УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №3.3

по предмету «Основы алгоритмизации и программирования»

Вариант 20

Выполнил:

Машевский Д.В

Гр. 351003

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Сортировка естественным слиянием.

**Код Delphi:**

Program LABA33;

Uses

System.SysUtils;

Type

TArray = Array Of Integer;

Const

MAX\_N = 100;

MIN\_N = 2;

MAX\_RANGE = 1000000000;

MIN\_RANGE = -1000000000;

Procedure GetInfo();

Begin

Writeln('The program implements sorting by natural merging');

End;

Procedure ZeroArr(Var Arr: TArray);

Var

I: Integer;

Begin

For I := 0 To High(Arr) Do

Arr[I] := 0;

End;

Function ChooseOption(): Integer;

Var

Choice: Integer;

Begin

Choice := 0;

Writeln('Enter 0 to work with data from the console, and 1 to work from the

file:');

Repeat

Try

Readln(Choice);

Except

Writeln('Invalid data entered. Re-enter:');

End;

Until (Choice = 0) Or (Choice = 1);

ChooseOption := Choice;

End;

Function PTF(): String;

Var

Path: String;

IsCorrect: Boolean;

Begin

Repeat

IsCorrect := True;

Write('Path to file: ');

Readln(Path);

If Not FileExists(Path) Then

Begin

Write('No such file was found. ');

IsCorrect := False;

End;

If IsCorrect And (ExtractFileExt(Path) <> '.txt') Then

Begin

IsCorrect := False;

Write('The file must have a txt extension.');

End;

Until IsCorrect;

PTF := Path;

End;

Function WorkWithDIg(Var Arr, DigArr: TArray): TArray;

Var

StartF, StopF, StartS, StopS: Integer;

I, J, Counter, SizeArr, PointerInd: Integer;

ChangedArr: TArray;

Begin

I := 0;

Counter := Length(DigArr) - Length(DigArr) Mod 4;

SizeArr := Length(DigArr) Div 2;

PointerInd := 0;

SetLength(ChangedArr, Length(Arr));

Repeat

StartF := DigArr[PointerInd];

Inc(PointerInd);

StopF := DigArr[PointerInd];

Inc(PointerInd);

StartS := DigArr[PointerInd];

Inc(PointerInd);

StopS := DigArr[PointerInd];

Inc(PointerInd);

While (StartF < StopF) And (StartS < StopS) Do

Begin

If Arr[StartF] > Arr[StartS] Then

Begin

ChangedArr[I] := Arr[StartS];

Inc(I);

Inc(StartS);

End

Else

Begin

ChangedArr[I] := Arr[StartF];

Inc(I);

Inc(StartF);

End;

End;

While StartF < StopF Do

Begin

ChangedArr[I] := Arr[StartF];

Inc(I);

Inc(StartF);

End;

While StartS < StopS Do

Begin

ChangedArr[I] := Arr[StartS];

Inc(I);

Inc(StartS);

End;

Until (PointerInd >= Counter) Or (DigArr[PointerInd] = 0);

If (I < SizeArr) Then

For J := DigArr[PointerInd] To DigArr[PointerInd + 1] - 1 Do

Begin

ChangedArr[I] := Arr[J];

Inc(I);

End;

WorkWithDIg := ChangedArr;

End;

Function NatMergeSort(Var Arr: TArray): TArray;

Var

I, PointInd, SizePointers: Integer;

DigArr: TArray;

Begin

SizePointers := 2 \* Length(Arr);

SetLength(DigArr, SizePointers);

Repeat

ZeroArr(DigArr);

PointInd := 0;

DigArr[PointInd] := 0;

Inc(PointInd);

For I := 1 To High(Arr) Do

If Arr[I] < Arr[I - 1] Then

Begin

DigArr[PointInd] := I;

Inc(PointInd);

DigArr[PointInd] := I;

Inc(PointInd);

End;

DigArr[PointInd] := Length(Arr);

Arr := WorkWithDIg(Arr, DigArr);

Until DigArr[1] = Length(Arr);

NatMergeSort := Arr;

End;

Function InpValidSize(Var Size: Integer): Integer;

Var

Line: String;

IsCorrect: Boolean;

Begin

Repeat

Readln(Line);

