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 23

Выполнил:

Хабиби. Я.

К3140

Проверил:

Афанасьев А.В.

Санкт-Петербург

2024г.

# 

[Содержание отчета 1](#_Toc179827353)

[Задачи по варианту 2](#_Toc179827354)

[Задача №1. Сортировка вставкой 2](#_Toc179827355)

[Задача №6. Пузырьковая сортировка 3](#_Toc179827356)

[Задача №8. Секретарь своп 5](#_Toc179827357)

[Вывод 6](#_Toc179827358)

## 

## Задача №1. Сортировка вставкой

Текст задачи: Используя код процедуры Insertion-sort, напишите программу и проверьте сортировку массива A = {31, 41, 59, 26, 41, 58}. Решение:

-------------------------------------------------------------------------- def insertion\_sort(arr):  
 for i in range(1, len(arr)): # начинаем со второго элемента  
 key = arr[i]  
 j = i - 1  
  
 while j >= 0 and key < arr[j]:  
 arr[j + 1] = arr[j] # сдвигаем элемент вправо  
 j -= 1  
  
 arr[j + 1] = key # вставляем key на его место  
 return arr

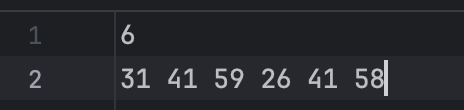
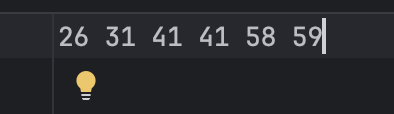
with open('input.txt', 'r') as infile:  
 n = int(infile.readline().strip()) # Читаем количество элементов  
 arr = list(map(int, infile.readline().strip().split())) # Читаем массив  
  
# проверка условия  
  
if 1 <= n <= 1000 and len(arr) == n and all(abs(x) <= 10\*\*9 for x in arr):  
 sorted\_arr = insertion\_sort(arr) # Если всё в порядке, сортируем  
 with open('output.txt', 'w') as outfile:  
 outfile.write(' '.join(map(str, sorted\_arr))) # Записываем отсортированный массив  
else:  
 print("Ошибка ввода данных.")

-------------------------------------------------------------------------- Объяснение решения:

Решение первой задачи состоит из двух частей: сортировка массива вставками и работа с файлами для чтения и записи данных.

1. **Сортировка вставками (insertion\_sort)**:
   * Мы начинаем с 1-го элемента (индекс 1), считаем его "ключом", и сравниваем его с предыдущими элементами.
   * Если элемент слева больше "ключа", мы сдвигаем его вправо.
   * Процесс продолжается, пока мы не найдём правильное место для "ключа", и затем вставляем его.
   * Таким образом массив постепенно становится отсортированным.

Результат работы кода на примерах из текста задачи:

|  |  |  |
| --- | --- | --- |
|  | Время выполнения  (в секундах) | Затраты памяти  (в МБ) |
| 1  0 | 0.0000038 | 19.5 |
| 6  31 41 59 26 41 58 | 0.0000083 | 19.89 |
| 1000  -749706382 149187416  -921224543 684411683… | 0.0231 | 19.74 |

Вывод по задаче: Мы научились писать алгоритм сортировки вставкой, проверять значения, переданные в функцию, при помощи декораторов, а также писать тесты для проверки алгоритма сортировки.

## Задача №6. Пузырьковая сортировка

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

Bubble\_Sort(A):

for i = 1 to A.length - 1

for j = A.length downto i+1

if A[j] < A[j-1]

ïîìåíÿòü A[j] è A[j-1] ìåñòàìè

Напишите код на Python и докажите корректность пузырьковой сортиров- ки. Для доказательства корректоности 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.

-------------------------------------------------------------------------- def bubble\_sort(A):  
 n = len(A)  
 for i in range(n - 1):  
 for j in range(n - 1, i, -1):  
 if A[j] < A[j - 1]:  
 A[j], A[j - 1] = A[j - 1], A[j]  
 return A

# Чтение данных из файла input.txt  
with open('input.txt', 'r') as infile:  
 n = int(infile.readline().strip()) # Читаем количество элементов  
 arr = list(map(int, infile.readline().strip().split())) # Читаем массив  
  
# Проверки  
if 1 <= n <= 1000 and len(arr) == n and all(abs(x) <= 10 \*\* 9 for x in arr):  
 # Сортируем массив  
 sorted\_arr = bubble\_sort(arr)  
 # Запись отсортированных данных в файл output.txt  
 with open('output.txt', 'w') as outfile:  
 outfile.write(' '.join(map(str, sorted\_arr))) # Записываем отсортированный массив  
else:  
 print("Ошибка: входные данные не соответствуют ограничениям.")

