Федеральное государственное автономное

Образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №5**

Алгоритмы обработки последовательностей

тема

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

подпись, дата инициалы, фамилия

Студент КИ16-17Б, №031620322 \_\_\_\_\_\_\_\_\_ В.С. Мараев

Номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

# Цель работы

# Изучение алгоритмов обработки последовательностей, внешней сортировки массивов методом слияния и их эффективности.

# Задание

Реализовать в программе алгоритм сортировки естественным слиянием. При сортировке последовательностей должны использоваться файлы, но не массивы.

# Теория

**Естественное слияние**

При сортировке простым слиянием данные разбиваются и сливаются в подпоследовательности длина, которых равна степени двойки (2 *k* , где *k* ≥ 0). Однако данные исходной последовательности могут быть уже частично упорядочены. В этом случае целесообразно просто объединить уже упорядоченные подпоследовательности. Две упорядоченные подпоследовательности длиной *m* и *n* , содержащиеся в двух файлах, дадут на фазе слияния одну подпоследо­вательность из *m* + *n* элементов. Сортировка, при которой сливаются упорядоченные подпоследовательности, называется есте­ственным слиянием.

Упорядоченные подпоследовательности на­зывают «сериями».   
В каждой серии элемент *ri* не больше, чем *ri* +1 . Таким образом, в сортировке естественным слиянием объе­диняются серии, а не последователь­ности (заранее) фиксированной длины. Если сли­ваются две последовательности, каждая из *n* серий, то результирующая также содержит ровно *n* серий.   
Следовательно, при каждом проходе общее число серий уменьшается вдвое и общее число пересылок в самом плохом случае равно *n* \* *log* ( *n* ), а в сред­нем даже меньше. Ожидаемое число сравнений, однако, значительно больше, поскольку кроме срав­нений, необходимых для отбора элементов при слия­нии, нужны еще дополнительные сравнения между последова­тельными элементами каждого файла, чтобы опре­делить конец серии.

Алгоритм сортировки естественным слиянием во многом похож на сортировку прямым слиянием. Единственное различие заключается с том, что длина подпоследовательностей в прямом слиянии кратна двойке: 1, 2, 4, 8, 16 и т. д., а в естественном слиянии длина серии вычисляется в процессе считывания элементов последовательности. И, именно, этим реализация алгоритма естественного слияния оказывается сложнее.

Серьезную трудность представляет собой определение конца серии, поскольку для этого необходимо сравнивать значения двух последовательных элементов. Однако, природа последовательности такова, что непосредственно до­ступен только один-единственный элемент. Для того чтобы «заглянуть вперед» можно организовать своеобразный буфер, введя переменную, куда будет считываться текущий (самый левый) элемент последовательности. Таким образом, можно будет сравнить только что считанный элемент, который теперь хранится в «буфере», со следующим за ним элементом последовательности. Если элемент, считанный в буфер, меньше следующего за ним, то элементы принадлежат одной серии. Если же элемент, содержащийся в буфере, оказывается больше последующего элемента, это обозначает конец текущей серии и начало новой.

Если ввести переменную, в которой при слиянии будут подсчитываться серии, то работу алгоритма можно завершить, когда в последовательности *a* будет всего одна серия. Это свидетельствует о том, что исходная последовательность уже отсортирована.

# Ход работы

1. Было разработано настольное Windows-forms приложение, реализующее алгоритм внешней сортировки естественным слиянием (рисунок 1). Пользователь может, как самостоятельно вписывать путь до файла с числами, так и указывать путь с помощью обозревателя (рисунок 2). При запуске программы автоматически указывается стандартный путь до директории программы. После завершения внешней сортировки программа уведомляет пользователя об её успешном или ошибочном окончании, а также укажет время выполнения сортировки (рисунок 3). Если файл изначально заполнен некорректно, программа уведомит пользователя об этом. Предоставлена возможность быстрого тестирования программы, включающее в себя: создания файла в корневой папке программы, заполнение файла N случайными элементами с последующей внешней сортировкой (рисунок 4). Число N задаётся пользователем. Также, обеспечена устойчивость программы при любых воздействиях, задаваемых пользователем.

