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 ПРОГРАММИРОВАНИЯ»**

Выполнила: студентка учебной группы

ИСПк-202-52-00

Широнина Анна Андреевна

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

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

1. **Формулировка заданий.**

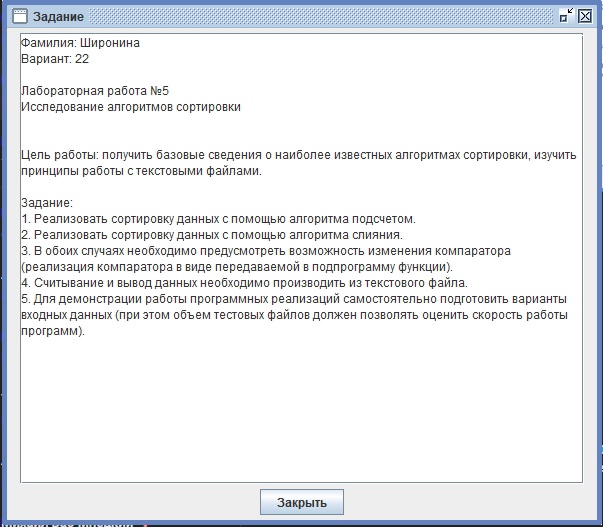


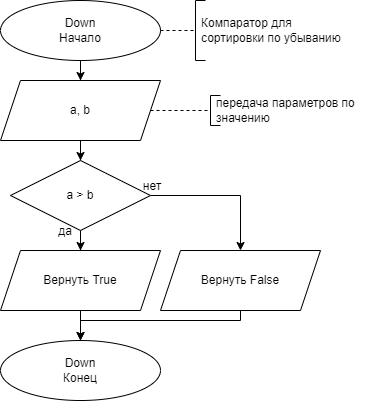
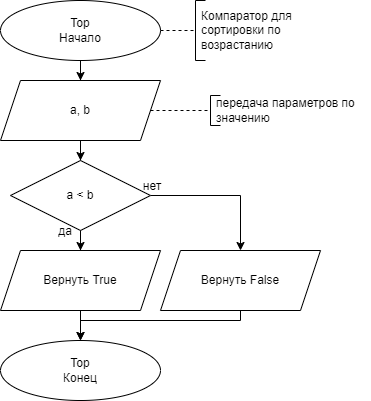
Рисунок 1 – Постановка задачки

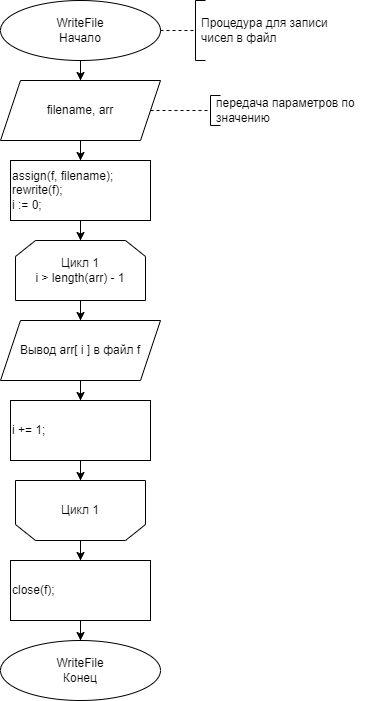
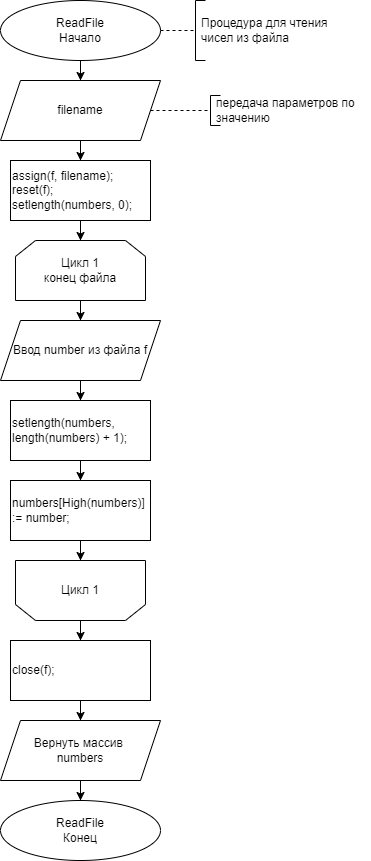
1. **Описание алгоритма**

