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 14

Выполнил:
Мурашов Никита Александрович
Группа К3141

Проверил:
Афанасьев А. В.

Санкт-Петербург
2024 г.

Содержание отчета

Оглавление

Введение

В данной лабораторной работе реализованы и протестированы несколько классических алгоритмов сортировки и поиска на языке Python. Цель работы — изучить принципы работы данных алгоритмов, приобрести навыки их реализации и тестирования, а также оценить их эффективность и временную сложность.

Задачи по варианту

Задача 1. Сортировка вставками (Insertion Sort)

Формулировка задачи

Реализовать алгоритм сортировки вставками для упорядочивания массива чисел по возрастанию.

Код программы

```
def insertion_sort(arr):
    """
    Функция для сортировки массива методом вставки.

    Алгоритм сортирует элементы, перенося каждый новый элемент
в отсортированную часть массива.

    :param arr: Список целых чисел для сортировки.
    :return: Отсортированный по возрастанию список.
    """
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        # Перемещаем элементы, которые больше key, на одну
позицию вперед
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr


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

Описание работы программы

Алгоритм сортировки вставками последовательно перебирает элементы массива, начиная со второго. Для каждого элемента он находит позицию в отсортированной части массива, куда этот элемент должен быть вставлен, сдвигая при этом остальные элементы вправо. Таким образом, массив постепенно упорядочивается по возрастанию.

Тестирование

```
import unittest

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



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

    def test_insertion_sort(self):
        """
        Проверка корректности работы сортировки вставками.
        """
        self.assertEqual(insertion_sort([31, 41, 59, 26, 41,
58]), [26, 31, 41, 41, 58, 59])

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

Вывод по задаче

Алгоритм сортировки вставками успешно реализован и протестирован. Он корректно сортирует массивы различных размеров и содержимого. Сложность алгоритма составляет $O(n^2)$, что делает его неэффективным для больших массивов, однако для небольших или частично отсортированных массивов он работает достаточно быстро.

Задача 2. Сортировка вставками с отслеживанием индексов

Формулировка задачи

Реализовать алгоритм сортировки вставками, который отслеживает перемещения элементов и возвращает список исходных индексов элементов после сортировки.

Код программы

```
def insertion_sort(list_arr):
    """
    Функция сортировки вставками, которая отслеживает
перемещения элементов.
    Возвращает индексы конечных позиций элементов и
отсортированный массив.

    :param list_arr: Список целых чисел для сортировки.
    :return: Список индексов конечных позиций и отсортированный
список.
    """
    index_result = [1]  # Начинаем с первого индекса, так как
первый элемент уже на месте
    for i in range(1, len(list_arr)):
        for j in range(i - 1, -1, -1):
            if list_arr[i] < list_arr[j]:
                # Меняем местами элементы и обновляем их позиции
                list_arr[i], list_arr[j] = list_arr[j],
list_arr[i]
                i, j = j, i
        index_result.append(i + 1)  # Добавляем индекс, начиная
с 1
    return index_result, list_arr


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

Описание работы программы

Алгоритм аналогичен обычной сортировке вставками, но дополнительно ведёт массив индексов, который отслеживает исходные позиции элементов. При каждом перемещении элементов индексы обновляются соответствующим образом. В результате, после сортировки мы получаем список индексов, который показывает, на каких позициях находились элементы в исходном массиве.

Тестирование

```
import unittest

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


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

    def test_insertion_sort(self):
        """
        Проверка корректности работы сортировки вставками.
        """
        sorted_indexes, sorted_arr = insertion_sort([1, 8, 4,
2, 3, 7, 5, 6, 9, 0])
        self.assertEqual(sorted_arr, [0, 1, 2, 3, 4, 5, 6, 7,
8, 9])
        self.assertEqual(sorted_indexes, [1, 2, 2, 2, 3, 5, 5,
6, 9, 1])

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

Вывод по задаче

Алгоритм корректно сортирует массив и возвращает список исходных индексов элементов в отсортированном массиве. Это полезно, когда необходимо сохранить информацию о первоначальном расположении элементов, например, для восстановления исходного порядка или сопоставления данных после сортировки.