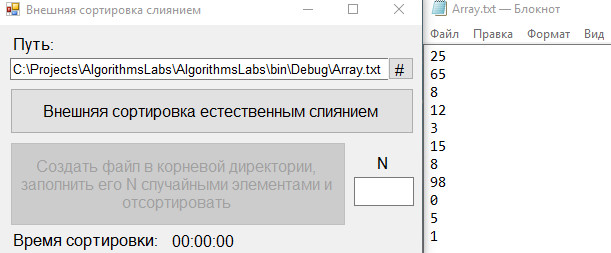


Рисунок 1 – Интерфейс приложения и начальный массив в текстовом файле

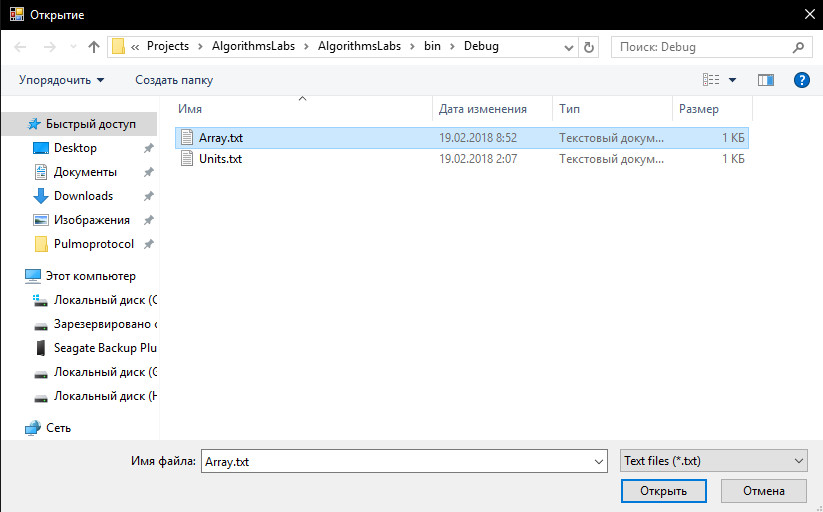


Рисунок 2 – Указание пути до сортируемого файла

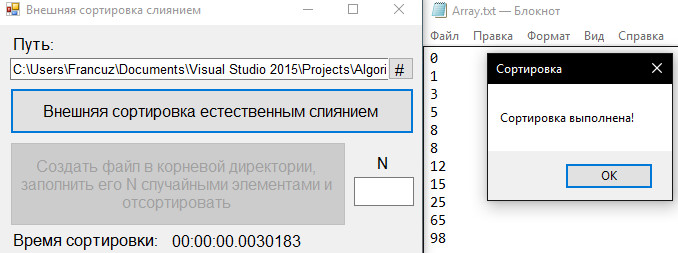


Рисунок 3 – Результат исполнения алгоритма внешней сортировки естественным слиянием

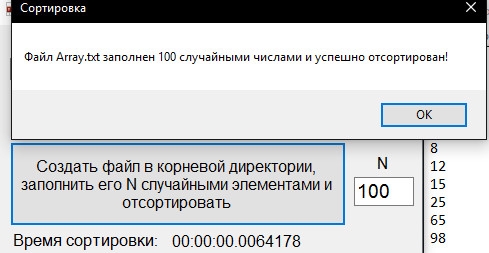


Рисунок 4 – Результат исполнения быстрого тестирования алгоритма внешней сортировки естественным слиянием на ста случайных числах

# Вывод

В ходе практической работы были разработан и отлажен алгоритм внешней сортировки массивов методом естественного слияния. Была исследована его эффективность. Доказано, что внешняя сортировка естественным слиянием даёт возможность сортировать числовые массивы, записанные в файлах, объём которых превышает объём оперативной памяти, предоставленной программе.

# Исходный код программы

Листинг 1 - файл MergeForm.cs

|  |
| --- |
| using System;  using System.Windows.Forms;  namespace AlgorithmsLabs  {  public partial class MergeForm : Form  {  MergeController Controller = new MergeController();  public MergeForm()  {  InitializeComponent();  Controller.SetStartPath(tbPath);  }  private void btnPath\_Click(object sender, EventArgs e)  {  Controller.GetPath(tbPath);  }  private void btnSort\_Click(object sender, EventArgs e)  {  string FileName = tbPath.Text;  lblSortTime.Text = Controller.SortArray(FileName);  }  private void btnRandomSort\_Click(object sender, EventArgs e)  {  lblSortTime.Text = Controller.SortRandomArray(Convert.ToInt32(tbN.Text));  }  private void tbN\_TextChanged(object sender, EventArgs e)  {  btnRandomSort.Enabled = !String.IsNullOrEmpty(tbN.Text);  }  private void NumberValidating(object sender, KeyPressEventArgs e)  {  char l = e.KeyChar;  if ((l < '0' || l > '9') && (l != '\b'))  e.Handled = true;  }  private void MergeForm\_FormClosed(object sender, FormClosedEventArgs e)  {  Application.Exit();  }  }  } |