2.1 Сортировка подсчетом. Находим максимальное значение в массиве для определения размера вспомогательного массива. Создаётся вспомогательный массив и на старте заполняется нулями. Проходится по всему исходному массиву и смотрится очередное значение в ячейке. Содержимое этой ячейки берётся и увеличивается на единицу значение вспомогательного массива под этим номером. После цикла во вспомогательном массиве хранятся данные, сколько раз встречается каждый элемент. Теперь проходится по вспомогательному массиву, и если в очередной ячейке лежит что-то больше нуля, то в исходный массив столько же раз отправляется номер этой ячейки.

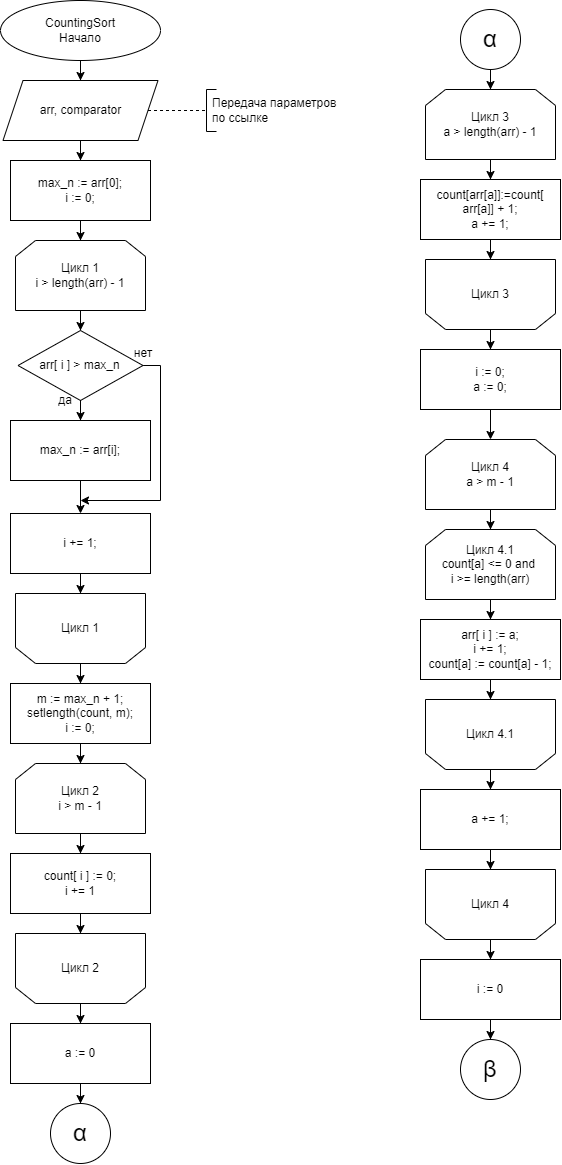
2.2 Сортировка слиянием. Сортируемый массив разбивается на две части примерно одинакового размера. Каждая из получившихся частей сортируется отдельно, например— тем же самым алгоритмом. Два упорядоченных массива половинного размера соединяются в один.  Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным). Соединение двух упорядоченных массивов в один. На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1. Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.

