Федеральное государственное автономное

образовательное учреждение

высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5**

«Алгоритмы сортировки последовательностей»

тема

Преподаватель \_\_\_\_\_\_\_\_\_ Р.Ю. Царев

подпись, дата инициалы, фамилия

Студент КИ16-16б, №031618575 \_\_\_\_\_\_\_\_\_ В.В. Макеев

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

**СОДЕРЖАНИЕ**

1 Цель работы…………………………………………………………...3

2 Задачи …………………………………………………………………3

3 Ход работы……………………………………………………………3

4 Вывод…………………………………………...……………………..4

ПРИЛОЖЕНИЕ А ……………………………………………….……..5

ПРИЛОЖЕНИЕ Б…………………………………………………..…..9

ПРИЛОЖЕНИЕ В……………………………………………………..12

**1 Цель работы**

# Изучение алгоритмов обработки последовательностей.

# 2 Задание

Реализовать в программе один из следующих алгоритмов: простого слияния, естественного слияния, многофазного слияния, многопутевого слияния.

**3 Ход работы**

С использованием языка программирования C# было создано графическое приложения, осуществляющее сортировку элементов последовательности файла с помощью алгоритма простого слияния. Суть его работы проста: сначала серии последовательности, длина которых определяется значением двойки, возведенной в степень, равной номеру итерации цикла, записываются в два файла. Затем происходит «слияние» пар серий, взятых из двух данных файлов, в порядке возрастания значений их элементов в первоначальный файл. Разделение последовательности на серии и их слияние продолжается до тех пор, пока длина серии не превысит половины количества всех элементов или пока они не будут упорядочены. Запись промежуточных результатов алгоритма и запросов на сортировку происходит в базу данных. Примеры работы программы приведены на рисунках 1-3.

