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 6

Выполнил:
Данилова А.В.
К3141

Проверила:
Артамонова В.Е.

Санкт-Петербург
2024 г.

Содержание отчета

Задачи по варианту

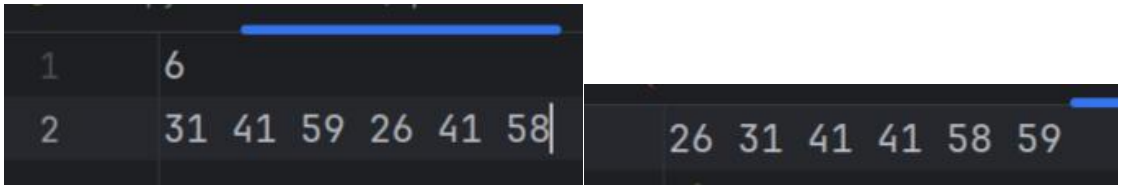
Задача №1. Сортировка вставкой

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

```
def insertion_sort():
    with open('input.txt', 'r') as file:
        n = -1
        for line in file:
            if n == -1:
                n = int(line)
                if not 1 <= n <= 1000:
                    with open('output.txt', 'w') as ans:  # проверка на
корректность
                        ans.write('---Incorrect data---')

        else:
            input_list = [int(elem) for elem in line.split() if
abs(int(elem)) <= 10e9]  # проверка на корректность
            for j in range(1, len(input_list)):
                key = input_list[j]
                i = j - 1
                while i >= 0 and input_list[i] > key:  # направление знака
меняется
                    input_list[i + 1] = input_list[i]
                    i = i - 1
                input_list[i + 1] = key
            with open('output.txt', 'w') as ans:
                ans.write(' '.join(map(str, input_list)))


insertion_sort()
```

Код открывает файл input.txt для чтения. Первая строка файла считывается и интерпретируется как количество чисел n. - Если n не находится в диапазоне от 1 до 1000, программа записывает сообщение об ошибке в файл output.txt и завершает выполнение. Далее считывает оставшиеся строки файла и создает список input_list, содержащий целые числа. При этом проверяется, что каждое число находится в диапазоне от -10^9 до 10^9. Применяется алгоритм сортировки вставками:
- Проходит по каждому элементу списка начиная со второго.
- Для каждого элемента (ключа) ищет его правильное место в уже отсортированной части списка, перемещая элементы, которые больше ключа, на одну позицию вправо.
После завершения сортировки записывает отсортированный список в файл output.txt

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

Вывод по задаче: В этой задаче был изучен метод сортировки вставкой.

Задача №2. Сортировка вставкой +

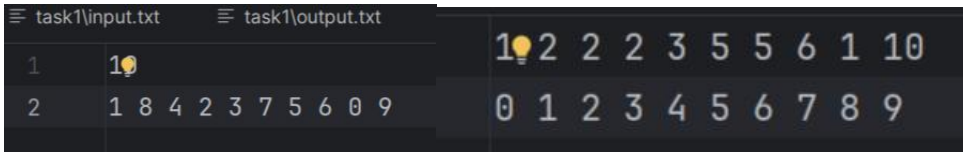
Измените процедуру Insertion-sort для сортировки таким образом, чтобы в выходном файле отображалось в первой строке n чисел, которые обозначают новый индекс элемента массива после обработки.

```
def insertion_sort_plus():
    with open('input.txt', 'r') as file:
        n = -1
        for line in file:
            if n == -1:
                n = int(line)
                if not 1 <= n <= 1000:
                    with open('output.txt', 'w') as ans:  # проверка на
корректность
                        ans.write('---Incorrect data---')

        else:
            input_list = [int(elem) for elem in line.split() if
abs(int(elem)) <= 10e9]  # проверка на корректность
            new_index = [1]  # создаем список новых индексов
            for j in range(1, len(input_list)):
                key = input_list[j]
                i = j - 1
                while i >= 0 and input_list[i] > key:  # направление знака
меняется
                    input_list[i + 1] = input_list[i]
                    i = i - 1
                input_list[i + 1] = key
                new_index.append(i + 2)  # прибавляется 1, т.к. в питоне
нумерация начинаеся с 0, а необходимо с 1
            with open('output.txt', 'w') as ans:
                ans.write(' '.join(map(str, new_index)) + '\n' + '
'.join(map(str, input_list)))
```

```
insertion_sort_plus()
```

К коду из первой задачи добавляется новый список с индексами, которой будет заполняться в каждой итерации цикла for по i новым индексом элемента.

Результат работы кода на примерах из текста задачи:



	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.0000078	19.75
10 1 8 4 2 3 7 5 6 9 0	0.000017	19.59

Вывод по задаче: Узнали, что, дополнив, изменив алгоритм сортировки вставкой, можно решать различного рода задачи.