1. **Схема алгоритма. Общие подпрограммы**

****

****

* 1. Сортировка с помощью алгоритма подсчета

****

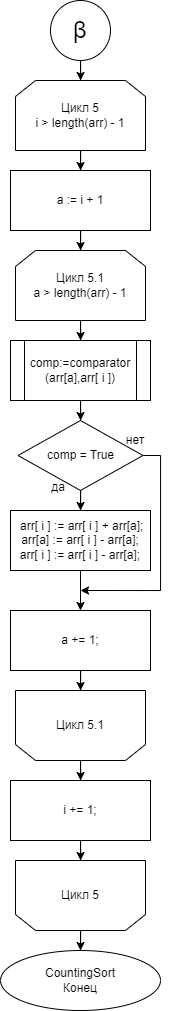
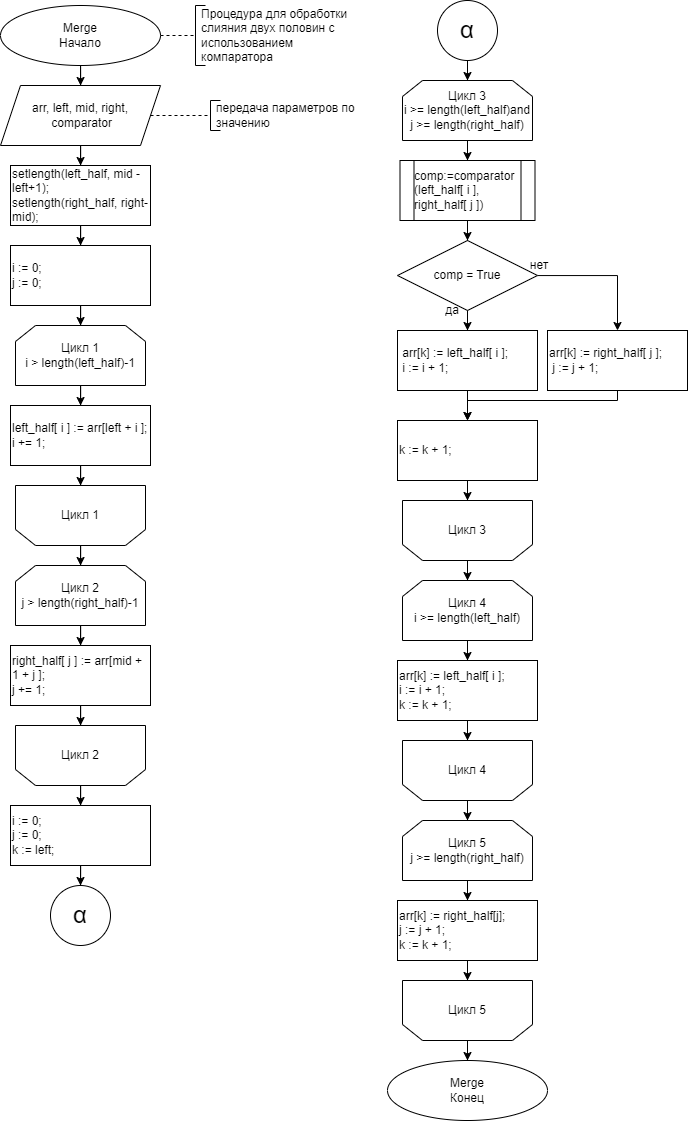


Рисунок 2 – Схема алгоритма задачи №1

3.2 Сортировка с помощью алгоритма слияния

****

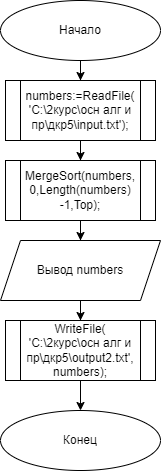
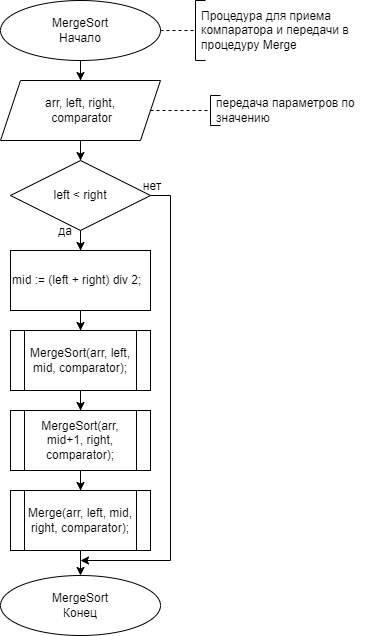


