Федеральное государственное автономное

Образовательное учреждение

Высшего образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт Космических и информационных технологий

институт

Кафедра «Информатика»

кафедра

**ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ №4**

Усовершенствованные алгоритмы сортировки массивов

тема

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

подпись, дата инициалы, фамилия

Студент КИ16-16Б, №031620303 \_\_\_\_\_\_\_\_\_ В.В. Кокташев

Номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2018

# Цель работы

# Изучение усовершенствованных алгоритмов сортировки массивов и сравнения их эффективности по времени выполнения, числу перестановок и количеству сравнений.

# Задание

1. Реализовать в программе алгоритм шейкерной сортировки.
2. Реализовать в программе алгоритм быстрой сортировки.
3. Сравнить эффективность реализованных алгоритмов по времени выполнения**,** числу перестановок и количеству сравнений.

# Теория

**Шейкерная сортировка**

Сортировка перемешиванием, или Шейкерная сортировка, или двунаправленная — разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

Во-первых, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

Во-вторых, при движении от конца массива к началу минимальный элемент «всплывает» на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (то есть части массива, где происходит движение) устанавливаются в месте последнего обмена на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

**Анализ шейкерной сортировки.**Анализ  шейкерной сортировки довольно сложен. Стоит отметить, что усовершенствования не влияют на число перемещений, они лишь сокращают число излишних проверок. К несчастью, обмен местами двух элементов – чаще всего более дорогостоящая операция, чем их сравнение. Поэтому очевидные на первый взгляд улучшения дают не такой уж большой вы­игрыш, как ожидалось.

Шейкерная сортировка с успехом используется лишь в тех слу­чаях, когда известно, что элементы почти упорядо­чены, что на практике бывает весьма редко. Анализ показывает, что «обменная» сорти­ровка и ее усовершенствования фактически оказываются хуже сортировок с по­мощью включений и с помощью выбора.

**Быстрая сортировка**

Быстрая сортировка является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного, в том числе, своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Любопытный факт: улучшение самого неэффективного прямого метода сортировки дало в результате один из наиболее эффективных улучшенных методов.

Один из самых быстрых известных универсальных алгоритмов сортировки массивов: в среднем O (n\*log (n)) обменов при упорядочении n элементов; из-за наличия ряда недостатков на практике обычно используется с некоторыми доработками.

Общая идея алгоритма состоит в следующем:

Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность (см.ниже).

Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие».[1]

Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, «меньшие опорного» и «равные и большие»; такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

* Выбрать элемент из массива. Назовём его опорным.
* Разбиение: перераспределение элементов в массиве таким образом, что элементы меньше опорного помещаются перед ним, а больше или равные после.
* Рекурсивно применить первые два шага к двум подмассивам слева и справа от опорного элемента. Рекурсия не применяется к массиву, в котором только один или отсутствуют элементы.

# Ход работы

1. Было разработано настольное Windows-forms приложение, реализующее алгоритмы быстрой и шейкерной сортировки. Пользователь может не только добавлять элементы массива вручную, но и генерировать их рандомно (рис. 1-4). Также, обеспечена устойчивость программы при любых воздействиях, задаваемых пользователем.