Try

Size := StrToInt(Line);

IsCorrect := (Size > MIN\_N) And (Size < MAX\_N);

Except

IsCorrect := False;

Writeln('Re-Enter');

End;

Until IsCorrect;

InpValidSize := Size;

End;

Function InpValidArr(Var Arr: TArray): TArray;

Var

I: Integer;

Line: String;

IsCorrect: Boolean;

Begin

For I := 0 to Length(Arr) - 1 do

Begin

Repeat

Readln(Line);

Try

Arr[I] := StrToInt(Line);

IsCorrect := (Arr[I] > MIN\_RANGE) And (Arr[I] < MAX\_RANGE);

Except

IsCorrect := False;

Writeln('Re-Enter');

End;

Until IsCorrect;

End;

InpValidArr := Arr;

End;

Function InputFromConsole(): TArray;

Var

StartArr: TArray;

Size: Integer;

Begin

Writeln('Enter the size[2, 100] and then the', #13#10,

'elements[-1000000000, 1000000000]');

Size := InpValidSize(Size);

SetLength(StartArr, Size);

Writeln('Enter the ', Size, ' elements');

StartArr := InpValidArr(StartArr);

InputFromConsole := StartArr;

End;

Function ReadArrFromFile(Var Arr: TArray; Var MyFile: TextFile): TArray;

Var

Line: String;

I: Integer;

IsCorrect: Boolean;

Begin

I := 0;

While (I < Length(Arr)) Do

Begin

IsCorrect:= True;

Try

Read(MyFile, Arr[I]);

Except

IsCorrect:= False;

Writeln('Re-Enter');

End;

If (Arr[I] > MAX\_RANGE) Or (Arr[I] < MIN\_RANGE) Then

Begin

IsCorrect:= False;

Writeln('Re-Enter');

End;

Inc(I);

If (I = Length(Arr)) And (Not Eof(MyFile)) Then

Begin

IsCorrect:= False;

Writeln('Re-Enter');

End;

End;

ReadArrFromFile := Arr;

End;

Function ReadSizeFromFile(Var Size: Integer; Var MyFile: TextFile): Integer;

Var

IsCorrect: Boolean;

Line: String;

Begin

Try

IsCorrect:= True;

Read(MyFile, Size);

Except

IsCorrect:= False;

Writeln('Re-Enter');

End;

If (Size > MAX\_N) Or (Size < MIN\_N) Then

Begin

IsCorrect:= False;

Writeln('Re-Enter');

End;

ReadSizeFromFile := Size;

End;

Function InputFromFile(): TArray;

Var

F: String;

InfFile: TextFile;

StartArr: TArray;

Size: Integer;

Begin

Writeln('Enter full path to file');

F := PTF();

AssignFile(InfFile, F);

Reset(InfFile);

Size := ReadSizeFromFile(Size, InfFile);

SetLength(StartArr, Size);

StartArr := ReadArrFromFile(StartArr, InfFile);

CloseFile(InfFile);

Writeln('Reading is successfull');

InputFromFile := StartArr;

End;

Function CopyArr(Var Arr: TArray): TArray;

Var

CopyedArr: TArray;

I: Integer;

Begin

SetLength(CopyedArr, Length(Arr));

For I := 0 To High(Arr) Do

CopyedArr[I] := Arr[I];

CopyArr := CopyedArr;

End;

Function Input(): TArray;

Var

StartArr: TArray;

Choice: Integer;

Begin

Choice := ChooseOption();

If (Choice = 0) Then

StartArr := InputFromConsole()

Else

StartArr := InputFromFile();

Input := StartArr;

End;

Procedure OutputInConsole(Var StartArr, SortArr: TArray);

Var

I: Integer;

Begin

Writeln('Start array');

For I := 0 To High(StartArr) Do

Write(StartArr[I], ' ');

Writeln(#13#10, 'Sorted array');

For I := 0 To High(SortArr) Do

Write(SortArr[I], ' ');

End;

Procedure OutputInFile(Var StartArr, SortArr: TArray);

Var

F: String;

I: Integer;

MyFile: TextFile;

Begin

Writeln('Enter full path to file');

F := PTF();

AssignFile(MyFile, F);

Rewrite(MyFile);