Рисунок 3 – Схема алгоритма задачи №2

1. **Код программы**
   1. **Сортировка с помощью алгоритма подсчета**

**type**

TComparator = **function**(a, b: integer): boolean;

// Компаратор для сортировки по возрастанию

**function** Top(a, b: Integer): boolean;

**begin**

Top := a < b;

**end**;

// Компаратор для сортировки по убыванию

**function** Down(a, b: Integer): boolean;

**begin**

Down := a > b;

**end**;

**procedure** CountingSort(**var** arr: **array of** integer; comparator: TComparator);

**var**

max\_n, m, i, a: integer;

count: **array of** integer;

**begin**

// Находим максимальное значение в массиве

max\_n := arr[0];

**for** i := 1 **to** length(arr) - 1 **do**

**if** arr[i] > max\_n **then**

max\_n := arr[i];

m := max\_n + 1; // Задаем размер массива count на единицу больше максимального значения

// Инициализируем массив count с нулями, размером m

setlength(count, m);

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

count[i] := 0;

// Подсчитываем количество каждого элемента в исходном массиве

**for** a := 0 **to** length(arr) - 1 **do**

count[arr[a]] := count[arr[a]] + 1; // Увеличиваем счетчик для значения arr[a]

// Переписываем исходный массив с учетом количества каждого элемента

i := 0;

// Заполняем массив, применяя компаратор

**for** a := 0 **to** m - 1 **do** // Проходим по значениям от 0 до max\_val

**begin**

**while** (count[a] > 0) **and** (i < length(arr)) **do**

**begin**

arr[i] := a;

i += 1; // Переходим к следующей позиции в исходном массиве

count[a] := count[a] - 1; // Уменьшаем счетчик

**end**;

**end**;

// Дополнительный шаг для изменения порядка в зависимости от компаратора

**for** i := 0 **to** length(arr) - 1 **do**

**for** a := i + 1 **to** length(arr) - 1 **do**

**if** comparator(arr[a], arr[i]) **then**

**begin**

// Меняем местами элементы

arr[i] := arr[i] + arr[a];

arr[a] := arr[i] - arr[a];

arr[i] := arr[i] - arr[a];

**end**;

**end**;

// Процедура для чтения чисел из файла

**function** ReadFile(filename: string): **array of** integer;

**var**

f: text;

number: integer;

numbers: **array of** integer;

**begin**

assign(f, filename);

reset(f);

setlength(numbers, 0);

**while not** eof(f) **do**

**begin**

readln(f, number);

setlength(numbers, length(numbers) + 1);

numbers[High(numbers)] := number;

**end**;

close(f);

ReadFile := numbers;

**end**;

// Процедура для записи чисел в файл

**procedure** WriteFile(filename: string; arr: **array of** Integer);

**var**

f: text;

i: integer;

**begin**

assign(f, filename);

rewrite(f);

**for** i := 0 **to** length(arr) - 1 **do**

**begin**

writeln(f, arr[i]);

**end**;

close(f);

**end**;

**var**

numbers: **array of** integer;

i: integer;

**begin**

// Читаем числа из файла

numbers := ReadFile('C:\2курс\осн алг и пр\дкр5\input.txt');

// Сортируем массив по возрастанию

CountingSort(numbers, Top);

// Выводим отсортированный массив

**for** i := 0 **to** length(numbers) - 1 **do**

write(numbers[i], ' ');

// Записываем отсортированный массив в файл

WriteFile('C:\2курс\осн алг и пр\дкр5\output1.txt', numbers);

**end**.

* 1. **Сортировка с помощью алгоритма слияния**

// Тип TComparator: определен тип для компараторов функции, который принимает два целых числа и возвращает логическое значение.

**type**

TComparator = **function**(a, b: integer): boolean;

// Процедура Merge: обработает слияние двух половин с использованием компаратора, переданного как параметр.