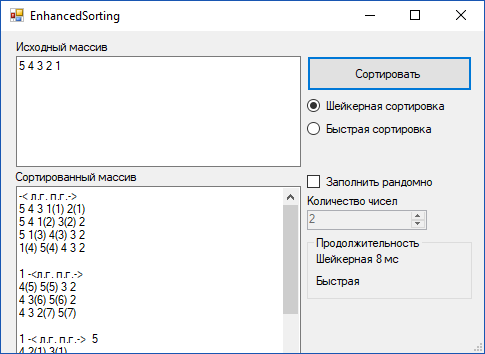


Рисунок 1 – Шейкерная сортировка, ручной ввод

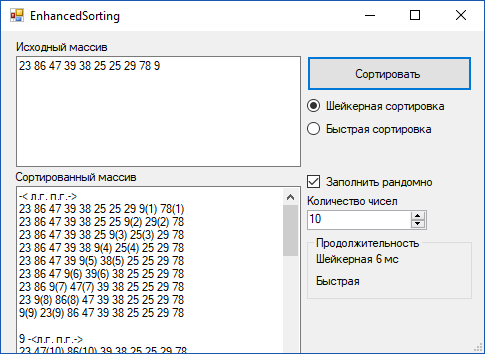


Рисунок 2 - Шейкерная сортировка, рандомный ввод

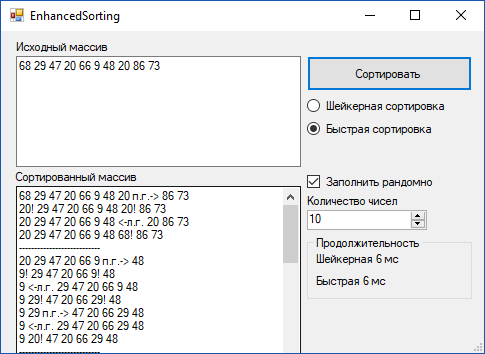


Рисунок 3 – Быстрая сортировка, рандомный ввод

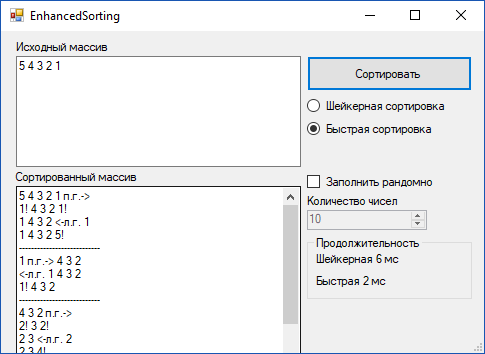


Рисунок 4 – Быстрая сортировка, ручной ввод

1. Проведено сравнение эффективности данных алгоритмов. Для каждого алгоритма в качестве примера использован массив, состоящий из 300 элементов. Каждый элемент массива имеет случайное значение (от 1 до 100) (рисунок 6). Для более точных результатов выполнение каждого алгоритма было рассмотрено на разных вариациях массивов, отличающихся как размерами, так и степенью отсортированности. Для каждого алгоритма программа высчитывает время его исполнения. На рисунках 7-8 можно проследить разницу в выполнении алгоритмов программой.

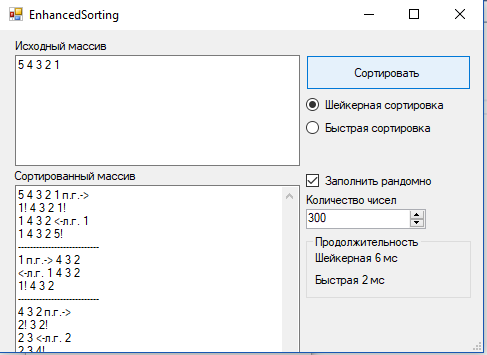


Рисунок 5 – Сравнение времени выполнения сортировок

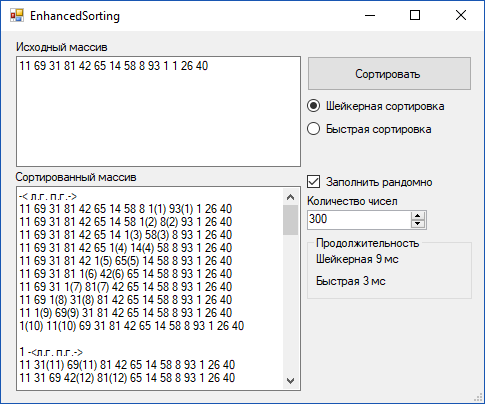


Рисунок 6 – Сравнение времени выполнения сортировок

1. В результате, время выполения сортировки практически всегда оказывается большим у шейкерной сортировки, по сравнению с быстрой. Взяв во внимание все данные результаты сравнений, можно с точность заявить, что алгоритм быстрой сортировки является более эффективным алгоритмом сортировки, относительно алгоритма шейкерной сортировки.