-------------------------------------------------------------------------- Объяснение решения:

Решение 6-й задачи использует алгоритм **пузырьковой сортировки** (bubble\_sort), чтобы отсортировать массив по неубыванию:

1. **Идея пузырьковой сортировки**:
   * Пузырьковая сортировка работает путём многократного прохода по массиву, где на каждой итерации самые "тяжёлые" элементы "всплывают" вправо (как пузырьки).
   * Элементы сравниваются попарно, и если текущий элемент больше следующего, они меняются местами, чтобы самый большой элемент переместился в конец.
2. **Реализация**:
   * Переменная n хранит длину массива.
   * **Внешний цикл** (for i in range(n - 1)) отвечает за количество проходов по массиву, так как нам нужно сделать несколько "всплываний", чтобы все элементы встали на свои места.
   * **Внутренний цикл** (for j in range(n - 1, i, -1)) проходит по массиву справа налево, сравнивая каждый элемент с предыдущим.
     + Если текущий элемент (A[j]) меньше предыдущего (A[j - 1]), они меняются местами, чтобы больший элемент "всплыл" дальше вправо.

--------------------------------------------------------------------------

Вывод по задаче:

В этой задаче мы поняли, что принцип сортировки вставкой можно использовать не только для сортировки по возрастанию, но и для сортировки по убыванию. Поняли, как работает множественное/позиционное присваивание и реализовали его через функцию swap (). Убедились, что сортировку вставкой можно реализовать с помощью рекурсии, но такой метод значительно уступает по времени (в 100 раз дольше по времени при длине массива в 1000 элементов) методу с использованием вложенного цикла.

## Задача №8. Секретарь своп

1. 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.
2. **Формат входного файла (input.txt).** В первой строке входного файла со- держится число n (3 ≤ n ≤ 5000) — число элементов в массиве. Во второй строке находятся n целых чисел, по модулю не превосходящих 109. Числа могут совпадать друг с другом.
3. **Формат выходного файла (output.txt).** В первых нескольких строках вы- ведите осуществленные Вами операции перестановки элементов. Каждая строка должна иметь следующий формат:

Swap elements at indices X and Y.

Здесь X и Y — различные индексы массива, элементы на которых нужно переставить(1≤X,Y ≤n).МистерСвоплюбитпорядок,поэтомусделайте так,чтобыX <Y. После того, как все нужные перестановки выведены, выведите следующую фразу: No more swaps needed.

-------------------------------------------------------------------------- def fswaps(arr):  
 n = len(arr)  
 swaps = []  
  
 # Реализация кастомного алгоритма, чтобы повторять пример из задачи  
 for i in range(n):  
 min\_idx = i  
 for j in range(i + 1, n):  
 if arr[j] < arr[min\_idx]:  
 min\_idx = j  
  
 # Меняем местами только, если элемент найден, и он отличается  
 if min\_idx != i:  
 arr[i], arr[min\_idx] = arr[min\_idx], arr[i]  
 swaps.append((i + 1, min\_idx + 1))  
  
 return swaps  
  
  
# Чтение данных из файла input.txt  
with open('input.txt', 'r') as infile:  
 n = int(infile.readline().strip()) # Читаем количество элементов  
 arr = list(map(int, infile.readline().strip().split())) # Читаем массив  
  
# Проверки  
if 3 <= n <= 5000 and len(arr) == n and all(abs(x) <= 10 \*\* 9 for x in arr):  
 # Сортируем массив с записью всех перестановок  
 swaps = fswaps(arr)  
  
 # Запись результатов в файл output.txt  
 with open('output.txt', 'w') as outfile:  
 for swap in swaps:  
 outfile.write(f"Swap elements at indices {swap[0]} and {swap[1]}\n")  
 outfile.write("No more swaps needed.\n")  
else:  
 print("Ошибка: входные данные не соответствуют ограничениям.")

-------------------------------------------------------------------------- Объяснение решения:

В этой задаче мы решали проблему сортировки массива, при этом фиксируя все произведённые перестановки. Для этого использовался модифицированный алгоритм сортировки выбором. Основная идея заключается в том, чтобы на каждом шаге находить минимальный элемент в оставшейся части массива и менять его местами с текущим элементом, если он отличается. Мы фиксировали каждую такую перестановку в списке swaps, чтобы затем записать все шаги сортировки в файл.

Сначала мы определяем длину массива и проходим по нему с помощью цикла. Для каждого элемента ищем минимальный элемент в оставшейся части массива. Если найденный минимальный элемент не совпадает с текущим элементом, они меняются местами, и мы добавляем пару индексов (в формате, начиная с 1) в список swaps. Таким образом, мы фиксируем все перестановки, которые происходят в процессе сортировки.

Задача заключалась не только в том, чтобы отсортировать массив, но и зафиксировать порядок перестановок, чтобы записать каждое действие в выходной файл. После завершения сортировки мы открываем файл output.txt и записываем туда все перестановки в требуемом формате. В конце добавляем строку "No more swaps needed."

Результат Работы:

Изображение выглядит как текст, Шрифт, снимок экрана, черный

Автоматически созданное описание

# Вывод

В ходе лабораторной работы мы вспомнили, как работать с файлами, как писать модульные тесты, как измерять затраты памяти и время работы алгоритма, а также изучили и реализовали простые алгоритмы сортировки и поиска элементов в массиве, научились решать нестандартные задачи