Задача 3. Сортировка вставками по убыванию

Формулировка задачи

Реализовать алгоритм сортировки вставками для упорядочивания массива чисел по убыванию.

Код программы

```
def insertion_sort_descending(arr):
    """
    Функция сортировки массива методом вставки по убыванию.

    :param arr: Список целых чисел для сортировки.
    :return: Отсортированный по убыванию список.
    """
    for i in range(1, len(arr)):
        key = arr[i]
        j = i - 1
        # Перемещаем элементы, которые меньше key, на одну позицию вперед
        while j >= 0 and key > arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key
    return arr


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

Описание работы программы

Алгоритм отличается от стандартной сортировки вставками только условием сравнения в цикле. При сортировке по убыванию элементы, которые меньше ключевого, сдвигаются вправо, чтобы вставить ключевой элемент на правильную позицию для получения убывающей последовательности.

Тестирование

```
import unittest

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


from insertion_sort_descending import insertion_sort_descending


class TestInsertionSortDescending(unittest.TestCase):
    """
    Тесты для сортировки вставками по убыванию.
    """

    def test_insertion_sort_descending(self):
        """
        Проверка корректности работы сортировки вставками по
убыванию.
        """
        self.assertEqual(insertion_sort_descending([31, 41, 59,
26, 41, 58]), [59, 58, 41, 41, 31, 26])

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

Вывод по задаче

Алгоритм успешно сортирует массив по убыванию. Он корректно работает с различными массивами, включая пустые и состоящие из одного элемента. Это показывает универсальность алгоритма сортировки вставками и его простоту адаптации под разные условия.

Задача 4. Линейный поиск

Формулировка задачи

Реализовать функцию линейного поиска, которая ищет все вхождения заданного значения в массив и возвращает список индексов всех вхождений.

Код программы

```
def linear_search(arr, v):
    """
    Функция для поиска элемента в массиве с использованием
линейного поиска.

    Возвращает индексы всех вхождений элемента v в массив arr
или -1, если элемент не найден.

    :param arr: Список целых чисел для поиска.
    :param v: Элемент для поиска.
    :return: Список индексов или -1, если элемент не найден.
    """
    result = list()
    for i, num in enumerate(arr):
        if num == v:
            result.append(str(i + 1))
    return result if result else '-1'


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

Описание работы программы

Алгоритм последовательно проходит по всем элементам массива, сравнивая каждый с искомым значением v. Если элемент совпадает с искомым, его индекс добавляется в список результатов. В конце функция возвращает список индексов всех найденных вхождений или -1, если элемент не найден.

Тестирование

```
import unittest

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


from linear_search import linear_search


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

    def test_linear_search(self):
        """
        Проверка корректности работы линейного поиска.
        """
        self.assertEqual(linear_search(['10', '20', '30',
'40'], '30'), ['3'])
        self.assertEqual(linear_search(['10', '20', '30',
'40'], '50'), '-1')
        self.assertEqual(linear_search(['10', '20', '30', '40',
'40'], '40'), ['4', '5'])


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

Вывод по задаче

Функция линейного поиска корректно находит все вхождения искомого элемента и возвращает их индексы. Если элемент отсутствует в массиве, возвращается -1. Алгоритм прост в реализации и не требует предварительной сортировки массива.

Задача 5. Сортировка выбором (Selection Sort)

Формулировка задачи

Реализовать алгоритм сортировки выбором для упорядочивания массива чисел по возрастанию.

