Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №5**

**«Исследование алгоритмов сортировки»**

**ПО ДИСЦИПЛИНЕ «Основы алгоритмизации и программирование»**

Выполнил: студент учебной группы

ИСПк-203-52-00

Яковлев А. С.

Преподаватель:

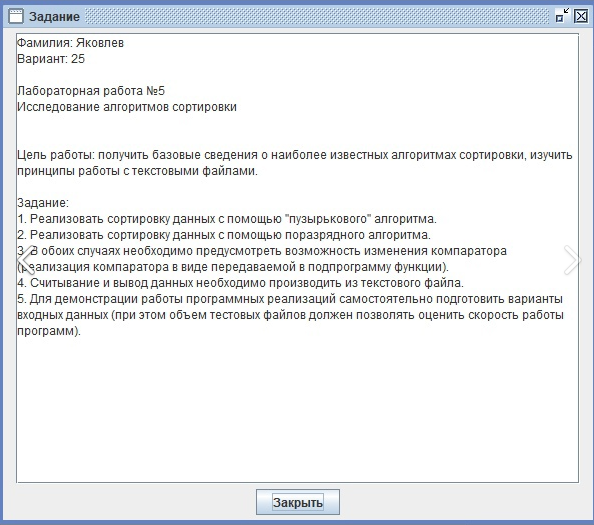
Сергеева Елизавета Григорьевна

Киров

2024

**Цель:** получить базовые сведения о наиболее известных алгоритмах сортировки, изучить принципы работы с текстовыми файлами.

**Формулировка задания (Вариант 25):**



**Описание алгоритма:**

1. Пузырьковая сортировка — это простой алгоритм сортировки, который многократно проходит по массиву, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке. Этот процесс повторяется до тех пор, пока массив не будет отсортирован:

• Начало с первого элемента массива.

• Сравнивает его со следующим элементом.

• Если первый элемент больше второго, меняем их местами.

• Переходит к следующему элементу и повторяем процесс.

• После каждой операции наибольший элемент "всплывает" в конец массива.

• Если за проход по массиву не было произведено ни одной замены, значит массив уже отсортирован, и алгоритм завершает свою работу.

2. Поразрядная сортировка — это алгоритм сортировки, который сортирует числа по отдельным разрядам. Он использует вспомогательный алгоритм подсчета (Counting Sort) для сортировки чисел по каждому разряду.

Алгоритм:

• Находит максимальное значение в массиве, чтобы определить количество разрядов.

• Для каждого разряда (начиная с младшего) выполняет следующие шаги:

• Используем подсчет для определения количества вхождений каждой цифры в текущем разряде.

• Преобразует массив подсчета так, чтобы он содержал позиции для каждого элемента.

• Создает выходной массив, в который помещаем элементы на основе позиций из массива подсчета.

• Копирует выходной массив обратно в исходный массив.

• Повторяет процесс для всех разрядов, пока не отсортирует все числа.

3. Процедура ReadDataFromFile, которая считывает целые числа из текстового файла и заполняет массив.

4. Сортировка данных:

• Сначала выполняется пузырьковая сортировка с использованием процедуры BubbleSort. Результаты записываются в файл.

• Затем выполняется поразрядная сортировка с использованием процедуры RadixSort. Результаты также записываются в файл.

Время выполнения каждой сортировки фиксируется и выводится на экран.

5. Отсортированные данные сохраняются в отдельные файлы для дальнейшего анализа.