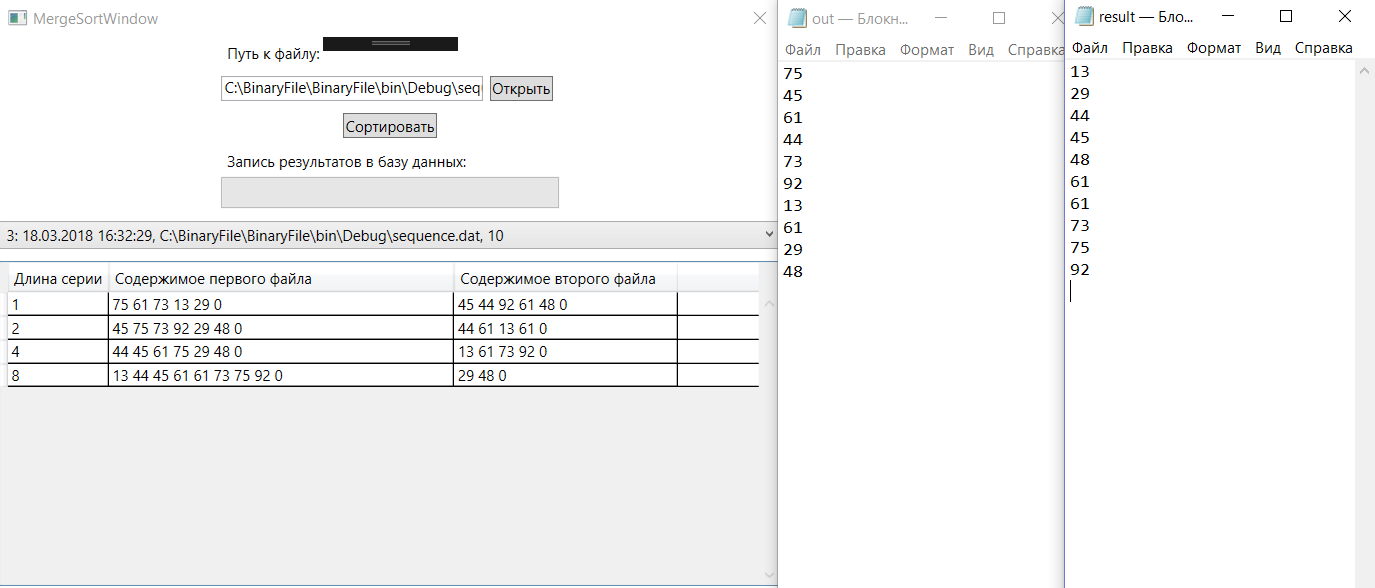


Рисунок 1 – Пример работы программы 1

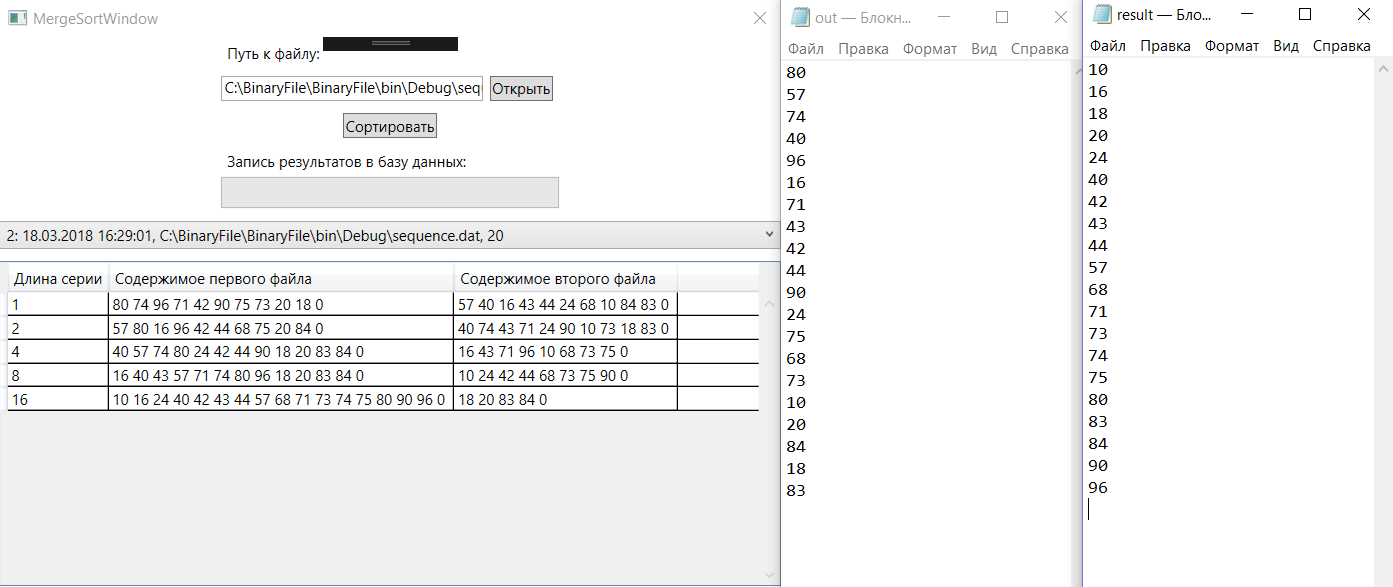


Рисунок 2 – Пример работы программы 2

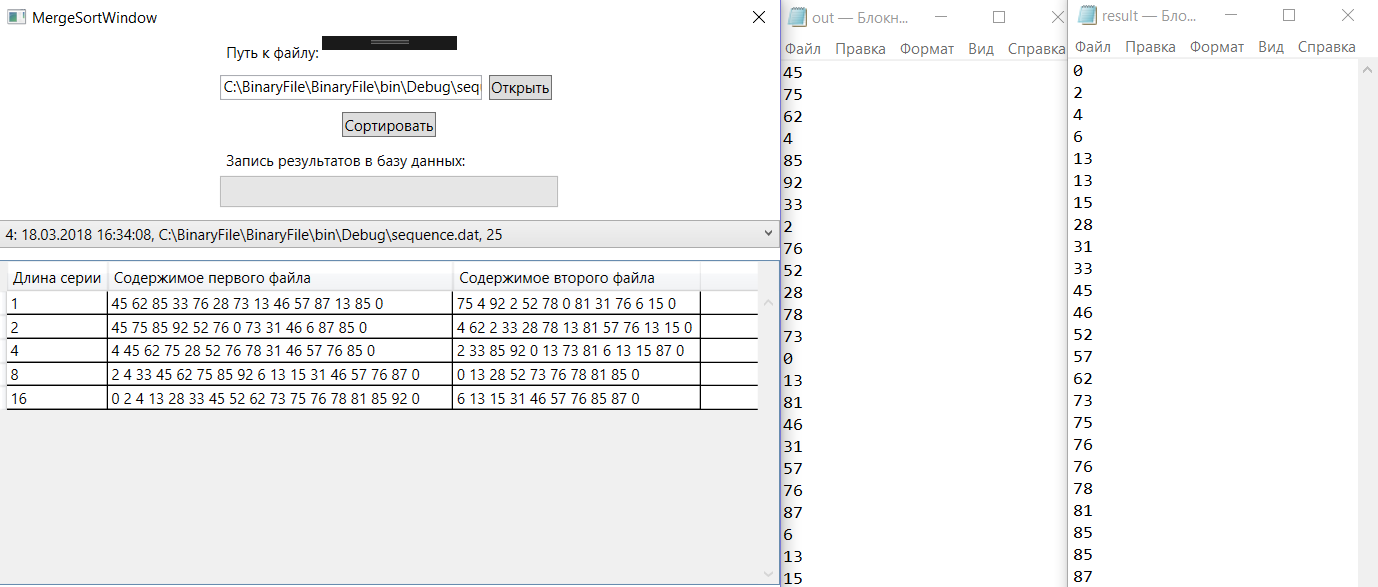


Рисунок 3 – Пример работы программы 3

# 4 Вывод

В ходе выполнения лабораторной работы был приобретен опыт реализации алгоритмов внешней сортировки на примере создания программы, сортирующей элементы файла с помощью простого слияния. Закреплены навыки программирования на языке C#, работы с платформой .NET Framework, создания пользовательского интерфейса, подключения к программе базы данных и её использования.

**ПРИЛОЖЕНИЕ А**

**Листинг исходного кода графической части программы**

MergeSortWindow.xaml:

|  |
| --- |
| <Window x:Class="LaboratoryWorks.MergeSortWindow"  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"  xmlns:d="http://schemas.microsoft.com/expression/blend/2008"  xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"  xmlns:local="clr-namespace:LaboratoryWorks"  mc:Ignorable="d"  Title="MergeSortWindow" Height="480" Width="640"  ResizeMode="NoResize">  <Grid>  <Grid.RowDefinitions>  <RowDefinition