УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №4.2

по предмету «Основы алгоритмизации и программирования»

Вариант 18

Выполнил:

Егоров А.С.

Гр. 351005

Проверил:

Данилова Г. В.

Минск 2023

**Задание:**

Дана последовательность чисел а 1 , а 2 , …, а n . Разработать рекурсивную процедуру сортировки последовательности методом быстрой сортировки

**Код программы Delphi:**

**UnitMain.pas**

Unit MainFormUnit4\_2;

Interface

Uses

Winapi.Windows, Winapi.Messages, System.SysUtils, System.Variants,

System.Classes, Vcl.Graphics,

Vcl.Controls, Vcl.Forms, Vcl.Dialogs, Vcl.Buttons, System.ImageList,

Vcl.ImgList, Vcl.Grids, Vcl.StdCtrls, Vcl.Menus,

InstructionUnit4\_2, AboutTheDeveloperUnit4\_2, BackendUnit4\_2,

OutputSortedArrayUnit4\_2, ExitUnit4\_2;

Type

// форма

TuVCLMain = Class(TForm)

lbTaskInfo: TLabel;

lbSizeInfo: TLabel;

eSize: TEdit;

bitBtAcceptSize: TButton;

strGrElementsOfArray: TStringGrid;

ImageList1: TImageList;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

MainMenu1: TMainMenu;

btFile: TMenuItem;

btOpenFile: TMenuItem;

btSaveFile: TMenuItem;

btInstruction: TMenuItem;

btAboutTheDeveloper: TMenuItem;

lbElementsInfo: TLabel;

bitBtSortArray: TBitBtn;

bitBtShowList: TBitBtn;

Procedure eSizeKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure eSizeKeyPress(Sender: TObject; Var Key: Char);

Procedure btInstructionClick(Sender: TObject);

Procedure btAboutTheDeveloperClick(Sender: TObject);

Procedure eSizeChange(Sender: TObject);

Procedure strGrElementsOfArrayKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure bitBtAcceptSizeClick(Sender: TObject);

Procedure strGrElementsOfArrayKeyPress(Sender: TObject; Var Key: Char);

Procedure bitBtSortArrayClick(Sender: TObject);

Procedure bitBtShowListClick(Sender: TObject);

Procedure btOpenFileClick(Sender: TObject);

Procedure btSaveFileClick(Sender: TObject);

Procedure strGrElementsOfArrayKeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure bitBtSortArrayKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Procedure FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Function FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Procedure FormCreate(Sender: TObject);

Procedure FormDestroy(Sender: TObject);

Private

Size: Integer;

IsFileSaved: Boolean;

IsArrayFilled: Boolean;

IsSortButtonPressed: Boolean;

WasChanges: Boolean;

BufferHandler: TBufferHandler;

Public

{ Public declarations }

End;

Var

uVCLMain: TuVCLMain;

Implementation

{$R \*.dfm}

Procedure TuVCLMain.btAboutTheDeveloperClick(Sender: TObject);

Begin

uVCLAboutTheDeveloper.Show;

End;

Procedure TuVCLMain.btInstructionClick(Sender: TObject);

Begin

uVCLInstruction.Show;

End;

Procedure TuVCLMain.btOpenFileClick(Sender: TObject);

Var

FileReader: TFileReader;

I: Integer;

Arr: TArray;

Begin

If OpenDialog1.Execute() Then

Begin

FileReader := TFileReader.Create;

FileReader.SetFileName(OpenDialog1.FileName);

If FileReader.IsFileGood() Then

Begin

Size := FileReader.InputSize();

If FileReader.GetStatus() Then

Begin

eSize.Text := IntToStr(Size);

bitBtAcceptSize.Click;

Arr := FileReader.InputArray(Size);

If FileReader.GetStatus() Then

Begin

For I := 1 To Size Do

Begin

strGrElementsOfArray.Cells[I, 1] := IntToStr(Arr[I - 1]);

ArraySorter.SetElementByIndex(Arr[I - 1], I - 1);

End;

bitbtSortArray.Enabled := True;

End

Else

MessageBox(uVCLMain.Handle,

'Элементы массива введены неправильно! Проверьте данные.',

'Ой-йой', MB\_ICONERROR)

End

Else

MessageBox(uVCLMain.Handle,'Размер введён неправильно или не

соответствует границам! Проверьте данные.',

'Ой-йой', MB\_ICONERROR);

End

Else

MessageBox(uVCLMain.Handle,

'Файл закрыт для чтения или не текстовый! ', 'Ошибка',

MB\_ICONERROR);

If Not FileReader.GetStatus() Then

Begin

eSize.Text := '';

BitbtAcceptSize.Enabled := False;

strGrElementsOfArray.Enabled := False;

strGrElementsOfArray.Visible := False;

End;

FileReader.Destroy;

FileReader := Nil;

End;

End;

Procedure TuVCLMain.btSaveFileClick(Sender: TObject);

Var

FileWriter: TFileWriter;

Begin

If SaveDialog1.Execute() Then

Begin

FileWriter := TFileWriter.Create;

FileWriter.SetFileName(SaveDialog1.FileName);

If FileWriter.IsFileGood() Then

Begin

FileWriter.OutputArray(ArraySorter.GetArray(), Size);

If FileWriter.GetStatus() Then

IsFileSaved := True

Else

MessageBox(uVCLMain.Handle, 'Упс.. Что-то пошло не так!',

'Ой-йой', MB\_ICONERROR);

End

Else

MessageBox(uVCLMain.Handle,

'Файл закрыт для записи или не текстовый!', 'Ой-йой',

MB\_ICONERROR);

FileWriter.Destroy;

FileWriter := Nil;

End;

End;

Procedure TuVCLMain.bitBtShowListClick(Sender: TObject);