**Схема алгоритма**

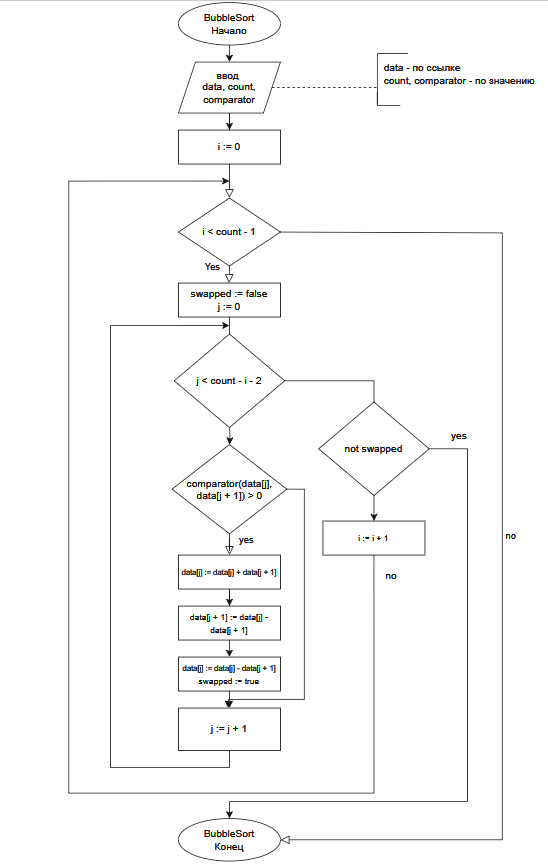


Рисунок 1 – схема алгоритма сортировки «пузырьковым» методом

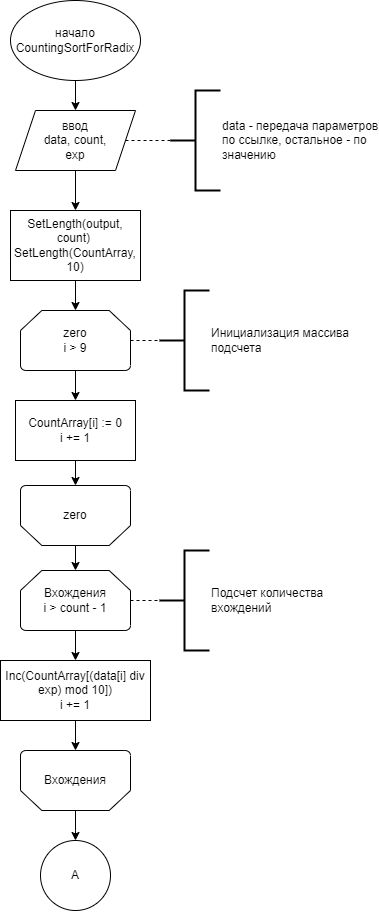


Рисунок 2 – схема алгоритма поразрядной сортировки (1)

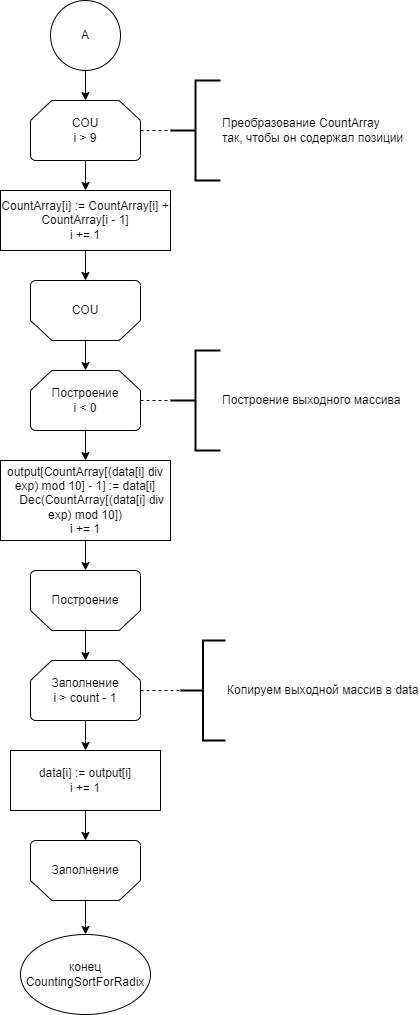


Рисунок 3 – схема алгоритма поразрядной сортировки (2)

