МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра информатики и систем управления

Методы сортировки
Отчет по лабораторной работе №5
Вариант № 17

по дисциплине

Алгоритмы и структуры данных

РУКОВОДИТЕЛЬ:

_____ Капранов С.Н.

СТУДЕНТ:

_____ Сапожников В.О.

19-ИВТ-3

Работа защищена «___» _____

С оценкой _____

Нижний Новгород 2020

Текст задания

Реализовать внешнюю сортировку

Используемый язык программирования: Java

Текст программы

Main.java

```
import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.InputMismatchException;

/**
 * Класс, содержащий точку входа в программу - метод  main.
 * Язык: java
 *
 * Реализация пятой лабораторной работы по диспилине: Алгоритмы
 * и структуры данных. Вариант№17.
 *
 * Текст задания: Реализовать внешнюю сортировку.
 *
 * @release: 8.12.20
 * @last_update: 8.12.20
 *
 * @author Vladislav Sapozhnikov 19-IVT-3
 */
public class Main
{
    /**
     * Точка входа в программу - функция main.
     *
     * @param args - список параметров запуска.
     * */
    public static void main(String[] args)
    {
        try
        {
            Sort.externalFileSorting(args[0]);   //сортировка
                                                  //переданного
                                                  //файла.
        }
        catch (FileNotFoundException e)
        {

            System.err.println("Файл: '" + args[0] + "' не +
                               + найден.");
        }
        catch (InputMismatchException e)
        {
            System.err.println("Неверное содержимое файла!");
            System.err.println("Ожидается: целочисленные +
```

```
                                + значения.");
        }
        catch (IOException e)
        {
            e.printStackTrace();
        }
    }
}
```