Задача №9. Сложение двоичных чисел.

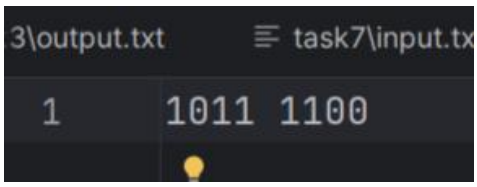
Расмотрим задачу сложения двух n-битовых двоичных целых чисел, хранящихся в n-элементных массивах A и B. Сумму этих двух чисел необходимо занести в двоичной форме в (n + 1)-элементный массив C. Напишите скрипт для сложения этих двух чисел.

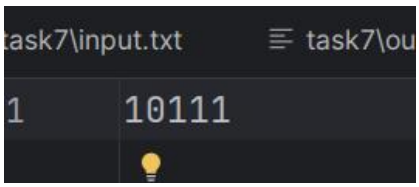
```
Def binary_addition():
    with open('input.txt', 'r') as file:
        for line in file:
            input_data = [elem for elem in line.split()]

    a = [int(elem) for elem in input_data[0]]  # первое число, каждая цифра
которого явл элементом списка a
    b = [int(elem) for elem in input_data[1]]  # второе число, каждая цифра
которого явл элементом списка b
    if not (len(a) == len(b) and 0 <= len(a) <= 10e3):  # проверка на
корректность
        with open('output.txt', 'w') as ans:
            ans.write('---Incorrect data---')
```

```
    else:
        n = len(a)
        c = [0 for I in range(n + 1)]  # заполним список с нулями
        for I in range(n, 0, -1):  # начнем заполнять с конца
            c[i] = (c[i] + a[I - 1] + b[I - 1]) % 2
            c[I - 1] += (a[I - 1] + b[I - 1]) // 2
        with open('output.txt', 'w') as ans:
            ans.write(''.join(map(str, c)))


binary_addition()
```

Код открывает файл input.txt и считывает его построчно. Каждая строка разбивается на элементы, которые затем сохраняются в списке input_data. Первое число (представлено как строка) преобразуется в список цифр a. - Второе число (также строка) преобразуется в список цифр b. Код проверяет, равны ли длины списков a и b, и находятся ли они в пределах от 0 до 10,000. Если условия не выполняются, в файл output.txt записывается строка ---Incorrect data---. Если данные корректны, создается список c, заполненный нулями, длиной на 1 больше, чем длина a или b (для учета возможного переноса). Сложение выполняется в цикле, начиная с конца списков a и b. Для каждого разряда: - Вычисляется сумма текущих разрядов и предыдущего переноса. - Результат записывается в соответствующий индекс списка c, а перенос обновляется. После завершения сложения результат (число в двоичном формате) записывается в файл output.txt.

Результат работы кода на примерах из текста задачи:





	Время выполнения	Затраты памяти
1011 1100	0.000002600019	18.76

Вывод по задаче: Были изучены, повторены: сортировка пузырьком, работа со строками и списками, алгоритмы сложения: - понимание алгоритма сложения двоичных чисел, включая обработку переноса.

Дополнительные задачи

Задача №3. 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?

```
def swap(a, b):
    return b, a
def insertion_sort_r():
    with open('input.txt', 'r') as file:
        n = -1
        for line in file:
            if n == -1:
                n = int(line)
                if not 1 <= n <= 1000:
                    with open('output.txt', 'w') as ans:  # проверка на
корректность
                        ans.write('---Incorrect data---')

        else:
            input_list = [int(elem) for elem in line.split() if
abs(int(elem)) <= 10e9]  # проверка на корректность
            for j in range(1, len(input_list)):
                key = input_list[j]
                i = j - 1
                while i >= 0 and input_list[i] < key:  # направление знака
меняется
                    input_list[i + 1], input_list[i] = swap(input_list[i +
1], input_list[i])
                    i = i - 1
            with open('output.txt', 'w') as ans:
                ans.write(' '.join(map(str, input_list)))


insertion_sort_r()
```

В коде из первой задачи меняем знак.

Да, алгоритм сортировки вставками можно написать с помощью рекурсии на Python.
Идея реализации:

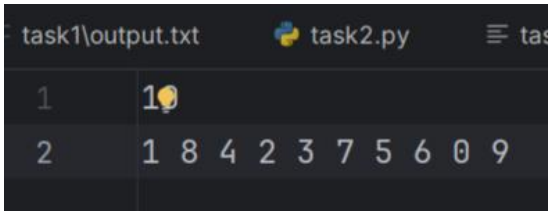
Базовый случай: если размер массива равен 1 или меньше, вернуть значение.