Листинг 2 - файл MergeController.cs

|  |
| --- |
| using System;  using System.Windows.Forms;  using System.IO;  using System.Diagnostics;  namespace AlgorithmsLabs  {  class MergeController  {  /// <summary>  /// Добавляет стартовый руть к файлу Array.txt  /// </summary>  /// <param name="tbPath">Строка для записи пути</param>  public void SetStartPath(TextBox tbPath)  {  FileInfo fi = new FileInfo("Array.txt");  if (fi.Exists)  tbPath.Text = fi.FullName;  }  /// <summary>  /// Получает путь до файла, открывая пользовательский диалог  /// </summary>  /// <param name="tbPath">Строка для записи пути</param>  public void GetPath(TextBox tbPath)  {  OpenFileDialog ofd = new OpenFileDialog();  ofd.Filter = "Text files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";  if (ofd.ShowDialog() == DialogResult.OK)  tbPath.Text = ofd.FileName;  }  /// <summary>  /// Сортирует массив из файла  /// </summary>  /// <param name="FileName">Путь до сортируемого файла</param>  /// <returns>Время сортировки</returns>  public string SortArray(string FileName)  {  if (new FileInfo(FileName).Exists)  {  Stopwatch timer = new Stopwatch();  timer.Start();  try  {  MergeSort.NaturalMergeSort(FileName);  }  catch  {  MessageBox.Show("Файл заполнен некорректно!", "Ошибка");  return "00:00:00";  }  timer.Stop();  MessageBox.Show("Сортировка выполнена!", "Сортировка");  return timer.Elapsed.ToString();  }  else MessageBox.Show("Некорректный путь!", "Ошибка");  return "00:00:00";  }  /// <summary>  /// Заполняет файл случайными числами и сортирует их  /// </summary>  /// <param name="N">колличество случайных чисел</param>  /// <returns>Время сортировки</returns>  public string SortRandomArray(int N)  {  string FileName = "Array.txt";  StreamWriter SW = new StreamWriter(FileName);  Random rand = new Random();  for (int i = 0; i < N; i++)  SW.WriteLine(rand.Next(0, 100));  SW.Close();  Stopwatch timer = new Stopwatch();  timer.Start();  MergeSort.NaturalMergeSort(FileName);  timer.Stop();  MessageBox.Show("Файл Array.txt заполнен " + N + " случайными числами и успешно отсортирован!", "Сортировка");  return timer.Elapsed.ToString();  }  }  } |

Листинг 3 - файл MergeSort.cs

|  |
| --- |
| using System;  using System.IO;  namespace AlgorithmsLabs  {  static class MergeSort  {  /// <summary>  /// Сортирует числа в файле методом естественного слияния  /// </summary>  /// <param name="FileName">Полный путь к сортируемому файлу</param>  static public void NaturalMergeSort(string FileName)  {  int s1, s2, a1, a2, mark;  StreamReader SR, SR1, SR2;  StreamWriter SW, SW1, SW2;  s1 = 1;  s2 = 1;  a1 = 0;  a2 = 0;  while (s1 > 0 && s2 > 0)  {  mark = 1;  s1 = 0;  s2 = 0;  SR = new StreamReader(FileName);  SW1 = new StreamWriter("f1.txt");  SW2 = new StreamWriter("f2.txt");  if (!SR.EndOfStream)  {  a1 = Convert.ToInt32(SR.ReadLine());  SW1.WriteLine(a1);  }  while (!SR.EndOfStream)  {  a2 = Convert.ToInt32(SR.ReadLine());  if (a2 < a1)  switch (mark)  {  case 1:  SW1.WriteLine('\'');  mark = 2;  s1++;  break;  case 2:  SW2.WriteLine('\'');  mark = 1;  s2++;  break;  }  if (mark == 1)  {  SW1.WriteLine(a2);  s1++;  }  else  {  SW2.WriteLine(a2);  s2++;  }  a1 = a2;  }  if (s2 > 0 && mark == 2)  SW2.WriteLine('\'');  if (s1 > 0 && mark == 1)  SW1.WriteLine('\'');  SR.Close();  SW1.Close();  SW2.Close();  SW = new StreamWriter(FileName);  SR1 = new StreamReader("f1.txt");  SR2 = new StreamReader("f2.txt");  if (!SR1.EndOfStream)  a1 = Convert.ToInt32(SR1.ReadLine());  if (!SR2.EndOfStream)  a2 = Convert.ToInt32(SR2.ReadLine());  bool file1, file2;  while (!SR1.EndOfStream && !SR2.EndOfStream)  {  file1 = false;  file2 = false;  while (!file1 && !file2)  {  if (a1 <= a2)  {  SW.WriteLine(a1);  file1 = EndBlock(SR1);  a1 = Convert.ToInt32(SR1.ReadLine());  }  else  {  SW.WriteLine(a2);  file2 = EndBlock(SR2);  a2 = Convert.ToInt32(SR2.ReadLine());  }  }  while (!file1)  {  SW.WriteLine(a1);  file1 = EndBlock(SR1);  a1 = Convert.ToInt32(SR1.ReadLine());  }  while (!file2)  {  SW.WriteLine(a2);  file2 = EndBlock(SR2);  a2 = Convert.ToInt32(SR2.ReadLine());  }  }  file1 = false;  file2 = false;  while (!file1 && !SR1.EndOfStream)  {  SW.WriteLine(a1);  file1 = EndBlock(SR1);  a1 = Convert.ToInt32(SR1.ReadLine());  }  while (!file2 && !SR2.EndOfStream)  {  SW.WriteLine(a2);  file2 = EndBlock(SR2);  a2 = Convert.ToInt32(SR2.ReadLine());  }  SW.Close();  SR1.Close();  SR2.Close();  }  File.Delete("f1.txt");  File.Delete("f2.txt");  }  /// <summary>  /// Проверяет на конец блока  /// </summary>  /// <param name="SR">Поток чтения файла</param>  /// <returns>Является ли концом блока</returns>  static private bool EndBlock(StreamReader SR)  {  if ((char)SR.Peek() == '\'')  {  SR.ReadLine();  return true;  }  else return false;  }  }  } |