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»**

Выполнил: студент учебной группы

ИСПк-202-52-00

Подпорин Антон Александрович

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

1. **Цель работы** - получить базовые сведения о наиболее известных алгоритмах сортировки, изучить принципы работы с текстовыми файлами.
2. **Формулировка задания:** Вариант 17.

**Задание:**

1. Реализовать сортировку данных с помощью вставок.

2. Реализовать сортировку данных с помощью алгоритма слияния.

3. В обоих случаях необходимо предусмотреть возможность изменения компаратора

(реализация компаратора в виде передаваемой в подпрограмму функции).

4. Считывание и вывод данных необходимо производить из текстового файла.

5. Для демонстрации работы программных реализаций самостоятельно подготовить варианты входных данных (при этом объем тестовых файлов должен позволять оценить скорость работы программ).

**3. Описание алгоритма**

**Процедура InsertSort:**

Это реализация сортировки вставкой. Принимает массив целых чисел и функцию-компаратор. Использует два вложенных цикла для сравнения и обмена соседних элементов, если они находятся в неправильном порядке. Внешний цикл проходит по всем элементам массива, а внутренний цикл сравнивает пары элементов и производит обмен, если это необходимо.

**Процедура MergeSort:**

Это реализация быстрой сортировки. Принимает массив целых чисел, границы сортируемого подмассива (low и high) и функцию-компаратор. Выбирает опорный элемент (pivot) и разделяет массив на элементы, меньшие и большие опорного элемента. Рекурсивно сортирует подмассивы, пока не будет достигнута базовая ситуация (когда low >= high).

**Функция CompareAsc:**

Определяет порядок сортировки по возрастанию. Возвращает разность двух элементов: если первый меньше второго, возвращает отрицательное значение; если равны — 0; если больше — положительное значение.

**Процедура RDF:**

Читает данные из текстового файла. Считывает целые числа до конца файла и сохраняет их в массив data. Увеличивает счетчик count, чтобы отслеживать количество считанных элементов.

**Процедура WDF:**

Записывает отсортированные данные в текстовый файл. Проходит по массиву и записывает каждый элемент в файл.

**Основная программа:**

Создает массив inputData для хранения данных. Читает данные из файла data.txt. Копирует данные в массив для сортировки вставками и выполняет сортировку. Записывает отсортированные данные в файл datainsert.txt. Копирует данные в массив для сортировки слияниями и выполняет сортировку. Записывает отсортированные данные в файл datamerge.txt. Измеряет время выполнения двух и выводит результаты на экран.

**Время:**

Время выполнения каждой сортировки измеряется с помощью переменных time1, time2, time3, time4, которые фиксируют время до и после выполнения каждой процедуры сортировки. Разница между этими временными метками - время, затраченное на каждую сортировку.

**Вывод результатов:**

В конце программы выводится информация о том, сколько времени потребовалось на каждую из сортировок, что позволяет сравнить скорость выполнения программ.