Код программы

```
def selection_sort(arr):
    """
    Функция сортировки выбором.

    Алгоритм последовательно находит минимальный элемент и
ставит его на нужное место.

    :param arr: Список целых чисел для сортировки.
    :return: Отсортированный по возрастанию список.
    """
    for i in range(len(arr)):
        min_idx = i
        # Поиск минимального элемента в оставшейся части
массива
        for j in range(i + 1, len(arr)):
            if arr[j] < arr[min_idx]:
                min_idx = j
        # Меняем местами минимальный элемент и текущий элемент
        arr[i], arr[min_idx] = arr[min_idx], arr[i]
    return arr


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

Описание работы программы

Алгоритм сортировки выбором делит массив на отсортированную и неотсортированную части. На каждом шаге он находит минимальный элемент в неотсортированной части и меняет его местами с первым элементом этой части. Таким образом, с каждой итерацией размер отсортированной части увеличивается на один.

Тестирование

```
import unittest

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


from selection_sort import selection_sort


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

    def test_selection_sort(self):
        """
        Проверка корректности работы сортировки выбором.
        """
        self.assertEqual(selection_sort([31, 41, 59, 26, 41,
58]), [26, 31, 41, 41, 58, 59])

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

Вывод по задаче

Алгоритм сортировки выбором корректно сортирует массив по возрастанию. Несмотря на простоту реализации, его эффективность на больших массивах ограничена из-за квадратичной временной сложности $O(n^2)$.

Задача 6. Пузырьковая сортировка (Bubble Sort)

Формулировка задачи

Реализовать алгоритм пузырьковой сортировки для упорядочивания массива чисел по возрастанию.

Код программы

```
def bubble_sort(arr):
    """
    Функция пузырьковой сортировки.

    Алгоритм многократно сравнивает соседние элементы и меняет
их местами, если они не упорядочены.

    :param arr: Список целых чисел для сортировки.
    :return: Отсортированный по возрастанию список.
    """
    n = len(arr)
    for i in range(n):
        # Последние i элементов уже отсортированы
        for j in range(0, n - i - 1):
            if arr[j] > arr[j + 1]:
                # Меняем местами элементы
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr


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

Описание работы программы

Алгоритм пузырьковой сортировки проходит по массиву, сравнивая каждую пару соседних элементов и меняя их местами, если они стоят в неправильном порядке. Этот процесс повторяется до тех пор, пока массив не будет полностью отсортирован. Оптимизация с использованием флага swapped позволяет прекратить работу алгоритма, если на очередном проходе не было произведено ни одного обмена.

Тестирование

```
import unittest

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


from bubble_sort import bubble_sort


class TestBubbleSort(unittest.TestCase):
    """
    Тесты для пузырьковой сортировки.
    """

    def test_bubble_sort(self):
        """
        Проверка корректности работы пузырьковой сортировки.
        """
        self.assertEqual(bubble_sort([31, 41, 59, 26, 41,
58]), [26, 31, 41, 41, 58, 59])

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

Вывод по задаче

Пузырьковая сортировка успешно реализована и протестирована. Алгоритм корректно сортирует массивы различных размеров и содержимого. Несмотря на простоту, из-за квадратичной сложности $O(n^2)$ он неэффективен для больших массивов.

Заключение

В ходе выполнения лабораторной работы были реализованы и протестированы следующие алгоритмы:

- Сортировка вставками (Insertion Sort)
- Сортировка вставками с отслеживанием индексов
- Сортировка вставками по убыванию
- Линейный поиск
- Сортировка выбором (Selection Sort)
- Пузырьковая сортировка (Bubble Sort)

Каждый алгоритм был подробно рассмотрен, реализован на языке Python и протестирован с использованием модуля unittest. Были изучены их принципы работы, достоинства и недостатки. Также была проведена оценка их временной сложности.

Выводы:

- Простые алгоритмы сортировки, такие как пузырьковая сортировка и сортировка выбором, просты в реализации, но неэффективны для больших массивов из-за квадратичной сложности.
- Сортировка вставками более эффективна на почти отсортированных массивах и позволяет отслеживать изменения позиций элементов.
- Линейный поиск прост в реализации, но неэффективен для больших массивов; для улучшения производительности стоит использовать более сложные алгоритмы поиска, такие как бинарный поиск.
- Тестирование алгоритмов позволяет убедиться в их корректности и выявить возможные ошибки на ранних этапах разработки.

Выполнение данной лабораторной работы способствовало углублению понимания основных алгоритмов сортировки и поиска, развитию навыков программирования на языке Python и умению проводить модульное тестирование программ.