**procedure** Merge(arr: **array of** Integer; left, mid, right: integer; comparator: TComparator);

**var**

i, j, k: integer;

left\_half, right\_half: **array of** integer;

**begin**

// Создаем левые и правые массивы

setlength(left\_half, mid - left + 1);

setlength(right\_half, right - mid);

// Копируем данные в левые и правые массивы

**for** i := 0 **to** length(left\_half) - 1 **do**

left\_half[i] := arr[left + i];

**for** j := 0 **to** length(right\_half) - 1 **do**

right\_half[j] := arr[mid + 1 + j];

// Слияние отсортированных половин обратно в основной массив

i := 0; // Индекс для левой половины

j := 0; // Индекс для правой половины

k := left; // Индекс для основного массива

**while** (i < length(left\_half)) **and** (j < length(right\_half)) **do**

**begin**

**if** comparator(left\_half[i], right\_half[j]) **then**

**begin**

arr[k] := left\_half[i];

i := i + 1;

**end**

**else**

**begin**

arr[k] := right\_half[j];

j := j + 1;

**end**;

k := k + 1;

**end**;

// Добавляем оставшиеся элементы из левой половины, если такие остались

**while** i < length(left\_half) **do**

**begin**

arr[k] := left\_half[i];

i := i + 1;

k := k + 1;

**end**;

// Добавляем оставшиеся элементы из правой половины, если такие остались

**while** j < length(right\_half) **do**

**begin**

arr[k] := right\_half[j];

j := j + 1;

k := k + 1;

**end**;

**end**;

// Процедура MergeSort: прием компаратора и передача его в процедуру Merge

**procedure** MergeSort(**var** arr: **array of** integer; left, right: integer; comparator: TComparator);

**var**

mid: integer;

**begin**

// Если массив содержит более одного элемента, продолжаем деление

**if** left < right **then**

**begin**

// Определяем середину массива

mid := (left + right) **div** 2;

// Рекурсивно вызываем MergeSort для каждой половины

MergeSort(arr, left, mid, comparator);

MergeSort(arr, mid + 1, right, comparator);

// Сливаем отсортированные половины

Merge(arr, left, mid, right, comparator);

**end**;

**end**;

// Компаратор для сортировки по возрастанию

**function** Top(a, b: Integer): boolean;

**begin**

Top := a < b;

**end**;

// Компаратор для сортировки по убыванию

**function** Down(a, b: Integer): boolean;

**begin**

Down := a > b;

**end**;

// Процедура для чтения чисел из файла

**function** ReadFile(filename: string): **array of** integer;

**var**

f: text;

number: integer;

numbers: **array of** integer;

**begin**

assign(f, filename);

reset(f);

setlength(numbers, 0);

**while not** eof(f) **do**

**begin**

readln(f, number);

setlength(numbers, length(numbers) + 1);

numbers[High(numbers)] := number;

**end**;

close(f);

ReadFile := numbers;

**end**;

// Процедура для записи чисел в файл

**procedure** WriteFile(filename: string; arr: **array of** Integer);

**var**

f: text;

i: integer;

**begin**

assign(f, filename);

rewrite(f);

**for** i := 0 **to** length(arr) - 1 **do**

**begin**

writeln(f, arr[i]);

**end**;

close(f);

**end**;

**var**

numbers: **array of** integer;

i: integer;

**begin**

// Читаем числа из файла

numbers := ReadFile('C:\2курс\осн алг и пр\дкр5\input1.txt');

// Сортируем массив по возрастанию

MergeSort(numbers, 0, Length(numbers) - 1, Top);

// Выводим отсортированный массив

**for** i := 0 **to** length(numbers) - 1 **do**

write(numbers[i], ' ');

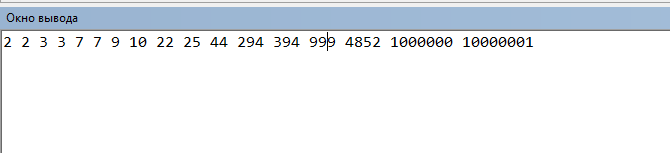
// Записываем отсортированный массив в файл