1. **Схема алгоритма с комментариями**

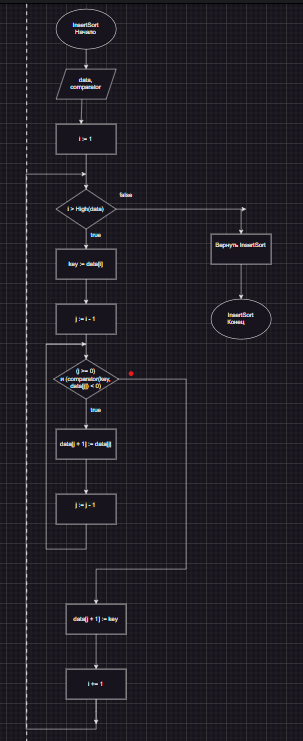


Рисунок 1 – Схема алгоритма процедуры InsertSort

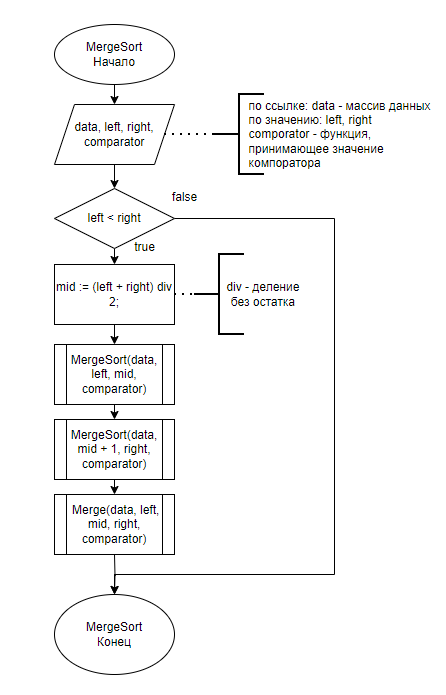


Рисунок 2 – Схема алгоритма процедуры MergeSort

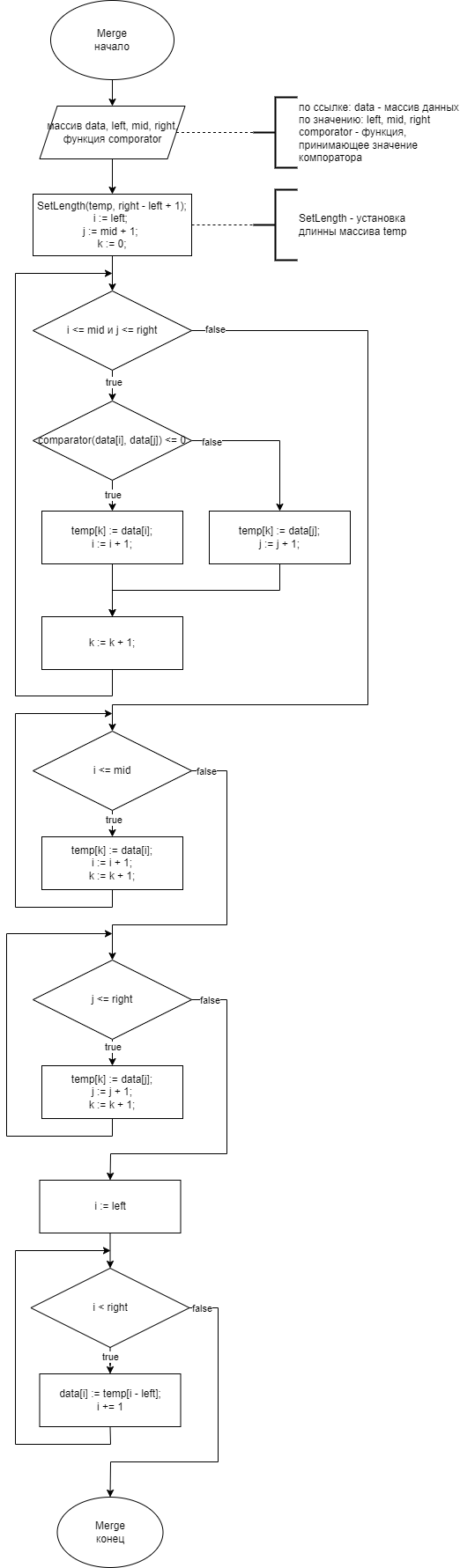


Рисунок 3 – Схема алгоритма процедуры Merge

Изображение выглядит как текст, диаграмма, дизайн

Автоматически созданное описание

Рисунок 4 – Схема алгоритма функции CompareAsc