Begin

Application.CreateForm(TuVCLOutputSortedArray, uVCLOutputSortedArray);

uVCLOutputSortedArray.ShowModal;

uVCLOutputSortedArray.Destroy;

uVCLOutputSortedArray := Nil;

End;

Procedure TuVCLMain.bitBtAcceptSizeClick(Sender: TObject);

Const

MAX\_SIZE: Integer = 1000;

MIN\_SIZE: Integer = 0;

Var

I: Integer;

Begin

// creating Grid

Size := StrToInt(eSize.Text);

If WasChanges Then

Begin

If (MIN\_SIZE < Size) And (Size < MAX\_SIZE) Then

Begin

If ArraySorter <> Nil Then

ArraySorter.Destroy();

ArraySorter := TArraySorter.Create(Size);

strGrElementsOfArray.RowCount := 2;

strGrElementsOfArray.ColCount := Size + 1;

strGrElementsOfArray.FixedCols := 1;

strGrElementsOfArray.FixedRows := 1;

strGrElementsOfArray.Cells[0, 0] := '№';

strGrElementsOfArray.Cells[0, 1] := 'Элемент';

For I := 1 To Size Do

Begin

strGrElementsOfArray.Cells[I, 0] := IntToStr(I);

strGrElementsOfArray.Cells[I, 1] := '';

End;

strGrElementsOfArray.Enabled := True;

lbElementsInfo.Visible := True;

strGrElementsOfArray.Visible := True;

bitBtSortArray.Visible := True;

bitBtShowList.Visible := True;

bitBtAcceptSize.Enabled := False;

ActiveControl := strGrElementsOfArray;

End

Else

MessageBox(uVCLMain.Handle,

'Размер не соответствует границам! Проверьте данные.', 'Ой-йой',

MB\_ICONERROR);

End;

WasChanges := False;

End;

Procedure TuVCLMain.eSizeChange(Sender: TObject);

Var

I, J: Integer;

SizeEdit: TEdit;

TempStr: String;

Begin

SizeEdit := TEdit(Sender);

BitbtAcceptSize.Enabled := Not String.IsNullOrEmpty(SizeEdit.Text);

// проверка на вставку

BufferHandler.EditText := SizeEdit.Text;

If Not BufferHandler.Status Then

Begin

MessageBox(uVCLMain.Handle, 'Вы ввели неправильные смиволы!', 'Ой-йой',

MB\_ICONERROR);

SizeEdit.Text := '';

End;

// ведущие 0

While (Length(SizeEdit.Text) > 0) And (SizeEdit.Text[1] = '0') Do

Begin

TempStr := SizeEdit.Text;

Delete(TempStr, 1, 1);

SizeEdit.Text := TempStr;

End;

If Not bitBtAcceptSize.Enabled Then

Begin

StrGrElementsOfArray.Enabled := False;

End;

If WasChanges Then

Begin

// Очистка всех ячеек StringGrid

For I := 1 To StrGrElementsOfArray.RowCount Do

Begin

For J := 1 To StrGrElementsOfArray.ColCount Do

Begin

StrGrElementsOfArray.Cells[J - 1, I - 1] := '';

End;

End;

IsArrayFilled := False;

LbElementsInfo.Visible := False;

StrGrElementsOfArray.Visible := False;

BitBtSortArray.Visible := False;

BitBtShowList.Visible := False;

BitBtSortArray.Enabled := False;

BitBtShowList.Enabled := False;

BtSaveFile.Enabled := False;

End;

WasChanges := True;

End;

Procedure TuVCLMain.bitBtSortArrayClick(Sender: TObject);

Var

UserArray: TArray;

Begin

UserArray := ArraySorter.GetUserArray();

ArraySorter.QuickSort(UserArray, 0, High(UserArray));

bitbtShowList.Enabled := True;

btSaveFile.Enabled := True;

IsFileSaved := False;

IsSortButtonPressed := True;

End;

Procedure TuVCLMain.bitBtSortArrayKeyDown(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Begin

If (bitBtShowList.Enabled) And ((Key = VK\_DOWN) Or (Key = VK\_RETURN)) Then

ActiveControl := BitBtShowList;

If (Key = VK\_UP) Then

ActiveControl := StrGrElementsOfArray;

End;

Procedure TuVCLMain.strGrElementsOfArrayKeyUp(Sender: TObject; Var Key: Word;

Shift: TShiftState);

Var

Counter, I: Integer;

Begin

Counter := 0;

For I := 1 To Size Do

Begin

If (Length(strGrElementsOfArray.Cells[I, 1]) <> 0) And

(strGrElementsOfArray.Cells[I, 1] <> '-') Then

Begin

ArraySorter.SetElementByIndex

(StrToInt(strGrElementsOfArray.Cells[I, 1]), I - 1);

Inc(Counter);

End;

End;

If Counter = Size Then

IsArrayFilled := True

Else

IsArrayFilled := False;

bitBtSortArray.Enabled := IsArrayFilled;

End;

Procedure TuVCLMain.FormCloseQuery(Sender: TObject; Var CanClose: Boolean);

Begin

If Not IsSortButtonPressed Or IsFileSaved Then

Begin

Application.CreateForm(TuVCLExit, uVCLExit);

uVCLExit.ShowModal;

CanClose := uVCLExit.GetStatus();

uVCLExit.Destroy();

End

Else If IsSortButtonPressed Then

Begin

Repeat

ExitCode := MessageBox(uVCLMain.Handle,'Сохранить данные в файл перед

выходом?', 'Подверждение', MB\_ICONQUESTION +

MB\_YESNOCANCEL);

If ExitCode = ID\_YES Then

Begin

BtSaveFileClick(uVCLMain);

CanClose := True;

End

Else If ExitCode = ID\_NO Then

CanClose := True

Else

CanClose := False;

Until IsFileSaved Or (ExitCode = ID\_NO) Or (ExitCode = ID\_CANCEL);

End;

End;

Procedure TuVCLMain.FormCreate(Sender: TObject);

Begin

BufferHandler := TBufferHandler.Create();

End;

Procedure TuVCLMain.FormDestroy(Sender: TObject);

Begin

BufferHandler.Destroy();

End;

Function TuVCLMain.FormHelp(Command: Word; Data: NativeInt;

Var CallHelp: Boolean): Boolean;

Begin

CallHelp := False;

End;

End.