Writeln(MyFile, 'Start array');

For I := 0 To High(StartArr) Do

Write(MyFile, StartArr[I], ' ');

Writeln(MyFile, #13#10, 'Sorted array');

For I := 0 To High(SortArr) Do

Write(MyFile, SortArr[I], ' ');

CloseFile(MyFile);

End;

Procedure Output(Var StartArr, SortArr: TArray);

Var

Choice: Integer;

Begin

Choice := ChooseOption();

If (Choice = 0) Then

OutputInConsole(StartArr, SortArr)

Else

OutputInFile(StartArr, SortArr);

End;

Var

StartArr, SortArr, SortArr1: TArray;

Begin

GetInfo();

StartArr := Input();

SortArr := CopyArr(StartArr);

SortArr1 := NatMergeSort(SortArr);

Output(StartArr, SortArr1);

Readln;

End.

**Код С++:**

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

const int

MIN\_NUMB = -1000000000,

MAX\_NUMB = 1000000000,

MIN\_SIZE = 2,

MAX\_SIZE = 100;

void getInfo() {

cout << "The program implements sorting by natural merging" << endl;

}

void zeroArr(int\* arr, int size) {

for (int i = 0; i < size; i++) {

arr[i] = 0;

}

}

int chooseOption() {

int choice;

bool isInCorrect;

cout << "Enter 0 if you want to work with data from the console, and 1 if from

the file : ";

do {

isInCorrect = false;

cin >> choice;

if (cin.fail() || cin.peek() != '\n') {

isInCorrect = true;

cout << "Invalid data entered. Re-enter:";

cin.clear();

while (cin.get() != '\n');

}

if (!(isInCorrect) && (choice != 1) && (choice != 0)) {

isInCorrect = true;

cout << "Re-enter: ";

cin.clear();

while (cin.get() != '\n');

}

} while (isInCorrect);

return choice;

}

string pathToFile() {

string path;

bool isIncorrect;

do

{

isIncorrect = false;

cout << "Enter the path to the file: ";

cin >> path;

ifstream fin(path);

if (!fin.is\_open())

{

cout << "No such file was found. ";

isIncorrect = true;

}

fin.close();

} while (isIncorrect);

return path;

}

int\* workWithDIg(int\* arr, int sizeArr, int\* digArr, int sizePointer) {

int startF, stopF, startS, stopS;

int i, j, counter, pointerInd;

int\* changedArr = new int[sizeArr];

i = 0;

counter = sizePointer - sizePointer % 4;

pointerInd = 0;

do

{

startF = digArr[pointerInd++];

stopF = digArr[pointerInd++];

startS = digArr[pointerInd++];

stopS = digArr[pointerInd++];

while (startF < stopF && startS < stopS)

{

if (arr[startF] > arr[startS])

changedArr[i++] = arr[startS++];

else

changedArr[i++] = arr[startF++];

}

while (startF < stopF)

{

changedArr[i++] = arr[startF++];

}

while (startS < stopS)

{

changedArr[i++] = arr[startS++];

}

} while (pointerInd < counter && changedArr[pointerInd] > 0);

if (i < sizeArr)

{

for (j = digArr[pointerInd++]; j < digArr[pointerInd]; j++)

{

changedArr[i++] = arr[j];

}

}

return changedArr;

}

int\* natMergeSort(int\* arr, int size) {

int i, pointInd, sizePointers = 0;

int\* digArr = new int[sizePointers];

sizePointers = 2 \* size;

do {

zeroArr(digArr, sizePointers);

pointInd = 0;

digArr[pointInd++] = 0;

for (i = 1; i < size; i++) {

if (arr[i] < arr[i - 1]) {

digArr[pointInd++] = i;

digArr[pointInd++] = i;

}

}

digArr[pointInd] = i;

arr = workWithDIg(arr, size, digArr, sizePointers);

} while (digArr[1] != size);

delete[] arr;

return arr;

}

int inpValidSize(int& size) {

bool isCorrect;

do {

isCorrect = true;

cin >> size;

if (cin.fail() || cin.peek() != '\n')

{

cout << "Error. \n";

isCorrect = false;

}

} while (!isCorrect);

return size;

}

int\* inpValidArr(int\* arr, int size)