Sort.java

```
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.LinkedList;
import java.util.Scanner;

/**
 * Интерфейс содержащий методы необходимые
 * для внешней сортировки.
 *
 * Предназначен для работы только с целочисленными значениями.
 * */
public interface Sort
{
    /**
     * Сортировка вставками.
     * На вход подается не сортированный целочисленный массив.
     * На выходе получается массив, все элементы которого
     * отсортированы, т.е. удовлетворяют условию:
     *                A1 < A2 < ... < An
     *  @param list - входной целочисленный массив.
     * */
    static void insertionSort(LinkedList<Integer> list)
    {
        for (int i = 1; i < list.size(); i++)    //проход по
                                                 //входному
        {                                        //массиву
            int value = list.get(i);             //'вытаскиваем'
                                                 //значение для
                                                 //сортировки
            int j = i - 1;                       //перемещение по
                                                 //элементам

            for (; j >= 0; j--)                   //перед
            {                                     //'вытащенным'
                                                  //значением

                if (value < list.get(j))         //если
                {                                //'вытащенный'
                                                 //элемент меньше
```

```
                                                //элемента
                                                //отсортированной
                                                //части, то
                    list.set(j+1, list.get(j)); //смещаем все
                                                //следующие
                                                //элементы
                                                //отсортированной
                                                //части вправо
                }
                else                            //если элемент
                {                               //больше
                    break;                      //сортированной
                                                //части,
                }                               //то - прервать
            }

            //В освободившееся место вставляем вытащенное
            //значение
            list.set(j+1, value);
        }
    }

    /**
     * Метод, производящий слияние двух отсортированных
     * частей в новый файл.
     *
     * @throws IOException - исключение потока ввода-вывода
     *
     * @param list1  - первый список для слияния в новый файл
     * @param list2  - второй список для слияния в новый файл
     * @param writer - поток вывода в файл
     * */
    static void merge(FileWriter writer, LinkedList<Integer>
         list1, LinkedList<Integer> list2) throws IOException {
        if (list1.isEmpty())                //если 1ый список
        {                                   //уже пуст
                                            //дозаписываем 2ой
            for (Integer integer : list2)   //в файл и прерываем
            {                               //рекурсию
                writer.write(integer + " ");
            }
            return;
        }

        else if (list2.isEmpty())           //если 2ой список
        {                                   //уже пуст
                                            //дозаписываем
                                            //1ый список
            for (Integer integer : list1)   //в файл и
            {                               //прерываем
                writer.write(integer + " ");   //рекурсию
```

```
            }
            return;
        }

        for (int j = 0; j < list1.size();j ++)
        {
            for (int k = 0; k < list2.size(); k++)
            {
                //если элемент из 1ого
                //списка меньше или равен элемента
                //из 2ого списка, то записываем
                //элемент из 1ого списка и
                //удаляем её из списка. Прерывание
                if (list1.get(j) <= list2.get(k))
                {
                    writer.write(list1.get(j) + " ");
                    list1.remove(j);
                    break;
                }
                //если элемент из 2ого
                //списка меньше элемента из 1ого списка,
                //то записываем элемент из второго списка
                //и удаляем его из списка
                else
                {
                    writer.write(list2.get(k) + " ");
                    list2.remove(k);
                    break;
                }
            }
            break;
        }
        //рекурсивный вызов пока оба списка
        //не окажутся пустыми
        merge(writer, list1, list2);
    }

    /**
     * Алгоритм внешней сортировки.
     *
     * @throws IOException - исключение потока ввода-вывода
     *
     * @param fileNames - список файлов отсортированных
     *                    на 1ом этапе сортировки: разделение
     *                    общего файла и сортировка его
     *                    элементов сортировкой вставками.
     *
     * @param ch        - счетчик для изменения имен новых
     *                    файлов.
     *                    необходим для рекурсивного вызова
     * */
```

```
    static void externalSort(LinkedList<String> fileNames, char
ch) throws IOException {
        ch = (char)((int)ch + 1);        //увеличение счетчика

        //список потоков чтения
        LinkedList<Scanner> scannerList = new LinkedList<>();

        //заполнение списка потоков чтения на основе
        //списка с файлами
        for (String fileName : fileNames)
        {
            scannerList.add(new Scanner(new File(fileName)));
        }

        //очищаем список имен файлов для нового заполнения
        fileNames.clear();

        //цикл работы с потоками чтения
        for (int i = 0, j=1;i<scannerList.size()-1;i += 2, j++)
        {   //формирование имени нового файла
            //если это конечное слияние, то задаем
            //имя output.txt
            //иначе формируем имя используя индексы
            String fileName;
            if (scannerList.size() == 2)
            {
                fileName = "output.txt";
            }
            else
            {
                fileName = "buff" + ch + j + ".txt";
            }

            //открытие потока ввода в файл
            FileWriter writer = new FileWriter(fileName);


            //добавление имени файла в список имен файла
            fileNames.add(fileName);

            //списки для записи элементов
            LinkedList<Integer> list1 = new LinkedList<>();
            LinkedList<Integer> list2 = new LinkedList<>();

            //элементы из i-ого потока ввода записываем в 1ый
            //список
            while(scannerList.get( i ).hasNextInt())
            {
                list1.add(scannerList.get( i ).nextInt());
            }
```

```
            //элементы из i+1 потока ввода записываем во 2ой
            //список
            while(scannerList.get(i+1).hasNextInt())
            {
                list2.add(scannerList.get(i+1).nextInt());
            }

            merge(writer, list1, list2);  //производим слияние
                                          //списков в новый файл

            writer.close();              //закрываем поток ввода

            //рекурсивный вызов пока в списке файлов не будет 1
            //имя
            while (fileNames.size() != 1)
            {
                externalSort(fileNames, ch);
            }
        }

        //закрытие потоков чтения
        for (Scanner scanner : scannerList)
        {
            scanner.close();
        }
    }

    /**
     * Метод внешней сортировки для файла
     *
     * @param path - путь к файлу для сортировки
     * */
    static void externalFileSorting(String path) throws
                                                        IOException
    {
        //1ый этап - разделение исходного файла
        //и их сортировки вставками
        LinkedList<String> fileNames =
                                  readAndWrite.splitAndSort(path);

        //внешняя сортировка
        externalSort(fileNames, 'A');
    }
}
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.util.InputMismatchException;
import java.util.LinkedList;
import java.util.Scanner;

/**
 * Интерфейс, реализующий методы для чтения
 * и записи файлов.
 * */
public interface readAndWrite
{
    /**
     * Вспомогательный метод, записывающий переданный список
     * в новый файл.
     *
     * @throws IOException - генерация исключения потока ввода-
     *                       вывода.
     *
     * @param list         - список для записи файл.
     * @param filenames    - список, в который заносятся имена.
     *                       созданных файлов.
     * @param counter      - счетчик созданных файлов.
     * */
    static void writeListToFile(LinkedList<Integer> list,
   LinkedList<String> filenames, int counter) throws IOException
    {
        //формирование имени нового файла
        String path = "buffA" + counter + ".txt";

        //открытие потока на запись в новый файл
        FileWriter writer = new FileWriter(path);

        //цикл для посимвольной записи списка в файл
        for (Integer integer : list)
        {
            writer.write(integer + " ");
        }

        filenames.add(path);  //добавление нового файла в список

        writer.close();       //закрытие потока вывода
    }
```