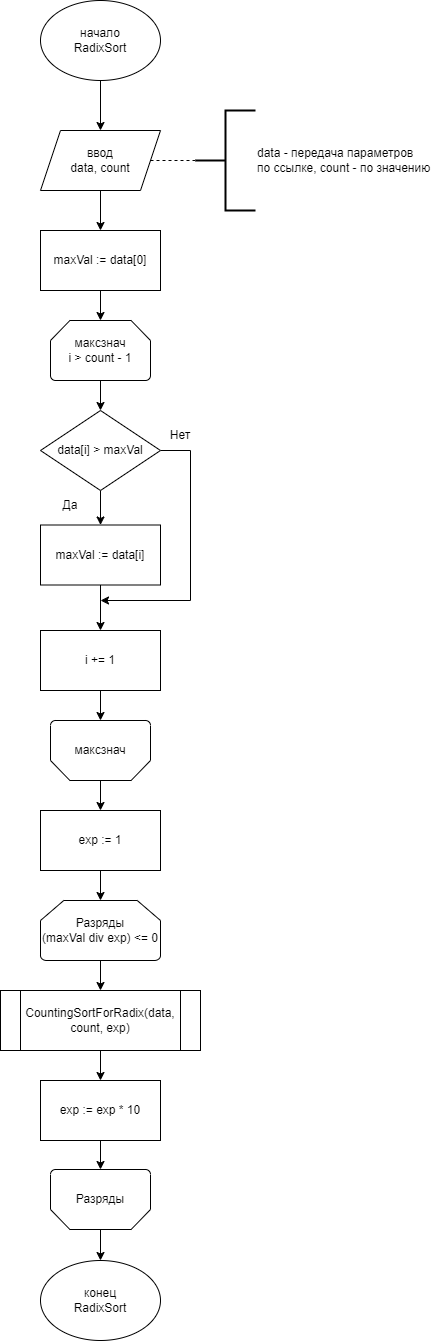


Рисунок 4 – схема алгоритма поразрядной сортировки (3)