{

int i;

i = 0;

bool isCorrect;

isCorrect = true;

while (i < size)

{

do {

cin >> arr[i];

if (cin.fail() || cin.peek() != '\n')

{

cout << "Error. \n";

isCorrect = false;

}

} while (!isCorrect);

i++;

}

return arr;

}

int\* readArrFromFile(int\* arr, int size, ifstream& file, string path)

{

int i;

i = 0;

bool isCorrect;

isCorrect = true;

ifstream fin(path);

while (i < size)

{

file >> arr[i];

if (file.fail() || file.peek() != '\n')

{

cout << "Error. \n";

isCorrect = false;

}

i++;

}

fin.close();

if (!isCorrect)

{

arr = inpValidArr(arr, size);

}

return arr;

}

int readSizeFromFile(int& size, ifstream& file, string path) {

bool isCorrect;

isCorrect = true;

ifstream fin(path);

file >> size;

if (file.fail() || file.peek() != '\n')

{

cout << "Error. \n";

isCorrect = false;

}

if ((size > MAX\_SIZE) || (size < MIN\_SIZE))

isCorrect = false;

return size;

}

int\* inputFromConsole(int& size)

{

cout << "Enter the size[2, 100] and then the \n"

<< "elements[-1000000000, 1000000000] through the Enter\n";

size = inpValidSize(size);

int\* defaultArr = new int[size];

defaultArr = inpValidArr(defaultArr, size);

return defaultArr;

}

int\* inputFromFile(int& size, string path)

{

string fileName;

path = pathToFile();

ifstream fin(path);

int\* startArr = new int;

size = readSizeFromFile(size, fin, path);

startArr = new int[size];

startArr = readArrFromFile(startArr, size, fin, path);

cout << "Reading is successfull\n";

return startArr;

}

int\* inputInf(int& size, string path)

{

int\* startArr = new int [size];

int choice = chooseOption();

if (choice == 0)

startArr = inputFromConsole(size);

else

startArr = inputFromFile(size, path);

return startArr;

}

void outputInConsole(int\* startArr, int\* sortedArr, int size)

{

int i;

cout << "Start array\n";

for (i = 0; i < size; i++)

cout << startArr[i] << " ";

cout << "\nSorted array\n";

for (i = 0; i < size; i++)

cout << sortedArr[i] << " ";

}

void outputInFile(int\* startArr, int\* sortedArr, int size)

{

string fileName = pathToFile();

ofstream file(fileName);

int i;

file << "Default array\n";

for (i = 0; i < size; i++)

file << startArr[i] << " ";

file << "\nSorted array\n";

for (i = 0; i < size; i++)

file << sortedArr[i] << " ";

cout << "Writing is successfull\n";

file.close();

}

int\* copyArr(int\* arr, int size) {

int\* copyedArr = new int[size];

int i;

for (i = 0; i < size; i++) {

copyedArr[i] = arr[i];

}

return copyedArr;

}

void outPut(int\* startArr, int\* sortedArr, int size)

{

int choice = chooseOption();

if (choice == 0)

outputInConsole(startArr, sortedArr, size);

else

outputInFile(startArr, sortedArr, size);

delete[] startArr;

delete[] sortedArr;

}

int main() {

int size;

int\* defaultArr;

int\* sortedArr;

string path;

getInfo();

defaultArr = inputInf(size, path);

sortedArr = copyArr(defaultArr, size);

sortedArr = natMergeSort(defaultArr, size);

outPut(defaultArr, sortedArr, size);

return 0;

}

**Код Java:**

import java.io.File;

import java.io.FileNotFoundException;

import java.util.Scanner;

public class LABA33 {

public static void main(String[] args) {

getInfo();

int[] startArr = input();

int[] sortArr = copyArr(startArr);

int[] sortArr1 = natMergeSort(sortArr);

output(startArr, sortArr1);

}

private static void getInfo() {

System.out.println("The program implements sorting by natural merging");

}

private static void zeroArr(int[] arr) {

for (int i = 0; i < arr.length; i++) {

arr[i] = 0;

}

}

private static int chooseOption() {

int choice = 0;

Scanner scanner = new Scanner(System.in);