Height="1\*"/>  <RowDefinition Height="2\*"/>  </Grid.RowDefinitions>  <StackPanel HorizontalAlignment="Center">  <Label Content="Путь к файлу:"/>  <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">  <TextBox x:Name="TBPath" Height="20" Width="210"/>  <Button Content="Открыть" Height="20" Margin="5" Click="BOpenFile\_Click"/>  </StackPanel>  <Button x:Name="BSort" Content="Сортировать" HorizontalAlignment="Center" Height="20" Margin="5" Click="BSort\_Click"/>  <Label Content="Запись результатов в базу данных:"/>  <ProgressBar x:Name="PBDatabase" Height="25"/>  </StackPanel>  <StackPanel Grid.Row="1">  <ComboBox x:Name="CBQuery" Margin="0 0 0 10" SelectionChanged="CBQuery\_SelectionChanged"/>  <DataGrid x:Name="DGIteration" IsReadOnly="True" Height="260" VerticalScrollBarVisibility="Visible">  <DataGrid.Columns>  <DataGridTextColumn Header="Длина серии" Binding="{Binding Path=Length}"/>  <DataGridTextColumn Header="Содержимое первого файла" Binding="{Binding Path=First}"/>  <DataGridTextColumn Header="Содержимое второго файла" Binding="{Binding Path=Second}"/>  </DataGrid.Columns>  </DataGrid>  </StackPanel>  </Grid>  </Window> |