WriteFile('C:\2курс\осн алг и пр\дкр5\output2.txt', numbers);

**end**.

1. **Результат выполнения программы**

5.1 Сортировка с помощью алгоритма подсчета

****

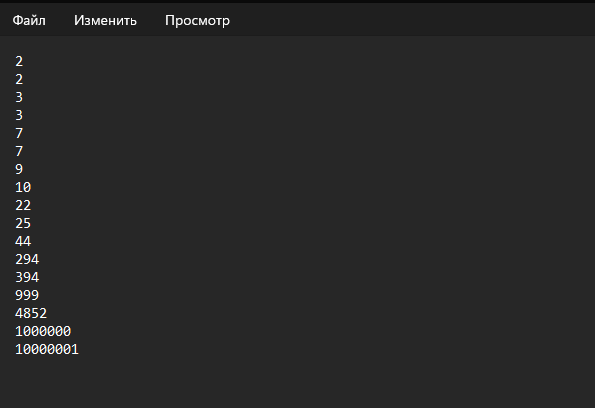
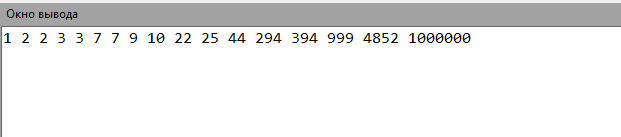
****

Рисунок 4 – Результат выполнения программы по задаче №1

5.2 Сортировка с помощью алгоритма слияния

****

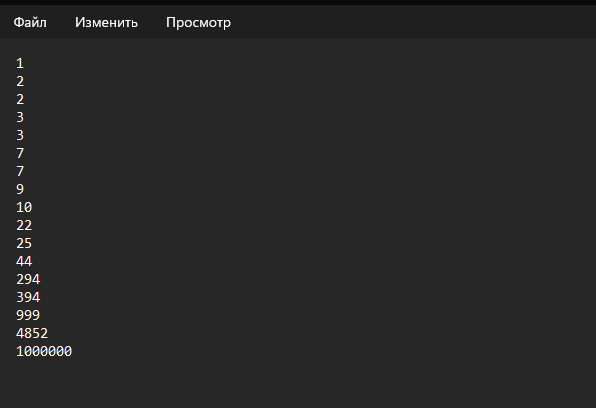
****

Рисунок 5 – Результат выполнения программы по задаче №2

1. **Вывод**

В ходе выполнения работы была поставлена цель получить базовые сведения о наиболее известных алгоритмах сортировки и изучить принципы работы с текстовыми файлами. Мы рассмотрели два различных алгоритма сортировки данных: сортировка подсчетом и сортировка слиянием.

Кроме того, мы изучили работу с текстовыми файлами на языке программирования Pascal. Овладение навыками чтения и записи данных в текстовые файлы является важной частью работы с информацией, позволяющей сохранять результаты обработки данных для последующего анализа. Мы рассмотрели, как открывать файлы, читать из них данные, а также записывать результаты обратно в файл.

Для работы над схемами алгоритма необходимо четко сформулировать проблему, которую необходимо решить, собрать входные данные, необходимые для алгоритма, описать последовательность шагов, которые необходимо выполнить для решения задачи, использовать стандартные символы для построения схемы алгоритма (овалы – начало и конец, прямоугольники – действия или процессы, ромбы – условия или проверки и т.д.), проверить алгоритм на разных входных данных для выявления возможных ошибок, проанализировать алгоритм и внести изменения для повышения его эффективности, записать и оформить полученные результаты. Работа над схемами алгоритма помогает визуализировать процесс и обеспечить более легкое понимание и оптимизацию решения задачи.

Таким образом, работа помогла углубить понимание алгоритмов сортировки и принципов работы с текстовыми файлами, что является полезным знанием для дальнейшей работы в области программирования и обработки данных.