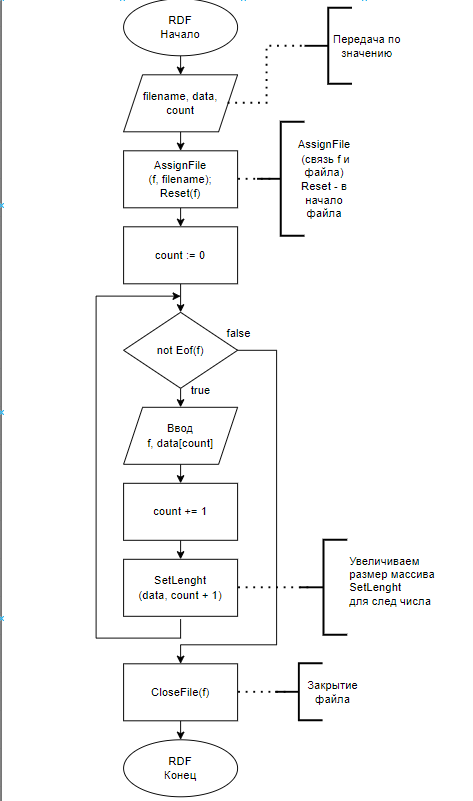


Рисунок 5 – Схема алгоритма процедуры RDF

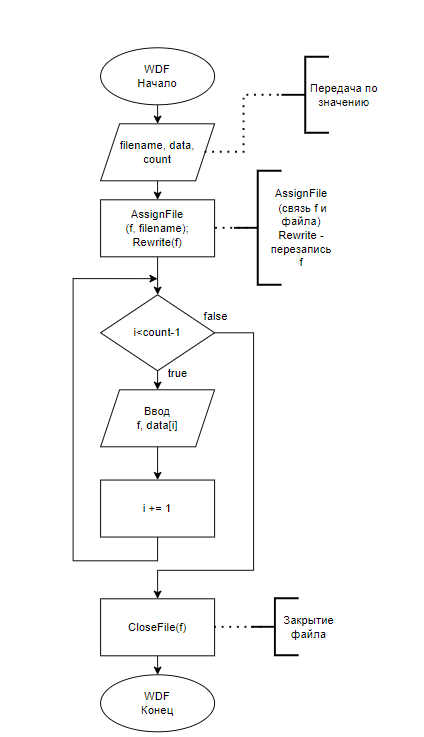


Рисунок 6 – Схема алгоритма процедуры WDF

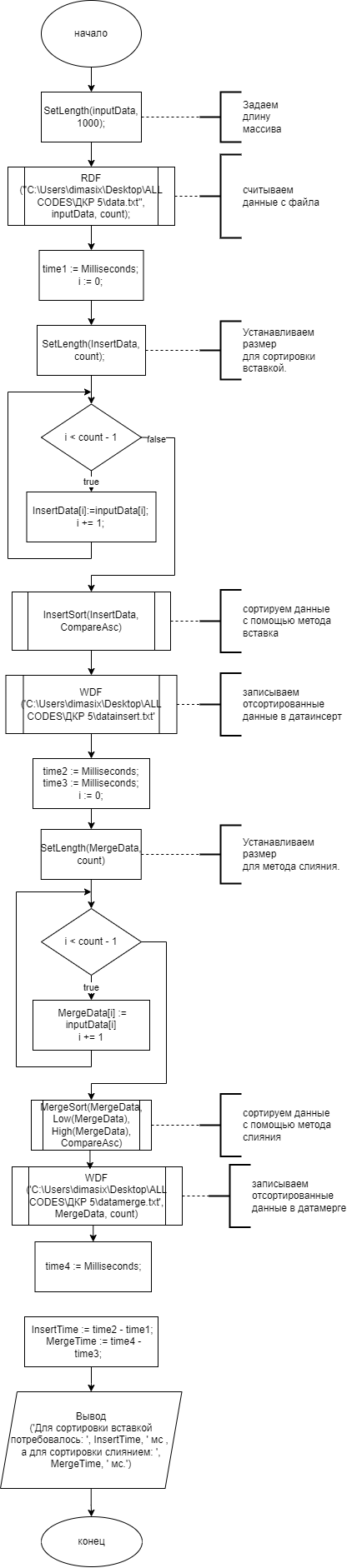




Рисунок 7 – Схема алгоритма основной программы

1. **Код программы**

**procedure** InsertSort(**var** data: **array of** Integer; comparator: **function**(a, b: Integer): Integer);

**var**

i, j, key: Integer;