System.out.println("Enter 0 to work with data from the console, and 1 to work

from the file:");

while (true) {

try {

choice = Integer.parseInt(scanner.nextLine());

if (choice == 0 || choice == 1) {

break;

} else {

System.out.println("Invalid data entered. Re-enter:");

}

} catch (NumberFormatException e) {

System.out.println("Invalid data entered. Re-enter:");

}

}

return choice;

}

private static String ptf() {

String path = "";

boolean isCorrect;

Scanner scanner = new Scanner(System.in);

do {

isCorrect = true;

System.out.print("Path to file: ");

path = scanner.nextLine();

File file = new File(path);

if (!file.exists()) {

System.out.print("No such file was found. ");

isCorrect = false;

}

if (isCorrect && !path.endsWith(".txt")) {

isCorrect = false;

System.out.print("The file must have a txt extension.");

}

} while (!isCorrect);

return path;

}

private static int[] workWithDig(int[] arr, int[] digArr) {

int startF, stopF, startS, stopS;

int i = 0;

int counter = digArr.length - digArr.length % 4;

int sizeArr = digArr.length / 2;

int pointerInd = 0;

int[] changedArr = new int[arr.length];

while (true) {

startF = digArr[pointerInd];

pointerInd++;

stopF = digArr[pointerInd];

pointerInd++;

startS = digArr[pointerInd];

pointerInd++;

stopS = digArr[pointerInd];

pointerInd++;

while (startF < stopF && startS < stopS) {

if (arr[startF] > arr[startS]) {

changedArr[i] = arr[startS];

i++;

startS++;

} else {

changedArr[i] = arr[startF];

i++;

startF++;

}

}

while (startF < stopF) {

changedArr[i] = arr[startF];

i++;

startF++;

}

while (startS < stopS) {

changedArr[i] = arr[startS];

i++;

startS++;

}

if (pointerInd >= counter || digArr[pointerInd] == 0) {

break;

}

}

if (i < sizeArr) {

for (int j = digArr[pointerInd]; j < digArr[pointerInd + 1]; j++) {

changedArr[i] = arr[j];

i++;

}

}

return changedArr;

}

private static int[] natMergeSort(int[] arr) {

int i, pointInd, sizePointers;

int[] digArr;

sizePointers = 2 \* arr.length;

digArr = new int[sizePointers];

do {

zeroArr(digArr);

pointInd = 0;

digArr[pointInd] = 0;

pointInd++;

for (i = 1; i < arr.length; i++) {

if (arr[i] < arr[i - 1]) {

digArr[pointInd] = i;

pointInd++;

digArr[pointInd] = i;

pointInd++;

}

}

digArr[pointInd] = arr.length;

arr = workWithDig(arr, digArr);

} while (digArr[1] != arr.length);

return arr;

}

private static int inpValidSize() {

int size = 0;

boolean isCorrect;

Scanner scanner = new Scanner(System.in);

do {

String line = scanner.nextLine();

try {

size = Integer.parseInt(line);

isCorrect = (size >= 2) && (size <= 100);

} catch (NumberFormatException e) {

isCorrect = false;

System.out.println("Re-Enter");

}

} while (!isCorrect);

return size;

}

private static int[] inpValidArr(int[] arr) {

boolean isCorrect;

Scanner scanner = new Scanner(System.in);

for (int i = 0; i < arr.length; i++) {

do {

String line = scanner.nextLine();

try {

arr[i] = Integer.parseInt(line);

isCorrect = (arr[i] >= -1000000000) && (arr[i] <= 1000000000);

} catch (NumberFormatException e) {

isCorrect = false;

System.out.println("Re-Enter");

}

} while (!isCorrect);

}

return arr;

}

private static int[] inputFromConsole() {

System.out.println("Enter the size[2, 100] and then the\n" +

"elements[-1000000000, 1000000000]");

int size = inpValidSize();

int[] startArr = new int[size];

System.out.println("Enter the " + size + " elements");

startArr = inpValidArr(startArr);

return startArr;

}

private static int[] readArrFromFile(int[] arr, Scanner scanner) {

int i = 0;

while (i < arr.length) {

boolean isCorrect = true;

try {

arr[i] = Integer.parseInt(scanner.next());

} catch (NumberFormatException e) {

isCorrect = false;