**UnitBackend.pas**  
Unit BackendUnit4\_2;

Interface

Uses System.SysUtils;

Type

TArray = Array Of Integer;

TArraySorter = Class

Private

UserArray: TArray;

Size: Integer;

Procedure Swap(Var First, Second: Integer);

Function Partition(Var Arr: TArray; First, Last: Integer): Integer;

Public

Constructor Create(Size: Integer);

Procedure SetElementByIndex(Element, I: Integer);

Function GetSize(): Integer;

Procedure QuickSort(Var Arr: TArray; First, Last: Integer);

Function GetUserArray(): TArray;

Function GetArray(): TArray;

End;

TFileReader = Class

Private

FileName: String;

InFile: TextFile;

Status: Boolean;

Function IsFileTxt(): Boolean;

Function IsFileReadable(): Boolean;

Public

Function GetStatus(): Boolean;

Procedure SetFileName(FileName: String);

Function InputSize(): Integer;

Function InputArray(Const SIZE: Integer): TArray;

Function IsFileGood(): Boolean;

End;

TFileWriter = Class

Private

OutFile: TextFile;

FileName: String;

Status: Boolean;

Function IsFileTxt(): Boolean;

Function IsFileWritable(): Boolean;

Public

Procedure SetFileName(FileName: String);

Function GetStatus(): Boolean;

Procedure OutputArray(Arr: TArray; Const SIZE: Integer);

Function IsFileGood(): Boolean;

End;

TBufferHandler = Class

Private

FStatus: Boolean;

FEditText: String;

Function GetStatus(): Boolean;

Public

Property Status: Boolean Read GetStatus;

Property EditText: String Write FEditText;

End;

Var

ArraySorter: TArraySorter;

Implementation

{ TArraySorting }

Constructor TArraySorter.Create(Size: Integer);

Begin

Self.Size := Size;

SetLength(UserArray, Size);

End;

Function TArraySorter.GetSize: Integer;

Begin

GetSize := Size;

End;

Function TArraySorter.GetArray: TArray;

Begin

GetArray := UserArray;

End;

Function TArraySorter.GetUserArray: TArray;

Begin

GetUserArray := UserArray;

End;

Function TArraySorter.Partition(Var Arr: TArray; First, Last: Integer): Integer;

Var

Pivot, I, J: Integer;

Begin

Pivot := Arr[Last];

I := First - 1;

For J := First To Last Do

Begin

If Arr[J] < Pivot Then

Begin

Inc(I);

Swap(Arr[I], Arr[J]);

End;

End;

Inc(I);

Swap(Arr[I], Arr[Last]);

Partition := I;

End;

Procedure TArraySorter.QuickSort(Var Arr: TArray; First, Last: Integer);

Var

Pivot: Integer;

Begin

If Last <= First Then

Exit(); // basic case

Pivot := Partition(Arr, First, Last);

QuickSort(Arr, First, Pivot - 1);

QuickSort(Arr, Pivot + 1, Last);

End;

Procedure TArraySorter.SetElementByIndex(Element, I: Integer);

Begin

UserArray[I] := Element;

End;

Procedure TArraySorter.Swap(Var First, Second: Integer);

Var

Temp: Integer;

Begin

Temp := First;

First := Second;

Second := Temp;

End;

{ TFileReader }

Function TFileReader.GetStatus: Boolean;

Begin

GetStatus := Status;

End;

Function TFileReader.IsFileTxt(): Boolean;

Var

FileType: String;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

Function TFileReader.IsFileReadable(): Boolean;

Begin

Try

Reset(InFile);

Status := True;

CloseFile(InFile);

Except

Status := False;

End;

IsFileReadable := Status;

End;

Function TFileReader.IsFileGood(): Boolean;

Begin

Status := False;

If Not FileExists(FileName) Or Not IsFileTxt() Or Not IsFileReadable() Then

Status := False

Else

Status := True;

IsFileGood := Status;

End;

Procedure TFileReader.SetFileName(FileName: String);

Begin

Self.FileName := FileName;

Assign(InFile, Self.FileName);

End;

Function TFileReader.InputSize: Integer;

Const

MIN\_SIZE: Integer = 1;

MAX\_SIZE: Integer = 99;

Var

Size: Integer;

Begin

Size := 0;

Reset(InFile);

Try

Read(InFile, Size);

Except

Status := False;

End;

If (Size < MIN\_SIZE) Or (MAX\_SIZE < Size) Then

Begin

Status := False;

End;

CloseFile(InFile);

InputSize := Size;

End;

Function TFileReader.InputArray(Const SIZE: Integer): TArray;

Const

MIN\_NUMBER: Integer = -99999;

MAX\_NUMBER: Integer = 99999;

Var

IsCorrect: Boolean;

Arr: TArray;

TempNumber, I: Integer;

Begin

SetLength(Arr, SIZE);

IsCorrect := False;

Reset(InFile);

Read(InFile, Arr[0]);

For I := 1 To SIZE Do

Begin

If Status Then

Begin

Try

Read(InFile, TempNumber);

Except

Status := False;

End;

If Status And ((TempNumber < MIN\_NUMBER) Or

(MAX\_NUMBER < TempNumber)) Then

Status := False

Else If Status Then

Arr[I - 1] := TempNumber;

End;

