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](#_Toc179199768)

[Задачи по варианту 3](#_Toc179199769)

[Задача №1. Сортировка вставкой 3](#_Toc179199770)

[Задача №2. Сортировка вставкой 6](#_Toc179199771)

[Задача №6. Пузырьковая сортировка 10](#_Toc179199772)

[Дополнительные задачи 14](#_Toc179199773)

[Задача №3. Сортировка вставкой по убыванию 14](#_Toc179199774)

[Задача №5. Сортировка выбором 17](#_Toc179199775)

[Задача №10. Палиндром 20](#_Toc179199776)

[Вывод 23](#_Toc179199777)

# 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 ≤ n ≤ 103) — число элементов в массиве. Во второй строке находятся n различных целых чисел, по модулю не превосходящих 109
* 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()

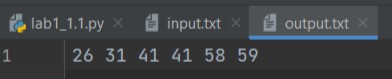
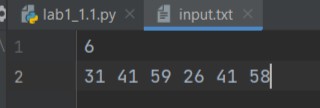
Текстовое объяснение решения.

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. Дальше 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:

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения, с | Затраты памяти, Кб |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.0153 | 0.386 |
| 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. Дальше 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.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | Время выполнения, c | 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] ≤ A′[2] ≤ ... ≤ A′[n], где A′- выход процедуры Bubble\_Sort, a 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. Дальше 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:

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

Автоматически созданное описание



Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

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) + ' Кб')  
tracemalloc.stop()

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

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) + ' Кб')  
tracemalloc.stop()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | 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 Swap. 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. Дальше 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 последовательно сравнения с элементами отсортированного списка первый элемент из неотсортированного встаёт на своё место.

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

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()

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

|  |  |  |
| --- | --- | --- |
|  | 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 диапазона значений входных данных из текста задачи | 0.3508 | 38.6542 |

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. Дальше 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*:

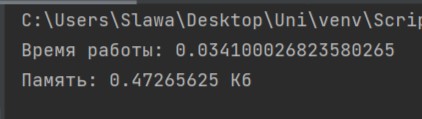
Параметрами функции являются n, число элементов массива, и A, сам массив. С помощью цикла for прохожусь по элементам массива. Эта функция работает поэтапно, разбивая список на отсортированную и неотсортированную части. В неотсортированной части находится локальных минимум и переставляется в конец отсортированного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

Результат работы кода на примерах из текста задачи:

Изображение выглядит как текст, Шрифт, снимок экрана, число

Автоматически созданное описание Изображение выглядит как текст, Шрифт, снимок экрана, часы

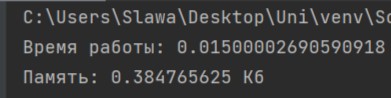
Автоматически созданное описание



Результат работы кода на максимальных и минимальных значениях:

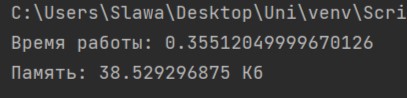
Минимальные значения

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()



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()



|  |  |  |
| --- | --- | --- |
|  | Время выполнения | Затраты памяти |
| Нижняя граница диапазона значений входных данных из текста задачи | 0.015 | 0.3847 |
| 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 ≤ n ≤ 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. 1сек.
* 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. Дальше 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:

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, Мультимедийное программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, часы

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

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.