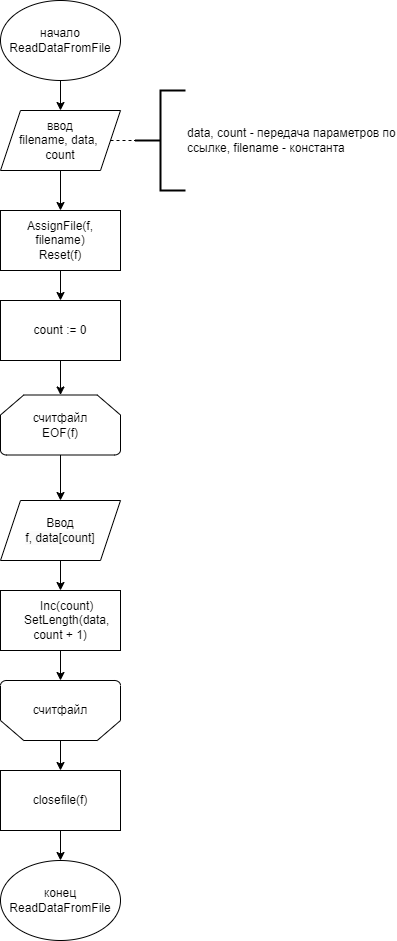


Рисунок 5 – схема алгоритма процедуры считывания с файл

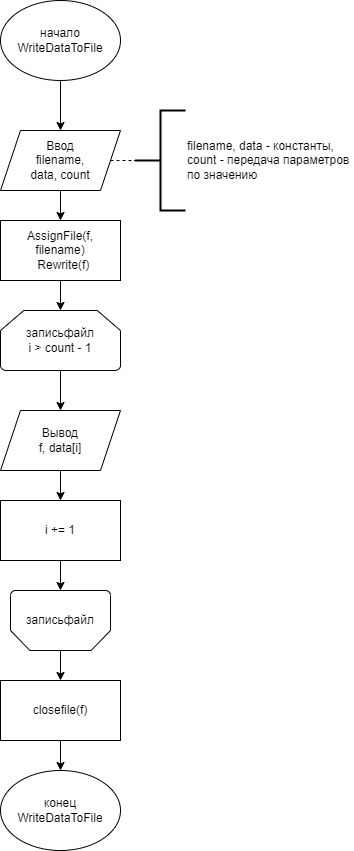


Рисунок 6 – схема алгоритма записи отсортированного массива в файл

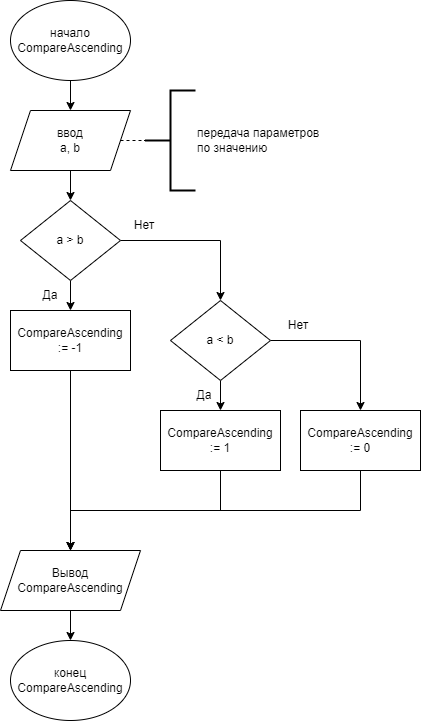


Рисунок 7 – схема алгоритма компаратора

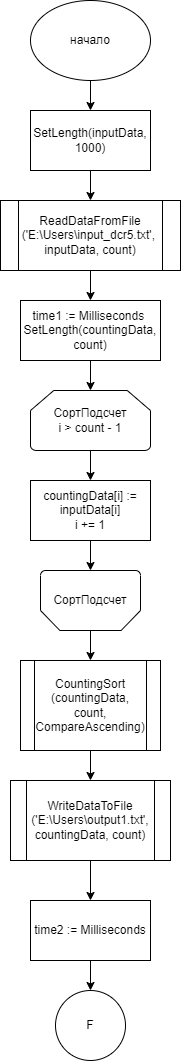


Рисунок 8 – схема алгоритма основной программы (1)

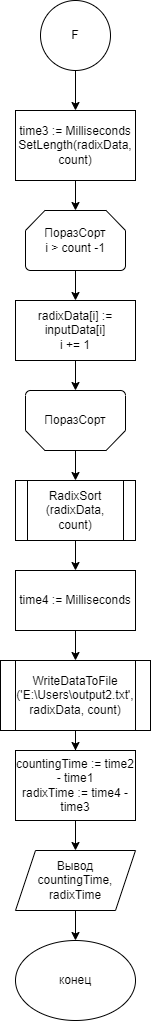


Рисунок 9 – схема алгоритма основной программы (2)

**Код программы:**

**procedure** BubbleSort(**var** data: **array of** integer; count: integer; comparator: **function**(a, b: integer): integer);

**var**

i, j: integer;

swapped: boolean;

**begin**

**for** i := 0 **to** count - 1 **do**

**begin**

swapped := false;

**for** j := 0 **to** count - i - 2 **do**

**begin**

**if** comparator(data[j], data[j + 1]) > 0 **then**

**begin**

// замена местами

data[j] := data[j] + data[j + 1];

data[j + 1] := data[j] - data[j + 1];

data[j] := data[j] - data[j + 1];

swapped := true;

**end**;

**end**;

// если не было обменов, массив уже отсортирован

**if not** swapped **then**

**Break**;

**end**;

**end**;

**procedure** CountingSortForRadix(**var** data: **array of** integer; count: integer; exp: integer);

**var**

output: **array of** integer;

CountArray: **array of** integer;

i: integer;

**begin**

SetLength(output, count);

SetLength(CountArray, 10); // для цифр от 0 до 9

// инициализация массива подсчета

**for** i := 0 **to** 9 **do**

CountArray[i] := 0;

// подсчет количества вхождений

**for** i := 0 **to** count - 1 **do**

Inc(CountArray[(data[i] **div** exp) **mod** 10]);

// преобразование CountArray так, чтобы он содержал позиции

**for** i := 1 **to** 9 **do**

CountArray[i] := CountArray[i] + CountArray[i - 1];

// построение выходного массива

**for** i := count - 1 **downto** 0 **do**

**begin**

output[CountArray[(data[i] **div** exp) **mod** 10] - 1] := data[i];

Dec(CountArray[(data[i] **div** exp) **mod** 10]);