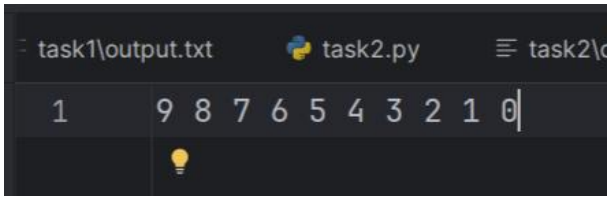
Рекурсивно отсортировать первые n-1 элементов.

Вставить последний элемент на правильную позицию в отсортированном массиве.

Но такой метод значительно уступает по времени

Результат работы кода на примерах из текста задачи:





	Время выполнения (в секундах)	Затраты памяти (в МБ)
1 0	0.000004	17.68
6 31 41 59 26 41 58	0.0000104	18.48

Вывод по задаче: Алгоритм сортировки вставками можно использовать и для сортировки по убыванию. Получен опыт самостоятельной инициализации функции swap.

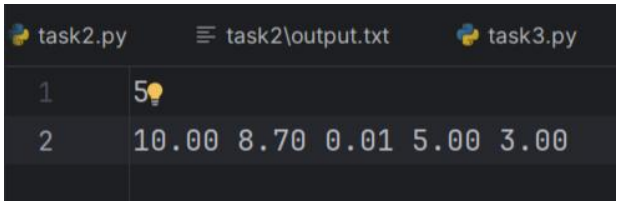
Задача №7. Знакомство с жителями Сортлэнда.

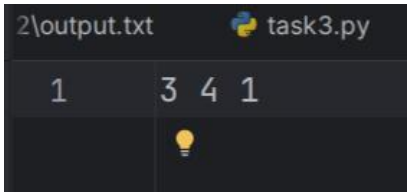
Владелец графства Сортлэнд, граф Бабблсортер, решил познакомиться со своими подданными. Число жителей в графстве нечетно и составляет n, где n может быть достаточно велико, поэтому граф решил ограничиться знакомством с тремя представителями народонаселения: с самым бедным жителем, с жителем, обладающим средним достатком, и с самым богатым жителем. Согласно традициям Сортлэнда, считается, что житель обладает средним достатком, если при сортировке жителей по сумме денежных сбережений он оказывается ровно посередине. Известно, что каждый житель графства имеет уникальный идентификационный номер, значение которого расположено в границах от единицы до n. Информация о размере денежных накоплений жителей хранится в массиве M таким образом, что сумма денежных накоплений жителя, обладающего идентификационным номером i, содержится в ячейке M[i]. Помогите секретарю графа мистеру Свопу вычислить идентификационные номера жителей, которые будут приглашены на встречу с графом.

```
Def sortland():
    with open('input.txt') as file:
        n = -1
        for line in file:
            if n == -1:
                n = int(line)  # считываем n
                if not (3 <= n <= 9999 and n % 2 != 0):  # случай
некорректных данных

                    with open('output.txt', 'w') as ans:
                        ans.write('---Incorrect data---')
                        break
            else:
                data = [float(elem) for elem in line.split() if float(elem)
<= 10e6]  # считываем список

                if len(data) != n or len(data) != len(set(data)):  # случай
некорректных данных

                    with open('output.txt', 'w') as ans:
                        ans.write('---Incorrect data---')  # случай
некорректных данных
                else:
                    id_numbers = [I + 1 for I in range(n)]
                    for I in range(len(data)):  # сортировка пузырьком
                        for j in range(I + 1, len(data)):
                            if data[j] < data[i]:
                                data[i], data[j] = data[j], data[i]
                                id_numbers[i], id_numbers[j] =
id_numbers[j], id_numbers[i]

                    with open('output.txt', 'w') as ans:
                        ans.write(str(id_numbers[0]) + ' ' +
str(id_numbers[n // 2]) + ' ' + str(id_numbers[n - 1]))


sortland()
```

Открывает файл input.txt и считывает его построчно. Первая строка должна содержать целое число n, которое обозначает количество элементов в следующем списке. Если n не соответствует условиям (нечетное число, меньше 3 или больше 9999), в файл output.txt записывается сообщение ---Incorrect data---. В последующих строках код ожидает увидеть список чисел. Для каждого числа проверяется, чтобы оно было не больше 10^6. Если длина списка не равна n или если в списке есть дубликаты, снова записывается сообщение ---Incorrect data---. Если данные корректны, код выполняет сортировку списка чисел с помощью метода пузырька. При этом также создается список id_numbers, который хранит индексы исходных элементов. После сортировки в файл output.txt записываются индексы первого элемента, среднего элемента и последнего элемента отсортированного списка.