MergeSortWindow.xaml.cs:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Diagnostics;  using System.IO;  using System.Linq;  using System.Threading.Tasks;  using System.Windows;  using System.Windows.Controls;  using Microsoft.Win32;  namespace LaboratoryWorks  {  /// <summary>  /// Класс окна внешней сортировки  /// </summary>  public partial class MergeSortWindow : Window  {  /// <summary>  /// Класс, содержащий свойства для записи информации об итерации и её вывода в таблицу  /// </summary>  private class Iteration  {  public string Length { get; set; }  public string First { get; set; }  public string Second { get; set; }  }  /// <summary>  /// Конструктор класса: инициализация графических элементов  /// </summary>  public MergeSortWindow()  {  InitializeComponent();  IEnumerable<MergeSortQuery> list = DBWork.GetQueryList();  if (list != null)  foreach (MergeSortQuery i in list)  AddQuery(i);  }  /// <summary>  /// Обработчик нажатия кнопки "Открыть"  /// </summary>  /// <param name="sender">Объект - источник события</param>  /// <param name="e">Данные события</param>  private void BOpenFile\_Click(object sender, RoutedEventArgs e)  {  OpenFileDialog file = new OpenFileDialog();  if (file.ShowDialog() == true) TBPath.Text = file.FileName;  }    /// <summary>  /// Обработчик нажатия кнопки "Сортировать"  /// </summary>  /// <param name="sender">Объект - источник события</param>  /// <param name="e">Данные события</param>  private void BSort\_Click(object sender, RoutedEventArgs e)  {  if (!File.Exists(TBPath.Text)) MessageBox.Show("Не удалось найти файл по указанному пути");  else  {  MergeSortQuery query = new MergeSortQuery() { Stamp = DateTime.Now, Path = TBPath.Text, Count = (new FileInfo(TBPath.Text)).Length / sizeof(double) };  Stopwatch sw = new Stopwatch();  sw.Start();  Sort.MergeSort(TBPath.Text);  sw.Stop();  query.Time = sw.ElapsedTicks;  DBWork.Add(query);  MessageBox.Show("Элементы файла упорядочены");  int max = 0;  for (int i = 1; Directory.Exists($"Serial{i}"); i <<= 1)  max++;  if (max != 0)  {  BSort.IsEnabled = false;  PBDatabase.Maximum = max;  PBDatabase.Value = 0;  }  else MessageBox.Show("Результаты сортировки успешно записаны в базу данных");  for (int i = 0; i < max; i++)  {  int number = (int)Math.Pow(2, i);  Task write = new Task(() =>  {  DBWork.Add(new MergeSortIteration() { SortNum = query.SortNum, SerialLength = number, FirstFile = File.ReadAllBytes($"Serial{number}/first.dat"), SecondFile = File.ReadAllBytes($"Serial{number}/second.dat") });  PBDatabase.Dispatcher.Invoke(() =>  {  PBDatabase.Value++;  if (PBDatabase.Value == PBDatabase.Maximum)  {  BSort.Dispatcher.Invoke(() => BSort.IsEnabled = true);  MessageBox.Show("Результаты сортировки успешно записаны в базу данных");  PBDatabase.Value = 0;  }  });  });  write.Start();  }  AddQuery(query);  }  }  /// <summary>  /// Обработчик события выбора элемента в списке запросов на сортировку  /// </summary>  /// <param name="sender">Объект - источник события</param>  /// <param name="e">Данные события</param>  private void CBQuery\_SelectionChanged(object sender, SelectionChangedEventArgs e)  {  DGIteration.Items.Clear();  if (CBQuery.SelectedItem != null)  {  IEnumerable<MergeSortIteration> list = (DBWork.GetIterationList(int.Parse(((string)CBQuery.SelectedItem).Split(':')[0]))).ToArray();  if (list != null)  foreach (MergeSortIteration i in list)  {  Iteration sprint = new Iteration() { Length = i.SerialLength.ToString(), First = DoubleToStr(i.FirstFile), Second = DoubleToStr(i.SecondFile) };  DGIteration.Items.Add(sprint);  }  }  }  /// <summary>  /// Получение чисел с плавающей точкой из массива байт и их запись в строку  /// </summary>  /// <param name="array">Исходный массив байт</param>  /// <returns>Строку, содержащую последовательность вещественных чисел</returns>  private string DoubleToStr(byte[] array)  {  string text = string.Empty;  int size = sizeof(double);  for (int i = 0; i < array.Length / size; i++)  {  byte[] number = new byte[size];  Array.Copy(array, i \* size, number, 0, size);  text += $"{BitConverter.ToDouble(number, 0)} ";  }  return text;  }  /// <summary>  /// Добавление объекта в список запросов на сортировку  /// </summary>  /// <param name="item">Запрос на сортировку</param>  private void AddQuery(MergeSortQuery item)  {  CBQuery.Items.Add($"{item.SortNum}: {item.Stamp}, {item.Path}, {item.Count}");  }  }  } |