End;

CloseFile(InFile);

InputArray := Arr;

End;

{ TFileWriter }

Function TFileWriter.GetStatus: Boolean;

Begin

GetStatus := Status;

End;

Function TFileWriter.IsFileTxt(): Boolean;

Var

FileType: String;

Begin

FileType := FileName.Substring(FileName.Length - 4);

If FileType = '.txt' Then

Status := True

Else

Status := False;

IsFileTxt := Status;

End;

Function TFileWriter.IsFileWritable(): Boolean;

Begin

Try

Rewrite(OutFile);

Status := True;

CloseFile(OutFile);

Except

Status := False;

End;

IsFileWritable := Status;

End;

Function TFileWriter.IsFileGood(): Boolean;

Begin

If Not FileExists(FileName) And Not IsFileTxt() Or Not IsFileWritable() Then

Status := False

Else

Status := True;

IsFileGood := Status;

End;

Procedure TFileWriter.SetFileName(FileName: String);

Begin

Self.FileName := FileName;

Assign(OutFile, Self.FileName);

End;

Procedure TFileWriter.OutputArray(Arr: TArray; Const SIZE: Integer);

Var

I: Integer;

Begin

Try

Rewrite(OutFile);

Writeln(OutFile, 'Изменённый массив:');

For I := 1 To SIZE Do

Write(OutFile, Arr[I - 1], ' ');

Writeln(OutFile);

Status := True;

Except

Status := False;

End;

CloseFile(OutFile);

End;

{ TBufferHandler }

Function TBufferHandler.GetStatus(): Boolean;

Const

GOOD\_KEYS: Set Of Char = ['0' .. '9'];

Var

I: Integer;

Begin

FStatus := True;

For I := Low(FEditText) To High(FEditText) Do

If FStatus And Not(FEditText[I] In GOOD\_KEYS) Then

FStatus := False;

GetStatus := FStatus;

End;

End.

**Код программы СSharp:**

**ErrorMesagges.cs**

enum ErrorMesagges  
{  
 EmWrongType,  
 EmWrongBoundOfNumber,  
 EmWrongWorkerId,  
 EmWrongKey,  
 EmEmptyTable,  
 EmCompanyNotFound  
}

**FileStatus.cs**  
  
public enum FileStatus  
{  
 FsGood,  
 FsNotFound,  
 FsNotTxt,  
 FsNotReadable,  
 FsNotWritable,  
 FsUnexpacted  
}

**ConsoleReader.cs**

public class ConsoleReader  
{  
 public int InputSizeOfArr()  
 {  
 int size;  
 bool isBad = true;  
 do  
 {  
 Console.WriteLine("Введите размер массива: ");  
 if (int.TryParse(Console.ReadLine(), out size))  
 isBad = false;  
 else  
 Console.WriteLine(MainMenu.ErrorMessagesArr

[(int)ErrorMessages.EmWrongType]);  
 if (!isBad && size < 1)  
 {  
 Console.WriteLine(MainMenu.ErrorMessagesArr

[(int)ErrorMessages.EmWrongBoundOfNumber]);  
 isBad = true;  
 }  
 } while (isBad);  
 return size;  
 }  
  
 public int[] InputElementsOfArr( int size)  
 {  
 int[] arr = new int[size];  
 bool isBad = true;  
 int number;  
 for (int i = 0; i < size; i++)  
 {  
 isBad = true;  
 do  
 {  
 Console.Write("Введите " + (i + 1) + " элемент: ");  
 if (int.TryParse(Console.ReadLine(), out number))  
 isBad = false;  
 else  
 Console.WriteLine(MainMenu.

ErrorMessagesArr[(int)ErrorMessages.EmWrongType]);  
 } while (isBad);  
 arr[i] = number;  
 }  
 return arr;  
 }  
}

**ConsoleWriter.cs**

using System.Drawing;  
  
public class ConsoleWriter  
{  
 public void Output(int[] arr)  
 {  
 Console.WriteLine("Отсортированный массив:");  
 for (int i = 0; i < arr.Length; i++)  
 {  
 Console.Write(arr[i] + " ");  
 }  
 Console.WriteLine();  
 }  
}

**FileReader.cs**

using System.Drawing;  
  
namespace ConsoleApp1;  
  
public class FileReader  
{  
 // constructors   
 public FileReader()  
 {  
 filePath = null;  
 }  
  
 public FileReader(String? filePath)  
 {  
 this.filePath = filePath;  
 }  
  
 // filePath field  
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set   
 {   
 filePath = value;  
 fileStatus = FileStatus.FsGood;  
 }  
  
 }  
  
 // fileStatus field  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 // methods of object  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 return fileStatus;  
 }  
  
 public int InputSizeOfArr()  
 {  
 int size = 0;  
 String? line = null;  
 try  
 {  
 using (StreamReader reader = new StreamReader(filePath))  
 {  
 try  
 {  
 line = reader.ReadLine();  
 if (line == null)  
 fileStatus = FileStatus.FsEmpty;  
 else if (!int.TryParse(line, out size))  
 fileStatus = FileStatus.FsWrongDataType;  
 else if (size < 1)  
 fileStatus = FileStatus.FsWrongDataType;  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotReadable;  
 }  
  
 return size;  
 }  
  
 public int[] InputElementsOfArr(int size)  
 {  
 String? line = null;  
 String[] strElements = null;  
 char splitter = ' ';  
 int[] arr = new int[size];  
 try  
 {  
 using (StreamReader reader = new StreamReader(filePath))  
 {  
 try  
 {  
 reader.ReadLine(); // skip reading size  
 line = reader.ReadLine();  
 strElements = line.Split(splitter);  
 }  
 catch (IOException e)  
 {  
 fileStatus = FileStatus.FsUnexpacted;  
 }  
 }  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotReadable;  
 }  
 if (size != strElements.Length)  
 {  
 fileStatus = FileStatus.FsWrongCount;  
 }  
 for (int i = 0; i < size; i++)  
 {  
 if ((fileStatus == FileStatus.FsGood) && !int.TryParse(strElements[i],

out arr[i]))  
 fileStatus = FileStatus.FsWrongDataType;  
 }  
 return arr;  
  