Результат работы кода на примерах из текста задачи:





	Время выполнения(в секундах)	Затраты памяти(МБ)
5 10.00 8.70 0.01 5.00 3.00	0.0020509490966796875	16.98

Вывод по задаче: Был отсортирован список с индексами жителей по их достатку. То есть мы узнали, что можем задать любой признак, по которому будет осуществляться сортировка.

Задача №10. Палиндром.

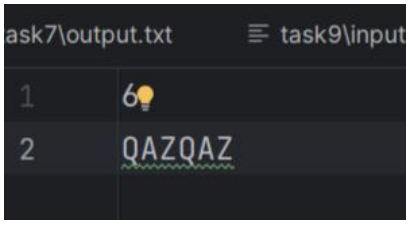
Палиндром – это строка, которая читается одинаково как справа налево, так и слева направо. На вход программы поступает набор больших латинских букв (не обязательно различных). Разрешается переставлять буквы, а также удалять некоторые буквы. Требуется из данных букв по указанным правилам составить палиндром наибольшей длины, а если таких палиндромов несколько, то выбрать первый из них в алфавитном порядке.

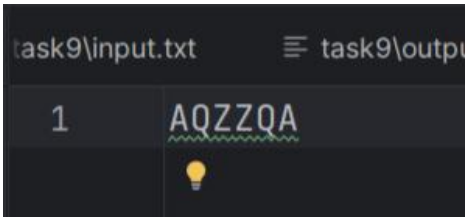
```
With open('input.txt') as file:
    n = -1
    for line in file:
        if n == -1:
            n = int(line)  # считываем n
            if not 1 <= n <= 10e5:  # случай некорректных данных

                with open('output.txt', 'w') as ans:
                    ans.write('---Incorrect data---')
        else:
            data = [elem for elem in line]  # считываем список

            if len(data) != n:

                with open('output.txt', 'w') as ans:
                    ans.write('---Incorrect data---')  # случай
некорректных данных
            else:
                for I in range(n):  # сортировка пузырьком
                    for j in range(I + 1, len(data)):
                        if data[j] < data[i]:
                            data[i], data[j] = data[j], data[i]

                mid = 'a'
```

```
                for I in range(n):
                    if data.count(data[i]) % 2 != 0 and data[i] < mid:
                        mid = data[i]
                        break
                    mid = ''  # средний элемент

                right = []  # правая часть ответа
                left = []  # левая
                for elem in data:
                    kolvo = left.count(elem) + right.count(elem)  #
количество в правой и левой части
                    if kolvo % 2 == 0:
                        if data.count(elem) - kolvo > 1:
                            left.append(elem)
                    else:
                        right.append(elem)
                with open('output.txt', 'w') as ans:
                    ans.write(''.join(left) + mid + ''.join(right[::-1]))
```

Код открывает файл input.txt и считывает его построчно. Первая строка файла должна содержать число n, которое указывает на количество элементов в следующей строке. Если n не находится в диапазоне от 1 до 100,000 (10e5), то в файл output.txt записывается строка ---Incorrect data---. Если длина следующей строки не равна n, также записывается ---Incorrect data---. Если данные корректны, производится сортировка списка data с помощью алгоритма пузырька. Далее реализуется поиск среднего элемента: - Переменная mid инициализируется как 'a'. Код ищет символ, который появляется нечетное количество раз и меньше текущего значения mid. Если такой символ найден, он становится новым средним элементом. Создаются два списка: left и right. – Для каждого элемента в отсортированном списке проверяется, сколько раз он уже встречался в обеих частях. Если количество его вхождений четное, и в левой части его меньше, чем в общем количестве, элемент добавляется в левую часть. В противном случае он добавляется в правую часть.В файл output.txt записывается результат: сначала элементы из left, затем средний элемент (если он есть), и затем элементы из right, перевернутые

Результат работы кода на примерах из текста задачи:





	Время выполнения (в секундах)	Затраты памяти (в МБ)

1 A	0.000013	19.71
6 ABCDEF	0.000018	19.71
6 QAZQAZ	0.000020	19.82

Вывод о задаче: При решение данной задачи сортировка использовалась для выбора первого ответа из палиндромов наибольшей длины в алфавитном порядке. Для построения палиндромов был создан алгоритм, сопоставляющий пары и находящий средний элемент, единственный встречающийся(необязательно) нечетное количество раз.

Вывод:

В процессе выполнения Лабораторной работы №1 были реализованы алгоритмы простых сортировок за n^2, проверены умение работать с файлами: открывать их для чтения и записи, а также обрабатывать данные, содержащиеся в них, понимание логики условий и ветвлений в коде, что позволяет принимать решения на основе состояния данных, важность проверки входных данных на соответствие заданным условиям.