**ПРИЛОЖЕНИЕ Б**

**Листинг исходного кода алгоритма сортировки**

Sort.cs:

|  |
| --- |
| using System;  using System.IO;  namespace LaboratoryWorks  {  /// <summary>  /// Класс алгоритма сортировки простым слиянием  /// </summary>  static class Sort  {  /// <summary>  /// Имена файлов для вывода серий  /// </summary>  private static readonly string[] addfile = { "first.dat", "second.dat" };  /// <summary>  /// Предикат, определяющий достижение конца потока чтения  /// </summary>  private static Predicate<BinaryReader> EOF = delegate (BinaryReader x) { return !(x.BaseStream.Position < x.BaseStream.Length); };  /// <summary>  /// Сортировка простым слиянием: вызов методов разделения последовательности и её объединения  /// </summary>  /// <param name="path">Путь до файла, элементы которого необходимо упорядочить</param>  public static void MergeSort(string path)  {  for (int i = 1; Directory.Exists(DirName(i)); i <<= 1)  Directory.Delete(DirName(i), true);  for (int i = 1; !IsSorted(path) && WriteToAddFiles(path, i); i <<= 1)  WriteToMainFile(path, i);  }  /// <summary>  /// Разделение последовательности на серии и их вывод в файлы  /// </summary>  /// <param name="path">Путь до файла, элементы которого необходимо упорядочить</param>  /// <param name="number">Количество элементов в серии</param>  /// <returns></returns>  public static bool WriteToAddFiles(string path, int number)  {  string dir = DirName(number);  Directory.CreateDirectory(dir);  BinaryReader reader = new BinaryReader(File.OpenRead(path));  if (reader.BaseStream.Length < number \* sizeof(double)) return false;  BinaryWriter[] writer = { new BinaryWriter(File.Create($"{dir}/{addfile[0]}")), new BinaryWriter(File.Create($"{dir}/{addfile[1]}")) };  int index = 0;  for (int i = 1; !EOF(reader); i++)  {  writer[index].Write(reader.ReadDouble());  if (i % number == 0) index = 1 - index;  }  for (int i = 0; i < 2; i++)  writer[i].Write((double)0);  writer[0].Close();  writer[1].Close();  reader.Close();  return true;  }  /// <summary>  /// Объединение двух последовательностей в одну и её вывод в файл  /// </summary>  /// <param name="path">Путь до файла, элементы которого необходимо упорядочить</param>  /// <param name="number">Количество элементов в серии</param>  private static void WriteToMainFile(string path, int number)  {  string dir = DirName(number);  BinaryReader[] reader = { new BinaryReader(File.OpenRead($"{dir}/{addfile[0]}")), new BinaryReader(File.OpenRead($"{dir}/{addfile[1]}")) };  BinaryWriter writer = new BinaryWriter(File.Create(path));  double[] item = new double[2];  for (int i = 0; i < item.Length; i++)  if (!(EOF(reader[i]))) item[i] = reader[i].ReadDouble();  while (!EOF(reader[0]) && !EOF(reader[1]))  {  int i = 0;  int j = 0;  while (i < number && j < number && !EOF(reader[0]) && !EOF(reader[1]))  if (item[0] < item[1])  {  Write(writer, reader[0], ref item[0]);  i++;  }  else  {  Write(writer, reader[1], ref item[1]);  j++;  }  while (i < number && !EOF(reader[0]))  {  Write(writer, reader[0], ref item[0]);  i++;  }  while (j < number && !EOF(reader[1]))  {  Write(writer, reader[1], ref item[1]);  j++;  }  }  while (!EOF(reader[0]))  Write(writer, reader[0], ref item[0]);  while (!EOF(reader[1]))  Write(writer, reader[1], ref item[1]);  reader[0].Close();  reader[1].Close();  writer.Close();  }  /// <summary>  /// Запись элемента в поток ввода и чтение следующего числа из потока вывода  /// </summary>  /// <param name="writer">Выходной поток</param>  /// <param name="reader">Входной поток</param>  /// <param name="item">Параметр, значение которого после вывода будет замещено следующим, содержащимся во входном потоке</param>  /// <param name="index">Счетчик</param>  private static void Write(BinaryWriter writer, BinaryReader reader, ref double item)  {  writer.Write(item);  item = reader.ReadDouble();  }  /// <summary>  /// Получение названия директории по длине серии  /// </summary>  /// <param name="length">Длина серии</param>  /// <returns>Имя директории</returns>  private static string DirName(int length)  {  return $"Serial{length}";  }  /// <summary>  /// Проверка упорядоченности последовательности  /// </summary>  /// <param name="path">Путь до файла с последовательностью</param>  /// <returns>Истина, если элементы последовательност отсортированы в порядке неубывания, иначе - ложь</returns>  private static bool IsSorted(string path)  {  BinaryReader reader = new BinaryReader(File.OpenRead(path));  double? prev = null;  bool sorted = true;  while (!EOF(reader))  if (prev == null)  prev = reader.ReadDouble();  else if (prev > (prev = reader.ReadDouble()))  {  sorted = false;  break;  }  reader.Close();  return sorted;  }  }  } |