# Вывод

В ходе практической работы были разработаны и отлажены алгоритмы быстрой и шейкерной сортировок. Результаты сравнения эффективности двух алгоритмов совпали с теоретическими ожиданиями. Алгоритм быстрой сортировки превзошел алгоритм шейкерной сортировки.

**ПРИЛОЖЕНИЕ А**

**Листинг кода приложения**

EnhancedSortingForm.cs

|  |
| --- |
| using System;  using System.Windows.Forms;  using ALabs.Algorithms.Converting;  using ALabs.Algorithms.Sorting;  using System.Diagnostics;  namespace ALabs.Forms  {  public partial class EnhancedSortingForm : Form  {  public EnhancedSortingForm()  {  InitializeComponent();  }  private void checkBox1\_CheckedChanged(object sender, EventArgs e)  {  if (checkBox1.Checked)  numericUpDown1.Enabled = true;  else  numericUpDown1.Enabled = false;  }  private void sortingButton\_Click(object sender, EventArgs e)  {  textBox2.Text = string.Empty;  string text = string.Empty;  Stopwatch timer = new Stopwatch();  if (!checkBox1.Checked)  {  if (radioButton1.Checked)  {  timer.Start();  textBox2.Text = SortingAlgorithms.ShakerSort(ConvertingAlgorithms.StringToIntArray(textBox1.Text));  timer.Stop();  ShakeTimeLabel.Text = "Шейкерная " + timer.ElapsedMilliseconds + " мс";  }  else  {  timer.Start();  textBox2.Text = SortingAlgorithms.FastSort(ConvertingAlgorithms.StringToIntArray(textBox1.Text), 0, ConvertingAlgorithms.StringToIntArray(textBox1.Text).Length-1);  timer.Stop();  FastTimeLabel.Text = "Быстрая " + timer.ElapsedMilliseconds + " мс";  }  }  else  {  Random random = new Random();  textBox1.Text = string.Empty;  for (int i = 0; i < numericUpDown1.Value; i++)  text += random.Next(1, 100) + " ";  textBox1.Text = text;  if (radioButton1.Checked)  {  timer.Start();  textBox2.Text = SortingAlgorithms.ShakerSort(ConvertingAlgorithms.StringToIntArray(textBox1.Text));  timer.Stop();  ShakeTimeLabel.Text = "Шейкерная " + timer.ElapsedMilliseconds + " мс";  }  else  {  timer.Start();  textBox2.Text = SortingAlgorithms.FastSort(ConvertingAlgorithms.StringToIntArray(textBox1.Text), 0, ConvertingAlgorithms.StringToIntArray(textBox1.Text).Length-1);  timer.Stop();  FastTimeLabel.Text = "Быстрая " + timer.ElapsedMilliseconds + " мс";  }  }  }  }  } |

