример из задачи	0.0341	0.472656
Верхняя граница диапазона значений ВХОДНЫХ ДАННЫХ ИЗ ТЕКСТА ЗАДАЧИ	0.35512	38.5292

Вывод по задаче:

В ходе работы над задачей мной был изучен способ сортировки выбором. Была изучена эффективность этого метода.

Задача №10. Палиндром

Текст задачи.

Палиндром — это строка, которая читается одинаково как справа налево, так и слева направо. На вход программы поступает набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы.

Требуется из данных букв по указанным правилам составить палиндром наибольшей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

- Формат входного файла (input.txt). В первой строке входных данных содержится число n ($1 \leq n \leq 100000$). Во второй строке задается последовательность из n больших латинских букв (буквы записаны без пробелов).
- Формат выходного файла (output.txt). В единственной строке выходных данных выдайте искомый палиндром.
- Ограничение по времени. 1 сек.
- Ограничение по памяти. 64 мб.

Листинг кода.

```
file_output =
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_10/tests/output.txt",
'w')
```

```

def palindrom(str_in):
    if len(str_in) == len(set(str_in)):
        return sorted(list(str_in))[0]
    else:
        dict_nechet_letters = dict()
        chet_letters = ''
        set_list = list(set(str_in))

        for letter in sorted(set_list):
            counter = str_in.count(letter)
            if counter % 2 == 0:
                chet_letters += letter * (counter // 2)
            else:
                dict_nechet_letters[letter] = counter

        result = list()
        if dict_nechet_letters:
            for letter in dict_nechet_letters:
                res = chet_letters + letter * dict_nechet_letters[letter] +
chet_letters[::-1]
                if result:
                    if len(res) > len(result[0]):
                        result = list()
                    elif len(res) < len(result[0]):
                        continue
                result.append(res)
            return min(result)
        return chet_letters + chet_letters[::-1]

with
open("C:/Users/Slawa/Desktop/Uni/asd_itmo/lab1/task_10/tests/input_1.txt",
'r') as f:
    file = f.readlines()
    n = int(file[0])
    if 1 <= n <= 100000:
        letters = list(file[1])
        result = palindrom(letters)
        file_output.write(result)
    else:
        print('Неверный ввод данных')

file_output.close()

```

Текстовое объяснение решения.

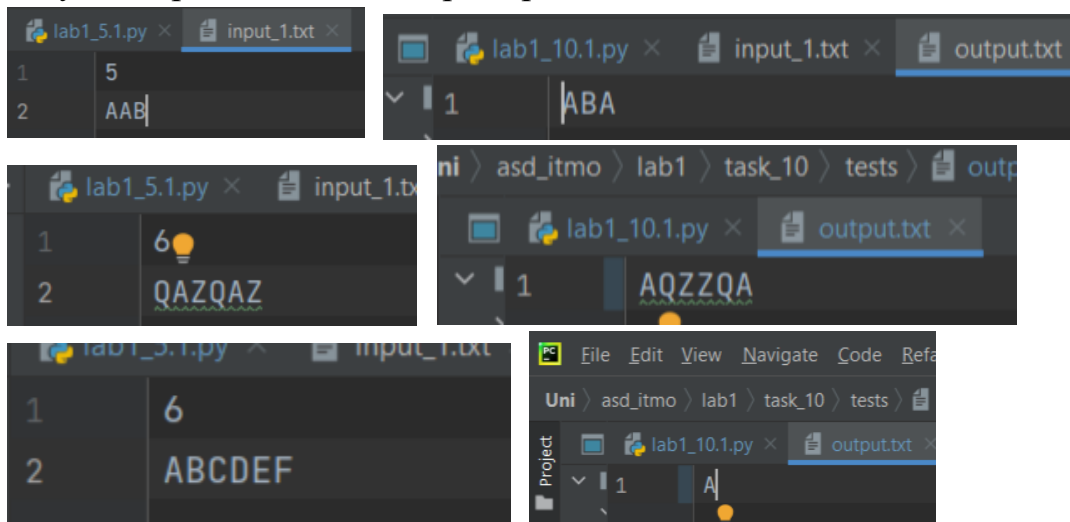
Открываю файл output.txt в режиме записи с помощью open(). С помощью оператора with открываю файл input.txt в режиме чтения. Записываю в переменную file все данные из файла с помощью readlines(). В переменную n записываю число элементов массива. Далее проверяю условие: если ввод данных не корректен, то предупреждаю пользователя об этом. Если же ввод данных корректен, то в переменную letter записываю строку из букв.

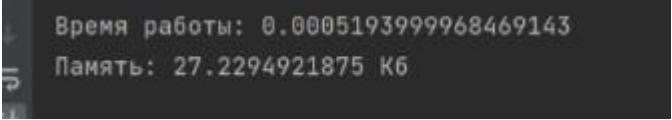
Записываю результат функции `palindrom` в переменную `result`. Записываю в `file_output` результат. В ином случае прошу ввести корректные данные. Закрываю файл.

Функция *palindrom*:

Параметром функции является `str_in`, строка из букв. Проверяю условие: если длина `str_in` равна количеству различных букв, то вывожу первую букву из отсортированной по возрастанию строке. В ином случае, определяю словарь нечётных букв `dict_nechet_letters` и строку чётных букв `chet_letters`. Прохожусь по каждой букве из `set(str_in)`. Считаю, сколько раз буква появляется в строке. Если количество чётное, то записываю её в строку `chet_letters` в количестве, равном половине количества вхождения этой буквы в строку. Если количество нечётное, то записываю её в словарь `dict_nechet_letters`, присваивая значения ключа количество вхождения этой буквы в строку. Создаю переменную `result`. Если есть нечётные буквы, то прохожусь по ключам словаря. В переменную `res` записываю строку, складывая чётные буквы по краям (справа наоборот с помощью среза) и этой букве. Если в списке `result` есть какие-то значения, то сравниваю их длину с `res`. Если она больше, то обновляю список `result`, записывая туда значение `res`, а если меньше, то пропускаю итерацию. В ином случае возвращаю строку чётных букв, сложенную со этой же строкой в обратном порядке.

Результат работы кода на примерах из текста задачи:





```
Время работы: 0.0005193999968469143  
Память: 27.2294921875 Кб
```

Вывод по задаче:

В ходе работы над задачей я изучила, как находить палиндром строки.

Вывод

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