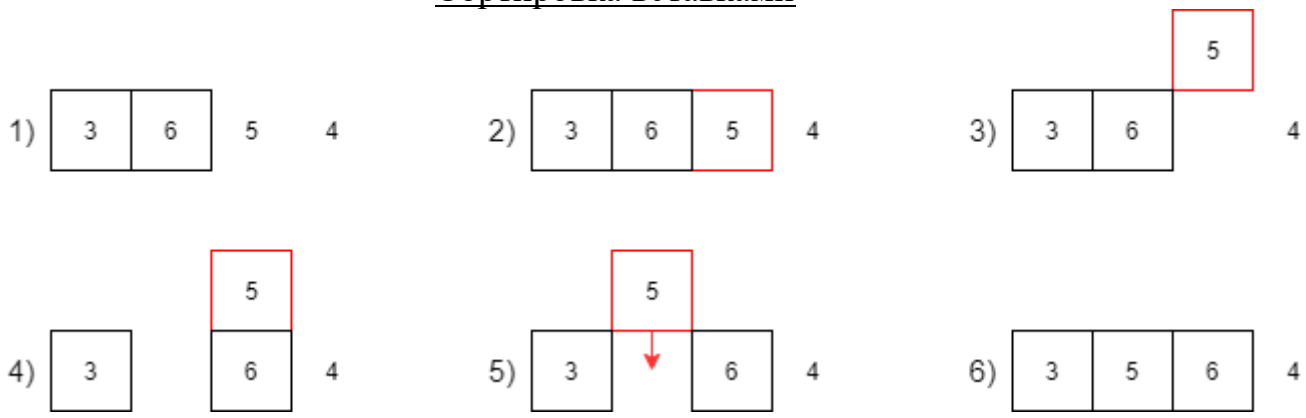
```
    /**
     * Вспомогательный метод, возвращающий кол-во элементов в
     * файле.
     *
     * @throws FileNotFoundException  - генерация исключения –
     *                                  файл не найден.
     * @throws InputMismatchException - генерация исключения –
     *                                  неверный тип считанных
     *                                  данных.
     *
     * @param path                    - путь к файлу.
     *
     * @return возвращает целочисленное значение - кол-во
     *         элементов в файле.
     * */
    static int getFileSize(String path) throws
                  FileNotFoundException, InputMismatchException
    {
        //открытие потока на чтение
        Scanner scanner = new Scanner(new File(path));

        int size = 0;

        //перебор элементов файла в цикле.
        while(scanner.hasNext())
        {
            scanner.nextInt();
            size++;
        }

        scanner.close();

        return size;
    }


    /**
     * Метод, разделяющий переданный файл на сортированные
     * четверти. Необходимо для дальнейшей сортировки и слияния.
     * Слияние происходит значительно быстрее если данные для
слияния
     * заранее отсортированы.
     *
     * Ограничение на кол-во разделяемых файлов - 4 было
     * установлено для более компактного представления работы.
     * в зависимости от объема данных и объема оперативной
     * памяти ограничение может варьироваться
     *
     * @throws FileNotFoundException  - генерация исключения –
     *                                  файл не найден.
     * @throws InputMismatchException - генерация исключения –
```

```
 *                                  неверный тип считанных
 *                                  данных.
 *
 * @param path                    - путь к файлу.
 *
 * @return возвращает список созданных файлов.
 * */
static LinkedList<String> splitAndSort(String path) throws
                          IOException, InputMismatchException
{
    //открытие потока чтения
    Scanner scanner = new Scanner(new File(path));

    //получаем общее кол-во элементов в файле
    int inputSize = getFileSize(path);

    //ограничиваем кол-во элементов для чтения
    //четвертью от общего кол-ва элементов
    int limit = 4;
    int fileSize  = inputSize/limit;

    //создание списка для записи прочтенных
    //элементов из файла.
    LinkedList<Integer> list     = new LinkedList<>();

    //создание списка для записи имен созданных файлов
    LinkedList<String> fileNames = new LinkedList<>();

    int counter = 1;    //счетчик созданных файлов.

    //цикл посимвольного чтения из файла
    while(scanner.hasNextInt())
    {
        //Если достигнуто ограничение и считываемый
        //промежуток не последний, то записываем сортируем
        //список в файл и очищаем список
        if (list.size() == fileSize && counter != limit)
        {
            writeListToFile(list, fileNames, counter);
            list.clear();
            counter++;
        }
        //иначе читаем до конца файла.
        //если кол-во элементов в файле на кратно
        //ограничению, то в последний промежуток
        //записываем остаточные данные
        list.add(scanner.nextInt());

        Sort.insertionSort(list);   //сортировка вставками
    }
```