 }  
}

**FileWriter.cs**

namespace ConsoleApp1;  
  
public class FileWriter  
{  
 // constructors   
 public FileWriter()  
 {  
 filePath = null;  
 }  
  
 public FileWriter(String? filePath)  
 {  
 this.filePath = filePath;  
 }  
  
 // filePath field  
 private string? filePath;  
  
 public string? FilePath  
 {  
 get => filePath;  
 set => filePath = value;  
 }  
  
 // fileStatus field  
 private FileStatus fileStatus;  
  
 public FileStatus FileStatus  
 {  
 get => CheckFileStatus();  
 private set => fileStatus = value;  
 }  
  
 // methods of object  
 private FileStatus CheckFileStatus()  
 {  
 FileInfo fileInfo = new FileInfo(filePath);  
 if (!fileInfo.Exists)  
 {  
 fileStatus = FileStatus.FsNotFound;  
 }  
 else if (!filePath.EndsWith(".txt"))  
 {  
 fileStatus = FileStatus.FsNotTxt;  
 }  
 return fileStatus;  
 }  
  
 public void Output(int[] arr)  
 {  
 try  
 {  
 using StreamWriter writer = new StreamWriter(filePath);  
 writer.WriteLine("Отсортированный массив:");  
 for (int i = 0; i < arr.Length; i++)  
 {  
 writer.Write(arr[i] + " ");  
 }  
  
 writer.WriteLine();  
 }  
 catch (UnauthorizedAccessException e)  
 {  
 fileStatus = FileStatus.FsNotWritable;  
 }  
 }  
}

**Program.cs**

MainMenu mainMenu = new MainMenu();  
  
int choose = 0;  
int size = 0;  
int[] arr = null;  
  
mainMenu.ShowProgramInfo();  
// input block  
do  
{  
 mainMenu.InputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleReader consoleReader = new ConsoleReader();  
 arr = consoleReader.InputElementsOfArr(consoleReader.InputSizeOfArr());  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileReader fileReader = new FileReader();  
 do  
 {  
 fileReader.FilePath = mainMenu.InputFilePath();  
 // reading size and elements and checking status every operation  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 size = fileReader.InputSizeOfArr();  
 }  
 if (fileReader.FileStatus == FileStatus.FsGood)  
 {  
 arr = fileReader.InputElementsOfArr(size);  
 }  
 mainMenu.ShowFileStatusMessage(fileReader.FileStatus);  
 } while (fileReader.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < (int)WorkingTypes.WtConsole || (int)WorkingTypes.WtFile < choose);  
  
// main block  
QuickSort(ref arr,0,arr.Length - 1);  
  
// output block  
do  
{  
 mainMenu.OutputShowMenu();  
 choose = mainMenu.InputChoice();  
 // initialing reader  
 switch ((WorkingTypes)choose)  
 {  
 case WorkingTypes.WtConsole:  
 {  
 ConsoleWriter consoleWriter = new ConsoleWriter();  
 consoleWriter.Output(arr);  
 }  
 break;  
 case WorkingTypes.WtFile:  
 {  
 FileWriter fileWriter = new FileWriter();  
 do  
 {  
 fileWriter.FilePath = mainMenu.InputFilePath();  
 if (fileWriter.FileStatus == FileStatus.FsGood)  
 fileWriter.Output(arr);  
 mainMenu.ShowFileStatusMessage(fileWriter.FileStatus);  
 } while (fileWriter.FileStatus != FileStatus.FsGood);  
 }  
 break;  
 default:  
 {  
 mainMenu.ShowWrongKeyMessage();  
 choose = 0;  
 }  
 break;  
 }  
} while (choose < 0 || (int)WorkingTypes.WtFile < choose);

**MainMenu.cs**  
  
public class MainMenu  
{  
  
 public static readonly String[] ErrorMessagesArr =  
 {  
 "Выбран не существующий пункт! Повторите еще раз.",  
 "Не верный тип данных! Повторите ещё раз.",  
 "Число не может быть меньше 1! Повторите ещё раз.",  
 };  
  
 private static readonly String[] FileStatusMessages =  
 {  
 "Операция прошла успешно!",  
 "Файл не найден! Повторите ещё раз.",  
 "Файл не текстовый! Повторите ещё раз.",  
 "Файл закрыт для чтения! Повторите ещё раз.",  
 "Файл закрыт для записи! Повторите ещё раз.",  
 "Файл пустой. Повторите ещё раз.",  
 "Информация в файле неправильная! Повторите ещё раз.",  
 "Количество элементов не совпадает с указанным размером массива!