**ПРИЛОЖЕНИЕ В**

**Листинг исходного кода классов для работы с базой данных**

DBWork.cs:

|  |
| --- |
| using System;  using SQLite;  using System.Collections.Generic;  namespace LaboratoryWorks  {  /// <summary>  /// Работа с базой данных: установка подключения, создание таблиц, выполнение запросов  /// </summary>  static class DBWork  {  /// <summary>  /// Экземпляр класса соединения с базой данных  /// </summary>  private static SQLiteConnection db;  /// <summary>  /// Конструктор класса: подключение к базе данных и создание таблиц  /// </summary>  static DBWork()  {  db = new SQLiteConnection("algorithms.db");  db.CreateTable<MergeSortQuery>();  db.CreateTable<MergeSortIteration>();  }  /// <summary>  /// Запись объекта класса в базу данных  /// </summary>  /// <typeparam name="T">Класс объекта</typeparam>  /// <param name="item">Объект для записи</param>  public static void Add<T>(T item)  {  db.Insert(item);  }  /// <summary>  /// Получение списка запросов на сортировку  /// </summary>  /// <returns>Список запросов</returns>  public static IEnumerable<MergeSortQuery> GetQueryList()  {  return db.Table<MergeSortQuery>();  }  /// <summary>  /// Получение списка итераций по идентификатору запроса на сортировку  /// </summary>  /// <param name="id">Идентификатор запроса</param>  /// <returns>Список, содержащий информацию об итерациях</returns>  public static IEnumerable<MergeSortIteration> GetIterationList(int id)  {  return from i in db.Table<MergeSortIteration>() where (i.SortNum == id) orderby i.SerialLength select i;  }  }  } |

MergeSortQuery.cs:

|  |
| --- |
| using System;  using SQLite;  namespace LaboratoryWorks  {  /// <summary>  /// Класс, представляющий таблицу "queries"  /// </summary>  [Table("queries")]  class MergeSortQuery  {  /// <summary>  /// Номер сортировки  /// </summary>  [PrimaryKey, AutoIncrement, Column("sortnum")]  public int SortNum { get; set; }  /// <summary>  /// Время начала сортировки  /// </summary>  [Column("stamp")]  public DateTime Stamp { get; set; }    /// <summary>  /// Путь до файла с последовательностью  /// </summary>  [Column("path")]  public string Path { get; set; }  /// <summary>  /// Число элементов в последовательности  /// </summary>  [Column("count")]  public long Count { get; set; }  /// <summary>  /// Время сортировки  /// </summary>  [Column("time")]  public long Time { get; set; }  }  } |

MergeSortIteration.cs:

|  |
| --- |
| using System;  using SQLite;  namespace LaboratoryWorks  {  /// <summary>  /// Класс, представляющий таблицу "iterations"  /// </summary>  [Table("iterations")]  class MergeSortIteration  {  /// <summary>  /// Номер сортировки  /// </summary>  [Column("sortnum")]  public int SortNum { get; set; }  /// <summary>  /// Длина серии  /// </summary>  [Column("serlength")]  public int SerialLength { get; set; }  /// <summary>  /// Содержимое файла с первой частью последовательности  /// </summary>  [Column("first")]  public byte[] FirstFile { get; set; }  /// <summary>  /// Содержимое файла со второй частью последовательности  /// </summary>  [Column("second")]  public byte[] SecondFile { get; set; }  }  } |