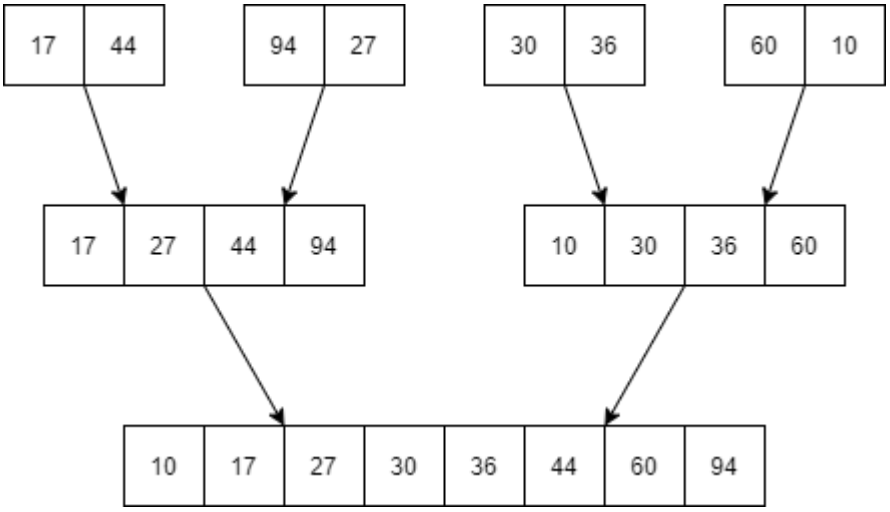
```
            //запись в файл
            writeListToFile(list, fileNames, counter);

            scanner.close();     //закрытие потока чтения

            return fileNames;    //вернуть список файлов.
        }
}
```

Используемые алгоритмы сортировки

Сортировка вставками

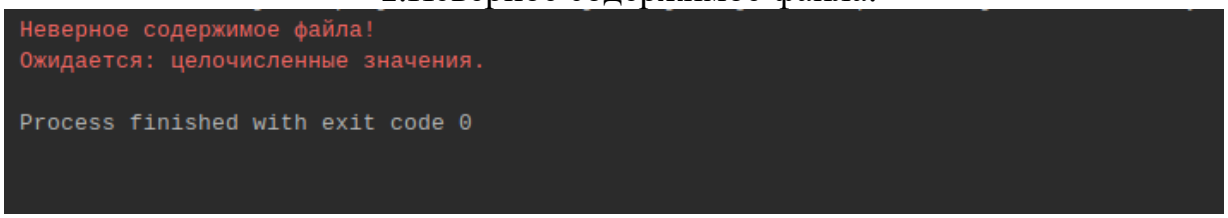


Сортировка слиянием

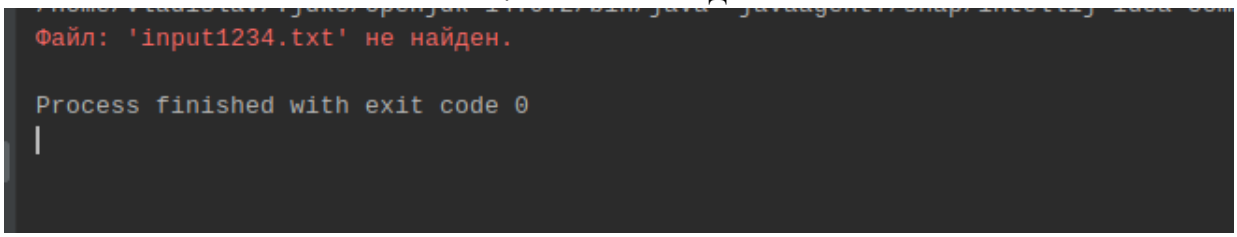


Скриншоты

1.Неверное содержимое файла.



2.Файл не найден.



3.Корректная работа программы.

Входные параметры

input.txt:

```
3 9 30 53 7 24 40 35 30 70 19 94 15 79 22 31 59 41 87 89 16 54 76 14 64 2 83 64 37 71 9 26 83 38 81 51 56 94 2 17 3 61 26
```

1ый этап

buffA1.txt:

```
3 7 9 24 30 30 35 40 53 70
```

buffA2.txt:

```
15 19 22 31 41 59 79 87 89 94
```

buffA3.txt:

```
2 14 16 37 54 64 64 71 76 83
```

buffA4.txt:

```
2 3 9 17 26 26 38 51 56 61 81 83 94
```

2ой этап

buffB1.txt (слияние A1 и A2):

```
3 7 9 15 19 22 24 30 30 31 35 40 41 53 59 70 79 87 89 94
```

buffB2.txt (слияние A3 и A4):

```
2 2 3 9 14 16 17 26 26 37 38 51 54 56 61 64 64 71 76 81 83 83 94
```

3ий этап

output.txt (слияние B1 и B2):

```
2 2 3 3 7 9 9 14 15 16 17 19 22 24 26 26 30 30 31 35 37 38 40 41 51 53 54 56 59 61 64 64 70 71 76 79 81 83 83 87 89 94 94
```