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 8

Выполнила:
Иванова Аайа Гаврильевна
К3141

Проверил:
Афанасьев А.В.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

Задание №1. Сортировка вставкой

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 ($1 \leq n \leq 10^3$) — 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.

1.

Листинг кода:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
        arg = arr[i]
        j = i - 1
        while j >= 0 and arg < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = arg
    return arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array_1 = f.readlines()
    array_2 = insertion_sort(list(map(int,
array_1.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array_2))),
file=f)
```

Текстовое объяснение задачи:
Проходится по элементам массива слева направо, вставляя каждый новый элемент в правильное место среди уже отсортированных элементов. Это подобно сортировке карт в руке.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file
__), '../src')))
from sort1 import insertion_sort


class TestInsertionSort(unittest.TestCase):
    """
    Набор тестов для функции сортировки вставками.
    """

    def test_sorted_array(self):
        """
        Проверка работы на уже отсортированном
массиве.
        """
        self.assertEqual(insertion_sort([1, 2, 3, 4,
5]), [1, 2, 3, 4, 5])

    def test_reverse_sorted_array(self):
        """
        Проверка работы на массиве, отсортированном в
обратном порядке.
        """
        self.assertEqual(insertion_sort([5, 4, 3, 2,
1]), [1, 2, 3, 4, 5])

    def test_unsorted_array(self):
        """
        Проверка работы на случайном
неотсортированном массиве.
        """
        self.assertEqual(insertion_sort([31, 41, 59,
```

```
26, 41, 58]), [26, 31, 41, 41, 58, 59])

    def test_array_with_duplicates(self):
        """
        Проверка работы на массиве с дубликатами.
        """
        self.assertEqual(insertion_sort([2, 3, 2, 3,
1]), [1, 2, 2, 3, 3])

    def test_single_element_array(self):
        """
        Проверка работы на массиве с одним элементом.
        """
        self.assertEqual(insertion_sort([42]), [42])

    def test_empty_array(self):
        """
        Проверка работы на пустом массиве.
        """
        self.assertEqual(insertion_sort([]), [])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять сортировку вставкой массива целых чисел.

Задание №3. Сортировка вставкой по убыванию

3 задача. 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:

```
def insertion_sort(arr):
    for i in range(1, len(arr)):
```

```
        arg = arr[i]
        j = i - 1
        while j >= 0 and arg > arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = arg
    return arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array_1 = f.readlines()
    array_2 = insertion_sort(list(map(int,
array_1.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array_2))),
file=f)
```

Текстовое объяснение задачи:
Модифицированная сортировка вставками, которая упорядочивает элменты в массиве по убыванию. Логика аналогична обычной сортировке вставками, только вставка идёт в обратном порядке.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file
__), '../src')))
from sort3 import insertion_sort


class
TestInsertionSortDescending(unittest.TestCase):
    """
    Набор тестов для функции сортировки вставками по
убыванию.
    """
```

```
    def test_basic_case(self):
        """
        Проверка корректной сортировки по убыванию на
обычном случае.
        """
        self.assertEqual(insertion_sort([4, 3, 2,
1]), [4, 3, 2, 1])
        self.assertEqual(insertion_sort([1, 2, 3,
4]), [4, 3, 2, 1])

    def test_with_duplicates(self):
        """
        Проверка корректной сортировки при наличии
дубликатов в массиве.
        """
        self.assertEqual(insertion_sort([3, 3, 1, 2,
3]), [3, 3, 3, 2, 1])

    def test_single_element(self):
        """
        Проверка корректности работы на массиве с
одним элементом.
        """
        self.assertEqual(insertion_sort([42]), [42])

    def test_empty_array(self):
        """
        Проверка работы на пустом массиве.
        """
        self.assertEqual(insertion_sort([]), [])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы изменили обычную сортировку вставкой массива целых чисел так, чтобы массив был упорядочен по убыванию.

Задание №5. Сортировка выбором

5 задача. 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:

```
def s_sort(arr):
    for i in range(len(arr)):
        min_idx = i
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array_1 = f.readlines()
    array_2 = s_sort(list(map(int,
array_1.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array_2))),
file=f)
```

Текстовое объяснение задачи:
Алгоритм проходит по массиву, находит минимальный элемент и меняет его местами с текущим эдементом. Процесс повторяется для каждой позиции массива.

Листинг тестов:

```
import unittest
import sys
```

```
import os

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file
__), '../src')))
from sort5 import s_sort


class TestSelectionSort(unittest.TestCase):
    """
    Набор тестов для функции сортировки выбором.
    """

    def test_basic_case(self):
        """
        Проверка корректной сортировки на обычном
массиве.
        """
        self.assertEqual(s_sort([31, 41, 59, 26, 41,
58]), [26, 31, 41, 41, 58, 59])

    def test_sorted_array(self):
        """
        Проверка корректности работы на уже
отсортированном массиве.
        """
        self.assertEqual(s_sort([1, 2, 3, 4, 5]),
[1, 2, 3, 4, 5])

    def test_reverse_sorted_array(self):
        """
        Проверка корректности работы на массиве,
отсортированном в обратном порядке.
        """
        self.assertEqual(s_sort([5, 4, 3, 2, 1]),
[1, 2, 3, 4, 5])

    def test_array_with_duplicates(self):
        """
        Проверка корректной работы на массиве с
дубликатами.
        """
        self.assertEqual(s_sort([3, 3, 2, 1, 3]),
```

```
[1, 2, 3, 3, 3])

    def test_single_element(self):
        """
        Проверка корректности работы на массиве с
одним элементом.
        """
        self.assertEqual(s_sort([42]), [42])

    def test_empty_array(self):
        """
        Проверка корректной работы на пустом массиве.
        """
        self.assertEqual(s_sort([]), [])


if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы научились осуществлять сортировку выбором массива целых чисел.

Дополнительные задачи

Задание №2. Сортировка вставкой +

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

Листинг кода:

```
def insertion_sort(arr):
    index = [0]
    counter = 0
    for i in range(1, len(arr)):
        for j in range(i - 1, -1, -1):
            if arr[i] < arr[j]:
                arr[i], arr[j] = arr[j], arr[i]
                i, j = j, i
        counter += 1
        index.append(counter)
```

```
    return index, arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array_1 = f.readlines()
    indexes, array_2 = insertion_sort(list(map(int,
array_1.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, indexes))),
file=f)
        print(' '.join(list(map(str, array_2))),
file=f)
```

Текстовое объяснение задачи:
Проходится по элементам массива слева направо, вставляя каждый новый элемент в правильное место среди уже отсортированных элементов. Также алгоритм отслеживает, как менялись индексы элементов в процессе сортировки.

Листинг тестов:

```
import unittest
import sys
import os
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__),
'../src')))
from sort2 import insertion_sort

class TestInsertionSortWithIndices(unittest.TestCase):
    """
    Набор тестов для функции сортировки вставками с
отслеживанием перемещений элементов.
    """

    def test_basic_case(self):
        """
        Проверка корректной сортировки и возвращаемых
индексов при обычном случае.
        """
        indexes, sorted_array = insertion_sort([4, 3, 2,
1])
        self.assertEqual(sorted_array, [1, 2, 3, 4])
        self.assertEqual(indexes, [0, 1, 2, 3])
```

```
    def test_with_duplicates(self):
        """
        Проверка корректной сортировки и возвращаемых
индексов при наличии дубликатов в массиве.
        """
        indexes, sorted_array = insertion_sort([3, 3, 1,
2, 3])
        self.assertEqual(sorted_array, [1, 2, 3, 3, 3])
        self.assertEqual(indexes, [0, 1, 2, 3, 4])

    def test_sorted_array(self):
        """
        Проверка корректности работы на уже
отсортированном массиве.
        """
        indexes, sorted_array = insertion_sort([1, 2, 3,
4, 5])
        self.assertEqual(sorted_array, [1, 2, 3, 4, 5])
        self.assertEqual(indexes, [0, 1, 2, 3, 4])

    def test_single_element(self):
        """
        Проверка корректности работы на массиве с одним
элементом.
        """
        indexes, sorted_array = insertion_sort([42])
        self.assertEqual(sorted_array, [42])
        self.assertEqual(indexes, [0])

    def test_empty_array(self):
        """
        Проверка работы на пустом массиве.
        """
        indexes, sorted_array = insertion_sort([])
        self.assertEqual(sorted_array, [])
        self.assertEqual(indexes, [0])

if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче: В ходе решения данной задачи мы научились осуществлять сортировку вставкой массива целых чисел и разработали алгоритм по сохранению данных о новых индексах элементов при их обработке.

Задание №4. Линейный поиск

4 задача. Линейный поиск

Рассмотрим задачу поиска.

- **Формат входного файла**. Последовательность из n чисел $A = a_1, a_2, \ldots, a_n$ 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 в 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.
- Дополнительно: попробуйте найти свинью, как в лекции. Используйте во входном файле последовательность слов из лекции, и найдите соответствующий индекс.

Листинг кода:

```
def l_search(arr, v):
    result = list()
    for i, num in enumerate(arr):
        if num == v:
            result.append(str(i))
    return result if result else '-1'


if __name__ == '__main__':
    with open('input.txt') as f:
        array, v = f.readlines()
    result = l_search(array.split(), v)
    result = ', '.join(result) if isinstance(result,
list) else '-1'
    with open('output.txt', 'w') as f:
        print(result, file=f)
```

Текстовое объяснение задачи:
Алгоритм, который проходит по всем элементам массива, чтобы найти заданное значение. Если значение найдено, возвращаются индексы всего его вхождений. Если вхождений нет, возвращается -1.

Листинг тестов:

```python
import unittest
import sys
import os

sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file__),
'../src')))
from sort4 import l_search


class TestLinearSearch(unittest.TestCase):
    """
    Набор тестов для функции линейного поиска.
    """

    def test_element_found(self):
        """
        Проверка корректного поиска элемента, который
присутствует в массиве.
        """
        self.assertEqual(l_search(['10', '20', '30',
'40'], '30'), ['2'])

    def test_element_not_found(self):
        """
        Проверка корректной обработки случая, когда
элемент отсутствует в массиве.
        """
        self.assertEqual(l_search(['10', '20', '30',
'40'], '50'), '-1')

    def test_multiple_occurrences(self):
        """
        Проверка корректного поиска элемента, который
встречается несколько раз в массиве.
        """
        self.assertEqual(l_search(['10', '20', '30', '30',
'40'], '30'), ['2', '3'])

    def test_single_element_found(self):
        """
        Проверка корректности работы на массиве с одним
элементом, который совпадает с искомым.
        """
        self.assertEqual(l_search(['42'], '42'), ['0'])
```

```
    def test_single_element_not_found(self):
        """
        Проверка корректности работы на массиве с одним
элементом, который не совпадает с искомым.
        """
        self.assertEqual(l_search(['42'], '43'), '-1')

    def test_empty_array(self):
        """
        Проверка корректной работы на пустом массиве.
        """
        self.assertEqual(l_search([], '10'), '-1')


if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:

В ходе решения данной задачи мы научились осуществлять линейный поиск некоторого элемента среди массива элементов.

Задание №6. Пузырьковая сортировка

6 задача. Пузырьковая сортировка

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 корректоности процедуры вам необходимо доказать, что она завершается и что $A'[1] \leq A'[2] \leq ... \leq A'[n]$, 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:

```
def b_sort(arr):
    l = len(arr)
    for i in range(l):
        for j in range(0, l - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1],
arr[j]
    return arr


if __name__ == '__main__':
    with open('input.txt') as f:
        n, array_1 = f.readlines()
    array_2 = b_sort(list(map(int,
array_1.split())))
    with open('output.txt', 'w') as f:
        print(' '.join(list(map(str, array_2))),
file=f)
```

Текстовое объяснение задачи:
Алгоритм многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке. На каждой итерации наибольший элемент отправляется в конец массива.

Листинг тестов:

```
import sys
import os
sys.path.insert(0,
os.path.abspath(os.path.join(os.path.dirname(__file
__), '../src')))
from sort6 import b_sort


class TestBubbleSort(unittest.TestCase):
    """
    Набор тестов для функции пузырьковой сортировки.
    """

    def test_basic_case(self):
        """
```

```
        Проверка корректной сортировки на обычном
массиве.
        """
        self.assertEqual(b_sort([31, 41, 59, 26, 41,
58]), [26, 31, 41, 41, 58, 59])

    def test_sorted_array(self):
        """
        Проверка корректности работы на уже
отсортированном массиве.
        """
        self.assertEqual(b_sort([1, 2, 3, 4, 5]),
[1, 2, 3, 4, 5])

    def test_reverse_sorted_array(self):
        """
        Проверка корректности работы на массиве,
отсортированном в обратном порядке.
        """
        self.assertEqual(b_sort([5, 4, 3, 2, 1]),
[1, 2, 3, 4, 5])

    def test_array_with_duplicates(self):
        """
        Проверка корректной работы на массиве с
дубликатами.
        """
        self.assertEqual(b_sort([3, 3, 2, 1, 3]),
[1, 2, 3, 3, 3])

    def test_single_element(self):
        """
        Проверка корректности работы на массиве с
одним элементом.
        """
        self.assertEqual(b_sort([42]), [42])

    def test_empty_array(self):
        """
        Проверка корректной работы на пустом массиве.
        """
        self.assertEqual(b_sort([]), [])
```

```
if __name__ == '__main__':
    unittest.main()
```

Вывод по задаче:
В ходе решения данной задачи мы научились осуществлять пузырьковую сортировку массива целых чисел, доказывать корректность ее работы.

Вывод

Эта лабораторная работа помогла понять принципы работы основных алгоритмов сортировки, их различные подходы к решению задачи упорядочивания данных. Каждый алгоритм демонстрирует уникальные аспекты сортировки, такие как выбор минимального элемента, использование пузырькового подхода или вставка элементов в уже отсортированную часть массива.