System.out.println("Re-Enter");

}

if (arr[i] > 1000000000 || arr[i] < -1000000000) {

isCorrect = false;

System.out.println("Re-Enter");

}

i++;

if (i == arr.length && scanner.hasNext()) {

isCorrect = false;

System.out.println("Re-Enter");

}

}

return arr;

}

private static int readSizeFromFile(Scanner scanner) {

int size = 0;

boolean isCorrect;

do {

try {

size = Integer.parseInt(scanner.next());

isCorrect = (size >= 2) && (size <= 100);

} catch (NumberFormatException e) {

isCorrect = false;

System.out.println("Re-Enter");

}

} while (!isCorrect);

return size;

}

private static int[] inputFromFile() {

System.out.println("Enter full path to file");

String f = ptf();

File file = new File(f);

Scanner scanner = null;

try {

scanner = new Scanner(file);

} catch (FileNotFoundException e) {

System.out.println("Error");

}

int size = readSizeFromFile(scanner);

int[] startArr = new int[size];

startArr = readArrFromFile(startArr, scanner);

scanner.close();

System.out.println("Reading is successful");

return startArr;

}

private static int[] copyArr(int[] arr) {

int[] copiedArr = new int[arr.length];

for (int i = 0; i < arr.length; i++) {

copiedArr[i] = arr[i];

}

return copiedArr;

}

private static int[] input() {

int[] startArr;

int choice = chooseOption();

if (choice == 0) {

startArr = inputFromConsole();

} else {

startArr = inputFromFile();

}

return startArr;

}

private static void outputInConsole(int[] startArr, int[] sortArr) {

System.out.println("Start array");

int i;

for (i = 0; i < startArr.length; i++) {

System.out.print(startArr[i] + " ");

}

System.out.println("\nSorted array");

for (i = 0; i < sortArr.length; i++) {

System.out.print(sortArr[i] + " ");

}

}

private static void outputInFile(int[] startArr, int[] sortArr) {

System.out.println("Enter full path to file");

String f = ptf();

int i;

File file = new File(f);

try {

java.io.PrintWriter printWriter = new java.io.PrintWriter(file);

printWriter.println("Start array");

for (i = 0; i < startArr.length; i++) {

printWriter.print(startArr[i] + " ");

}

printWriter.println("\nSorted array");

for (i = 0; i < sortArr.length; i++) {

printWriter.print(sortArr[i] + " ");

}

printWriter.close();

} catch (FileNotFoundException e) {

System.out.println("Error");

}

}

private static void output(int[] startArr, int[] sortArr) {

int choice = chooseOption();

if (choice == 0) {

outputInConsole(startArr, sortArr);