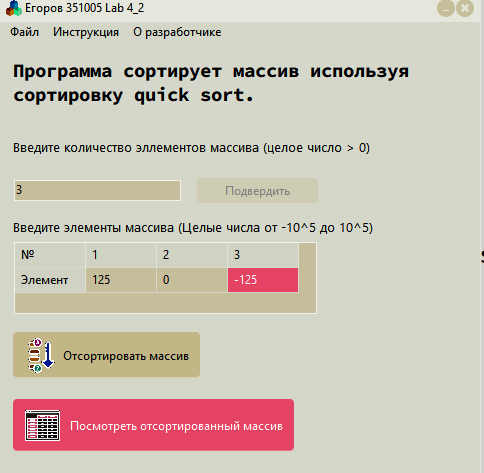
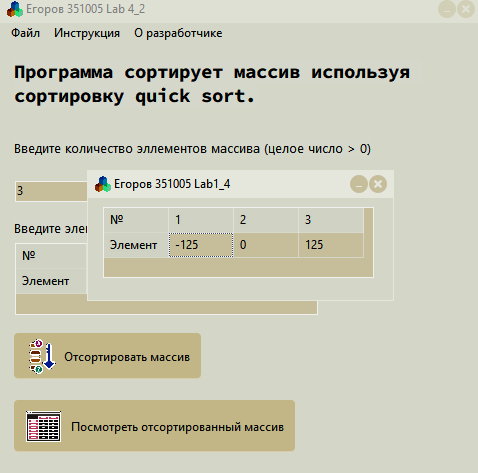
Потворите еще раз.",  
 "Упс... Что-то пошло не так!"  
 };  
  
 public void ShowProgramInfo()  
 {  
 Console.WriteLine("Программа сортирует массив, использую алгоритм quick

sort.");  
 }  
  
 public int InputChoice()  
 {  
 int choose = 0;  
 bool isBad = true;  
 do  
 {  
 Console.Write("Ваш выбор: ");  
 if (int.TryParse(Console.ReadLine(), out choose))  
 isBad = false;  
 else  
 Console.WriteLine(ErrorMessagesArr[(int)ErrorMessages.EmWrongType]);  
 } while (isBad);  
  
 return choose;  
  
 }  
  
 public void ShowWrongKeyMessage()  
 {  
 Console.WriteLine(ErrorMessagesArr[(int)ErrorMessages.EmWrongKey]);  
 }  
  
 public string? InputFilePath()  
 {  
 Console.WriteLine("Введите путь к файлу:");  
 return Console.ReadLine();  
 }  
  
 public void ShowFileStatusMessage(FileStatus fileStatus)  
 {  
 Console.WriteLine(FileStatusMessages[(int)fileStatus]);  
 }  
  
 public void InputShowMenu()  
 {  
 Console.WriteLine("Меню ввода:\n" +  
 "1. Ввести значения с консоли;\n" +  
 "2. Выгрузить значения из файла.");  
 }  
  
 public void OutputShowMenu()  
 {  
 Console.WriteLine("Меню вывода:\n" +  
 "1. Вывести отсортированный массив в консоль;\n" +  
 "2. Записать отсортированный массив в файл.");  
 }  
}

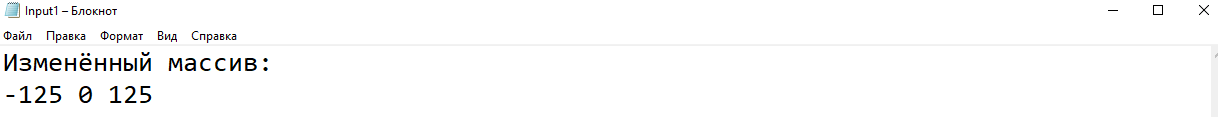
**Sorter.cs**  
  
public class Sorter  
{  
 private protected int Partition(int[] arr, int start, int end)  
 {  
 int pivot = arr[end];  
 int i = start - 1;  
 for (int j = start; j < end; j++)  
 {  
 if (arr[j] < pivot)  
 {  
 i++;  
 (arr[i], arr[j]) = (arr[j], arr[i]);  
   
 }  
 }  
  
 i++;  
 (arr[i], arr[end]) = (arr[end], arr[i]);  
 return i;  
 }  
  
 public void QuickSort(ref int[] arr, int start, int end)  
 {  
 if (end <= start) return;  
 int pivot = Partition(arr, start, end);  
 QuickSort(ref arr,start,pivot - 1);  
 QuickSort(ref arr,pivot + 1,end);  
 }  
  