SortingAlgorithms.cs

|  |
| --- |
| using System;  namespace ALabs.Algorithms.Sorting  {  public class SortingAlgorithms  {  /// <summary>  /// Шейкерная сортировка  /// </summary>  /// <param name="mas">Вxодной массив</param>  /// <returns>Отсортированный массив</returns>  public static string ShakerSort(int[] mas)  {  string text = string.Empty; int u = 1;  int leftMark = 0; //Левая граница  int rightMark = mas.Length - 1;//Правая граница  while (leftMark <= rightMark)  {  for (int i = 0; i < leftMark; i++)  text += mas[i] + " ";  text += "-< л.г. ";  text += "п.г.-> ";  for (int i = rightMark + 1; i < mas.Length; i++)  text += mas[i] + " ";  text += Environment.NewLine;  for (int i = rightMark; i > leftMark; i--)//Проходим массив справа налево  if (mas[i - 1] > mas[i]) //Если предыдущий элемент больше текущего, меняем иx местами  {  int buff;  buff = mas[i];  mas[i] = mas[i - 1];  mas[i - 1] = buff;  for (int j = leftMark; j <= rightMark; j++)  if (j == i || j == i - 1)  text += mas[j] + "(" + u + ") ";  else  text += mas[j] + " ";  text += Environment.NewLine;  u++;  }  leftMark++; //Сдвигаем левую границу вправо  text += Environment.NewLine;  for (int i = 0; i < leftMark; i++)  text += mas[i] + " ";  text += "-<л.г. ";  text += "п.г.-> ";  for (int i = rightMark + 1; i < mas.Length; i++)  text += mas[i] + " ";  text += Environment.NewLine;  for (int i = leftMark+1; i <= rightMark; i++)//Проходим массив слева направо  if (mas[i - 1] > mas[i]) //Если предыдущий элемент больше текущего, меняем иx местами  {  int buff;  buff = mas[i];  mas[i] = mas[i - 1];  mas[i - 1] = buff;  for (int j = leftMark; j <= rightMark; j++)  if (j == i || j == i - 1)  text += mas[j] + "(" + u + ") ";  else  text += mas[j] + " ";  text += Environment.NewLine;  u++;  }    rightMark--;//Сдвигаем правую границу влево  text += Environment.NewLine;    u = 1;  }  for (int i = 0; i < mas.Length; i++)  text += mas[i] + " ";  return text;  }  /// <summary>  /// Быстрая сортировка  /// </summary>  /// <param name="mas">Вxодной массив</param>  /// <returns>Отсортированный массив</returns>  public static string FastSort(int[] mas, int left, int right)  {  string text = string.Empty;  int pivot; // разрешающий элемент  int l\_hold = left; //левая граница  int r\_hold = right; // правая граница  pivot = mas[left];    while (left < right) // пока границы не сомкнутся  {  while ((mas[right] >= pivot) && (left < right))  right--; // сдвигаем правую границу пока элемент [right] больше [pivot]  for (int i = l\_hold; i < r\_hold + 1; i++)  if (i != right)  text += mas[i] + " ";  else  text += mas[i] + " п.г.-> ";  text += Environment.NewLine;  if (left != right) // если границы не сомкнулись  {  mas[left] = mas[right]; // перемещаем элемент [right] на место разрешающего  for (int i = l\_hold; i < r\_hold + 1; i++)  if (i != left && i != right)  text += mas[i] + " ";  else  text += mas[i] + "! ";  text += Environment.NewLine;  left++; // сдвигаем левую границу вправо  }  while ((mas[left] <= pivot) && (left < right))  left++; // сдвигаем левую границу пока элемент [left] меньше [pivot]  for (int i = l\_hold; i < r\_hold + 1; i++)  if (i != left)  text += mas[i] + " ";  else  text += "<-л.г. " + mas[i] + " ";  text += Environment.NewLine;  if (left != right) // если границы не сомкнулись  {  mas[right] = mas[left]; // перемещаем элемент [left] на место [right]  for (int i = l\_hold; i < r\_hold + 1; i++)  if (i != left && i != right)  text += mas[i] + " ";  else  text += mas[i] + "! ";  text += Environment.NewLine;  right--; // сдвигаем правую границу вправо  }  }    mas[left] = pivot; // ставим разрешающий элемент на место  for (int i = l\_hold; i < r\_hold + 1; i++)  if (i != left)  text += mas[i] + " ";  else  text += mas[i] + "! ";  text += Environment.NewLine + "---------------------------" + Environment.NewLine;  pivot = left;  left = l\_hold;  right = r\_hold;  if (left < pivot) // Рекурсивно вызываем сортировку для левой и правой части массива  text += FastSort(mas, left, pivot - 1);  if (right > pivot)  text += FastSort(mas, pivot + 1, right);    return text;  }  }  } |