**begin**

**for** i := 1 **to** High(data) **do**

**begin**

key := data[i];

j := i - 1;

**while** (j >= 0) **and** (comparator(key, data[j]) < 0) **do**

**begin**

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

j := j - 1;

**end**;

data[j + 1] := key;

**end**;

**end**;

**procedure** Merge(**var** data: **array of** Integer; left, mid, right: Integer; comparator: **function**(a, b: Integer): Integer);

**var**

i, j, k: Integer;

temp: **array of** Integer;

**begin**

SetLength(temp, right - left + 1);

i := left;

j := mid + 1;

k := 0;

**while** (i <= mid) **and** (j <= right) **do**

**begin**

**if** comparator(data[i], data[j]) <= 0 **then**

**begin**

temp[k] := data[i];

i := i + 1;

**end**

**else**

**begin**

temp[k] := data[j];

j := j + 1;

**end**;

k := k + 1;

**end**;

**while** i <= mid **do**

**begin**

temp[k] := data[i];

i := i + 1;

k := k + 1;

**end**;

**while** j <= right **do**

**begin**

temp[k] := data[j];

j := j + 1;

k := k + 1;

**end**;

**for** i := left **to** right **do**

data[i] := temp[i - left];

**end**;

**procedure** MergeSort(**var** data: **array of** Integer; left, right: Integer; comparator: **function**(a, b: Integer): Integer);

**var**

mid: Integer;

**begin**

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

**begin**

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

MergeSort(data, left, mid, comparator);

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

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

**end**;

**end**;

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

**begin**

Result := a - b; // сравнение по возрастанию

**end**;

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

**var**

f: TextFile;

num: integer;

**begin**

AssignFile(f, filename);

Reset(f);

count := 0;

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

**begin**

Read(f, num);

data[count] := num;

Inc(count);

**end**;

CloseFile(f);

**end**;

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

**var**

f: TextFile;

i: integer;

**begin**

AssignFile(f, filename);

Rewrite(f);

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

WriteLn(f, data[i]);

CloseFile(f);

**end**;

**var**

inputData: **array of** integer;

InsertData: **array of** integer;

MergeData: **array of** integer;

count, i, time1, time2, time3, time4, InsertTime, MergeTime: integer;

**begin**

SetLength(inputData, 1000);

RDF('C:\Users\dimasix\Desktop\ALL CODES\ДКР 5\data.txt', inputData, count);

// Сортировка вставкой

time1 := Milliseconds; // Milliseconds возвращает текущее время в миллисекундах

SetLength(InsertData, count); // размер для вставочной сортировки

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

InsertData[i] := inputData[i];

InsertSort(InsertData, CompareAsc);

WDF('C:\Users\dimasix\Desktop\ALL CODES\ДКР 5\datainsert.txt', InsertData, count);

time2 := Milliseconds;

// Сортировка слиянием

time3 := Milliseconds;

SetLength(MergeData, count); // размер для сортировки слиянием

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

MergeData[i] := inputData[i];

MergeSort(MergeData, Low(MergeData), High(MergeData), CompareAsc);

WDF('C:\Users\dimasix\Desktop\ALL CODES\ДКР 5\datamerge.txt', MergeData, count);

time4 := Milliseconds;

InsertTime := time2 - time1;

MergeTime := time4 - time3;

WriteLn('Для сортировки вставкой потребовалось: ', InsertTime, ' мс , а для сортировки слиянием: ', MergeTime, ' мс.');