}

**Скриншоты:**

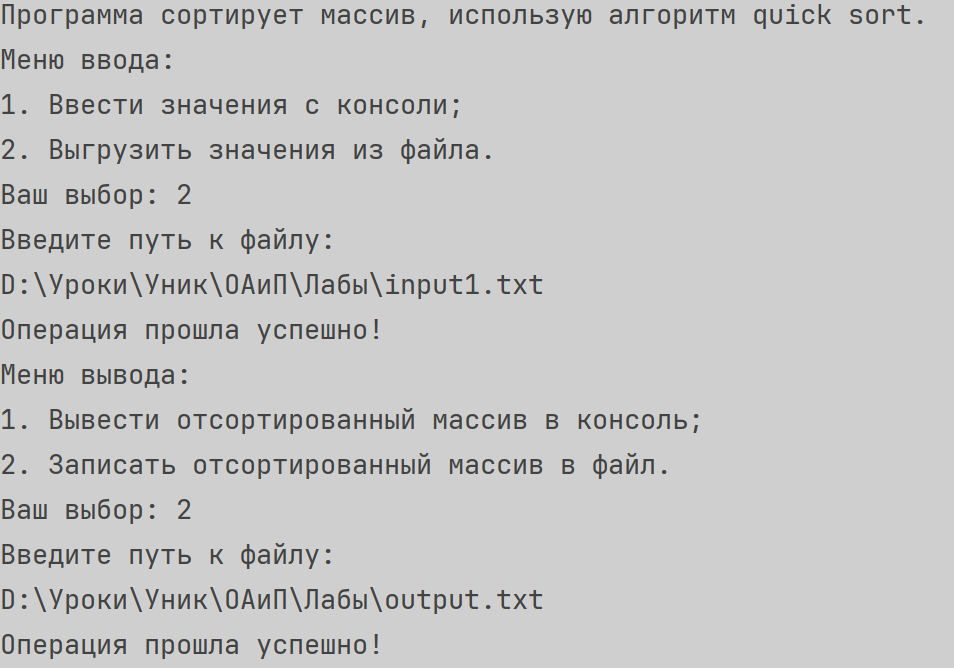
**Delphi:**



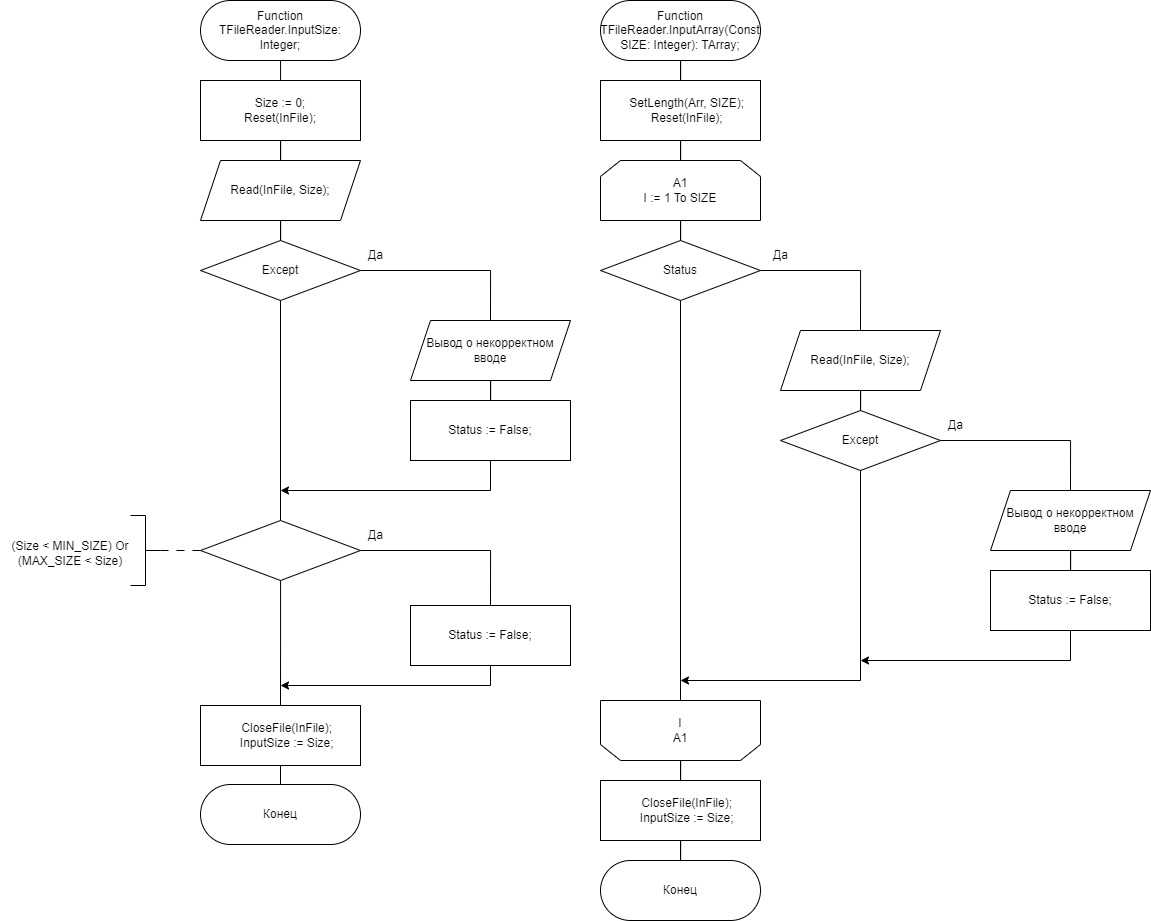
**СSharp:**

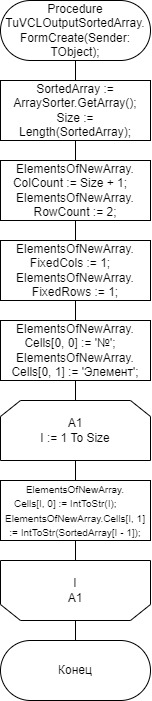
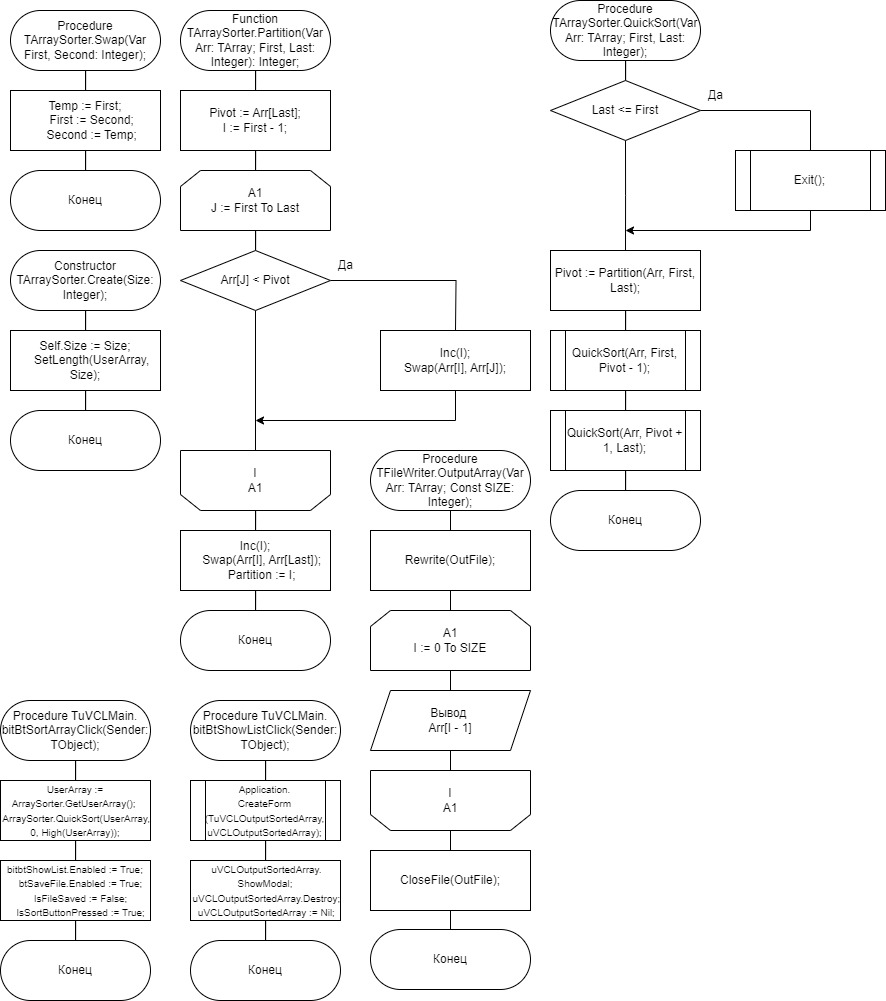