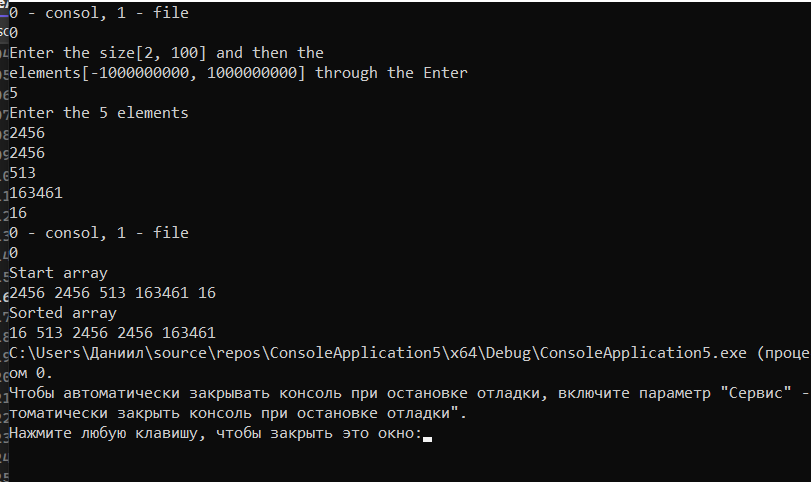
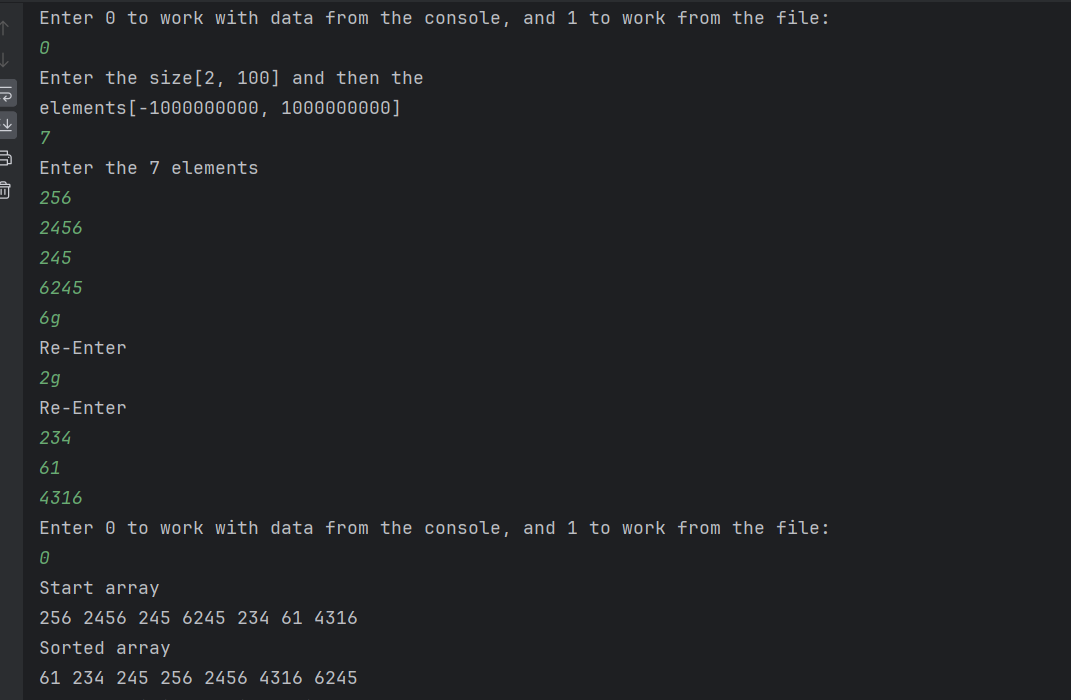
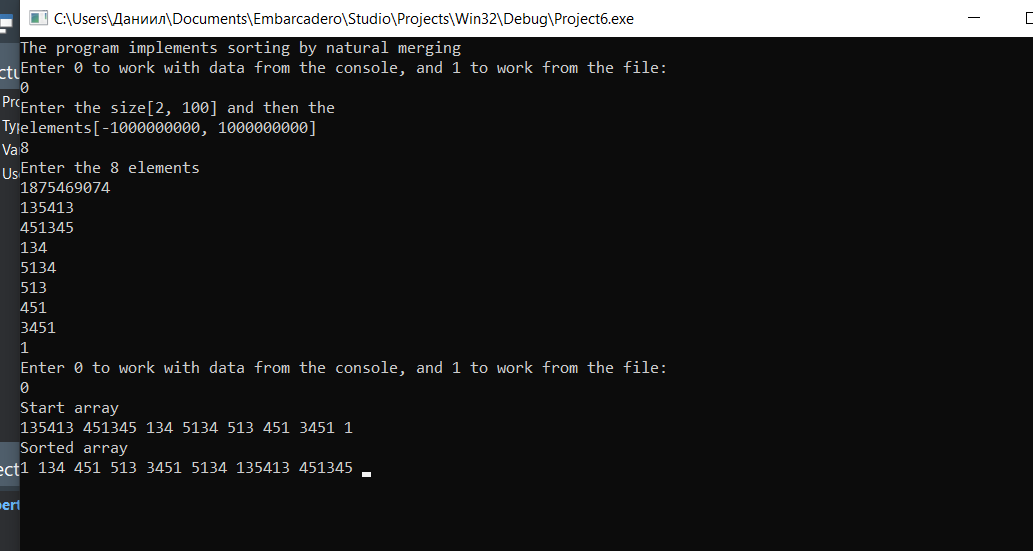
} else {

outputInFile(startArr, sortArr);

}

}

**Скриншоты:**



**Блок-схема:**