**end**.

1. **Результат выполнения программы**

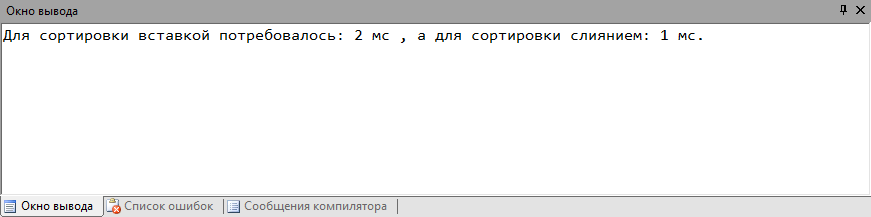


Рисунок 7 – Результат выполнения программы

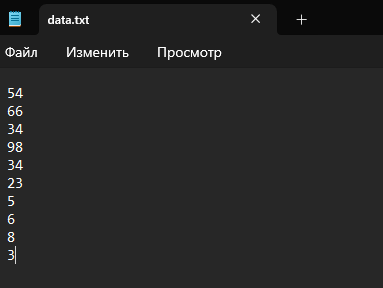


Рисунок 8 – Результат выполнения программы

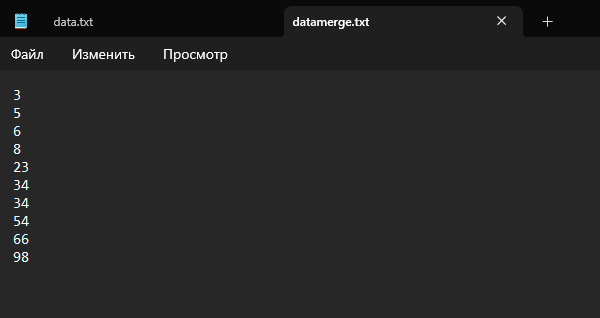


Рисунок 9 – Результат выполнения программы (слияние)

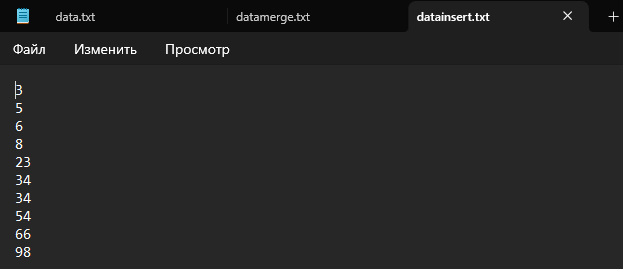


Рисунок 10 – Результат выполнения программы (вставка)

1. **Вывод:**

В процессе изучения алгоритмов сортировки и работы с текстовыми файлами автор изучил основы которые имеют большое значение в программировании и разработке программного обеспечения.

1. Сортировка вставками:

алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов.

2. Сортировка слиянием:

алгоритм сортировки, который упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получать только последовательно, например — потоки) в определённом порядке.

Оба алгоритма имеют свои сильные и слабые стороны.

Работа с текстовыми файлами в программировании включает следующие основные принципы:

Открытие файла: перед тем, как взаимодействовать с файлом, его необходимо открыть. В зависимости от нужного режима (чтение, запись, добавление) мы загружаем файл в программу.

Чтение и запись данных: программы могут считывать текстовые данные по строкам или целиком, а также записывать в файл данные, формируя нужный формат. При чтении данных важно обрабатывать возможные ошибки, такие как отсутствие файла.

Закрытие файла: после завершения всех операций с файлом важно его закрыть. Это освобождает ресурсы системы и гарантирует сохранение внесенных изменений.

Изучение алгоритмов сортировки, таких как пузырьковая и быстрая сортировка, а также принципов работы с текстовыми файлами, значительно расширяет наши навыки программирования. Эти знания позволяют разработать более эффективные программы и грамотно управлять данными, что является критически важным в современном программировании и разработке приложений.