**end**;

// копирование выходного массив в data

**for** i := 0 **to** count - 1 **do**

data[i] := output[i];

**end**;

**procedure** RadixSort(**var** data: **array of** integer; count: integer);

**var**

maxVal, exp: integer;

**begin**

maxVal:= data[0];

**for var** i:= 1 **to** count - 1 **do**

**if** data[i] > maxVal **then**

maxVal:= data[i];

// применяем CountingSort для каждого разряда

exp := 1;

**while** (maxVal **div** exp) > 0 **do**

**begin**

CountingSortForRadix(data, count, exp);

exp := exp \* 10;

**end**;

**end**;

**procedure** ReadDataFromFile(**const** filename: string; **var** data: **array of** Integer; **var** count: Integer);

**var**

f: Text;

**begin**

AssignFile(f, filename);

Reset(f);

count := 0;

**while not** Eof(f) **do**

**begin**

Read(f, data[count]);

Inc(count);

SetLength(data, count + 1); // увеличиваем размер массива для следующего числа

**end**;

CloseFile(f);

**end**;

**procedure** WriteDataToFile(**const** filename: string; **const** data: **array of** integer; count: integer);

**var**

f: Text;

**begin**

AssignFile(f, filename);

Rewrite(f);

**for var** i := 0 **to** count - 1 **do**

WriteLn(f, data[i]);

CloseFile(f);

**end**;

**function** CompareAscending(a, b: integer): integer;

**begin**

**if** a < b **then**

Result := -1

**else if** a > b **then**

Result := 1

**else**

Result := 0;

**end**;

**var**

inputData: **array of** integer;

bubbleData: **array of** integer;

radixData: **array of** integer;

count, i, time1, time2, time3, time4, countingTime, radixTime: integer;

**begin**

SetLength(inputData, 1000);

ReadDataFromFile('C:\5dkr\input.txt', inputData, count);

// пузырьковая сортировка

time1 := Milliseconds;

SetLength(bubbleData, count);

**for** i := 0 **to** count - 1 **do**

bubbleData[i] := inputData[i];

BubbleSort(bubbleData, count, CompareAscending);

WriteDataToFile('C:\5dkr\output1.txt', bubbleData, count);

time2 := Milliseconds;

// поразрядная сортировка

time3 := Milliseconds;

SetLength(radixData, count);

**for** i := 0 **to** count - 1 **do**

radixData[i] := inputData[i];

RadixSort(radixData, count);

time4 := Milliseconds;

WriteDataToFile('C:\5dkr\output2.txt', radixData, count);

countingTime := time2 - time1;

radixTime := time4 - time3;

writeLn('для сортировки данных пузырковым методом потребовалось: ', countingTime, ' миллисекунд. для поразрядной сортировки: ', radixTime, '.');

writeLn('результат можно посмотреть в соответствующих файлах.');

**end**.

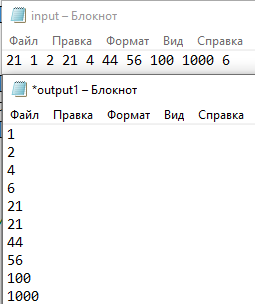


Рисунок 11 – результат выполнения программы для поразрядной сортировки

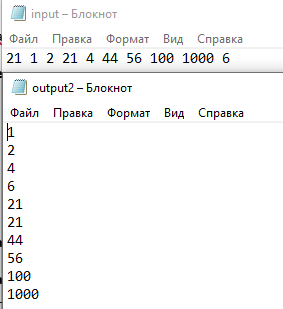


Рисунок 12 – результат выполнения программы для сортировки «пузырьковым» методом

**Вывод:**

Мы изучили основные методы сортировки, такие как сортировка подсчетом и поразрядная сортировка, и успешно реализовали их в программе. Также мы узнали о компараторах и их назначении. Мы научились измерять время выполнения кода для сравнения эффективности различных сортировок. Кроме того, мы освежили знания о работе с файлами в Паскале и применили их при создании программы. В результате была разработана программа, которая считывает массив из файла и сохраняет отсортированные массивы, полученные с помощью разных методов сортировки, в отдельные файлы.