**CSharp:**



**Блок-схема:**

****



**UnitTest CSharp**

**Код**

using ConsoleApp1;  
using JetBrains.Annotations;  
using Microsoft.VisualStudio.TestTools.UnitTesting;  
  
namespace Lab\_4\_2.Tests;  
  
[TestClass]  
[TestSubject(typeof(FileReader))]  
public class FileReaderTests  
{  
 private static FileReader fileReader = new FileReader();   
  
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyWriteFile\_FsNotReadable()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyWriteFile.txt";  
 fileReader.InputSizeOfArr();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotReadable);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongSizeFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongSizeFile.txt";  
 fileReader.InputSizeOfArr();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongCountFile\_FsWrongCount()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongCountFile.txt";  
 fileReader.InputElementsOfArr(fileReader.InputSizeOfArr());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongCount);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsWrongDataType()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\WrongTypeFile.txt";  
 fileReader.InputElementsOfArr(fileReader.InputSizeOfArr());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsWrongDataType);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenEmptyFile\_FsEmpty()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\EmptyFile.txt";  
 fileReader.InputSizeOfArr();  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsEmpty);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsGood()  
 {  
 fileReader.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFile.txt";  
 fileReader.InputElementsOfArr(fileReader.InputSizeOfArr());  
 Assert.AreEqual(fileReader.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(FileWriter))]  
public class FileWriterTexts  
{  
 private static FileWriter fileWriter = new FileWriter();  
   
 [TestMethod]  
 public void CheckFileStatus\_givenOnlyReadFile\_FsNotWritable()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\onlyReadFile.txt";  
 int[] arr = [1, 2, 3, 4];  
 fileWriter.Output(arr);  
 Assert.AreEqual( fileWriter.FileStatus,FileStatus.FsNotWritable);  
 }   
   
 [TestMethod]  
 public void CheckFileStatus\_givenMp3File\_FsNotTxt()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\music.mp3";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotTxt);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenUnexistenFile\_FsNotFound()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\NotExist.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsNotFound);  
 }  
   
 [TestMethod]  
 public void CheckFileStatus\_givenWrongElementFile\_FsGood()  
 {  
 fileWriter.FilePath = "D:\\Уроки\\Уник\\ОАиП\\Лабы\\GoodFile.txt";  
 Assert.AreEqual(fileWriter.FileStatus,FileStatus.FsGood);  
 }  
}  
  
[TestClass]  
[TestSubject(typeof(Sorter))]  
public class QuickSortTest  
{  
 private static Sorter sorter = new Sorter();   
 [TestMethod]  
 public void QuickSort\_SortedArray\_TheSameArray()  
 {  
 int[] testArray = [-1,0,121,224,636,800,12121];  
 sorter.QuickSort(ref testArray,0,testArray.Length - 1);  
 int[] answer = [-1,0,121,224,636,800,12121];  
 CollectionAssert.AreEquivalent(testArray,answer);  
 }  
   
 [TestMethod]  
 public void QuickSort\_UnSortedArrayOfUnsignedNumbers\_SortedArray()  
 {  
 int[] testArray = [12121,10,121,0,224,800,636];  
 sorter.QuickSort(ref testArray,0,testArray.Length - 1);  
 int[] answer = [0,10,121,224,636,800,12121];  
 CollectionAssert.AreEquivalent(testArray,answer);  
 }   
   
 [TestMethod]  
 public void QuickSort\_UnSortedArrayOfAllNumbers\_SortedArray()  
 {  
 int[] testArray = [12121,-21512,121,0,-2552,224,800,636];  
 sorter.QuickSort(ref testArray,0,testArray.Length - 1);  
 int[] answer = [-21512,-2552,0,121,224,636,800,12121];  
 CollectionAssert.AreEquivalent(testArray,answer);  
 }   
   
 [TestMethod]  
 public void QuickSort\_UnSortedArrayOfUnsignedNumbersWithDuplicates\_SortedArray()  
 {  
 int[] testArray = [800,12121,10,121,0,224,800,10,636];  
 sorter.QuickSort(ref testArray,0,testArray.Length - 1);  
 int[] answer = [0,10,10,121,224,636,800,800,12121];  
 CollectionAssert.AreEquivalent(testArray,answer);  
 }   
   
 [TestMethod]  
 public void QuickSort\_UnSortedArrayOfAllNumbersWithDuplicates\_SortedArray()  
 {  
 int[] testArray = [0,12121,-21512,121,0,-2552,224,800,-21512,636];  
 sorter.QuickSort(ref testArray,0,testArray.Length - 1);  
 int[] answer = [-21512,-21512,-2552,0,0,121,224,636,800,12121];  
 CollectionAssert.AreEquivalent(answer,testArray);  
 }   
   
}

**Скриншот**

