СОДЕРЖАНИЕ

[ВСТУПЛЕНИЕ 3](#_Toc17218524)

[1. СОРТИРОВКА ПУЗЫРЬКОМ 4](#_Toc17218525)

[2. СОРТИРОВКА ВСТАВКАМИ 6](#_Toc17218526)

[3. СОРТИРОВКА ПЕРЕМЕШИВАНИЕМ 8](#_Toc17218527)

[4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ ПРОГРАММЫ 10](#_Toc17218528)

[5. РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЯ 11](#_Toc17218529)

[6. ЧИСЛЕННЫЙ ЭКСПЕРИМЕНТ 14](#_Toc17218530)

[ЗАКЛЮЧЕНИЕ 16](#_Toc17218531)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 17](#_Toc17218532)

[ПРИЛОЖЕНИЕ А 18](#_Toc17218533)

# ВСТУПЛЕНИЕ

Практически в каждом программном проекте возникает необходимость обработки большого числа единообразно организованных данных. В таких случаях подобные данные удобно обрабатывать как единое целое, для чего он представляется в виде массива — именованной последовательности данных одного типа.

С понятием "массив" приходится сталкиваться при решении научно-технических и экономических задач обработки совокупностей большого количества значений. Массивы упрощают процесс управления данными, когда используется несколько элементов данных одного типа. Массивы полезны тем, что они помогают обрабатывать большие объемы данных такими способами, которые оказались бы нереализуемыми.

Сортировка — один из наиболее распространенных процессов современной обработки данных. Сортировкой называется распределение элементов множества по группам в соответствии с определенными правилами. Например, сортировка элементов массива, каждый элемент которого, начиная со второго, не больше стоящего от него слева, называется сортировкой по не возрастанию.

В данной курсовой работе рассматриваются несколько способов сортировки массива. В настоящее время существует множество алгоритмов сортировки массивов, которые применяются в зависимости от того какие условия стоят перед разрабатываемой программой.

Программа написана на языке программирования C#. Для визуальной части я выбрал WPF это система для построения клиентских приложений Windows с визуально привлекательными возможностями взаимодействия с пользователем.

В теоретической части представлены алгоритмы сортировок. В практической части реализовала программа, показывающая процесс сортировки массива.

Объект исследования: сортировки массивов.

Постановка задачи: составить программу показывающая процесс сортировки массива.

# СОРТИРОВКА ПУЗЫРЬКОМ

Сортировка пузырьком (англ. Bubble sort) [1] — для понимания и реализации этот алгоритм — простейший, но эффективен он лишь для небольших массивов. Сложность алгоритма: О(n2).

Алгоритм считается учебным и практически не применяется вне учебной литературы, вместо него на практике применяются более эффективные алгоритмы сортировки. В то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов, таких как шейкерная сортировка, пирамидальная сортировка и быстрая сортировка.

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются n-1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма) [4].

Особенность данного алгоритма заключается в следующем: после первого завершения внутреннего цикла максимальный элемент массива всегда находится на n-ой позиции. При втором проходе, следующий по значению максимальный элемент находится на n-1 месте. И так далее. Таким образом, на каждом следующем проходе число обрабатываемых элементов уменьшается на 1 и нет необходимости «обходить» весь массив от начала до конца каждый раз.

Так как подмассив из одного элемента не нуждается в сортировке, то для сортировки требуется делать не более n-1 итераций внешнего цикла. Поэтому в некоторых реализациях внешний цикл всегда выполняется ровно n-1 и не отслеживается, были или не были обмены на каждой итерации.

Введение индикатора (флажка F) действительно произошедших во внутреннем цикле обменов уменьшает число лишних проходов в случаях с частично отсортированными массивами на входе. Перед каждым проходом по внутреннему циклу флажок сбрасывается в 0, а после действительно произошедшего обмена устанавливается в 1. Если после выхода из внутреннего цикла флажок равен 0, то обменов не было, то есть массив отсортирован и можно досрочно выйти из программы сортировки.

Псевдокод алгоритма с проверкой действительно произошедших обменов во внутреннем цикле.

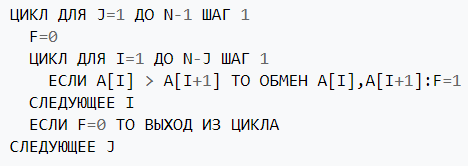


Рисунок 1. Псевдокод алгоритма пузырьком

# СОРТИРОВКА ВСТАВКАМИ

Сортировка вставками (англ. Insertion sort) [2] — алгоритм сортировки, в котором элементы входной последовательности просматриваются по одному, и каждый новый поступивший элемент размещается в подходящее место среди ранее упорядоченных элементов. Сложность алгоритма: О(n2).

На вход алгоритма подаётся последовательностьтn чисел: a1, a2, ..., an. Сортируемые числа также называют ключами. Входная последовательность на практике представляется в виде массива с n элементами. На выходе алгоритм должен вернуть перестановку исходной последовательности a1', a2', ..., an', чтобы выполнялось следующее соотношение a1'≤ a2'≤ ... ≤ an'.

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан.

Данный алгоритм можно ускорить при помощи использования бинарного поиска для нахождения места текущему элементу в отсортированной части.

На вход процедуре сортировки подаётся массив A[1..n], состоящий из элементов последовательности A[1], A[2], ..., A[n], которые требуется отсортировать. n соответствует — размеру исходного массива. Для сортировки не требуется привлечения дополнительной памяти, кроме постоянной величины для одного элемента, так как выполняется перестановка в пределах массива. В результате работы процедуры во входном массиве оказывается требуемая выходная последовательность элементов [4].

Псевдокод алгоритма:

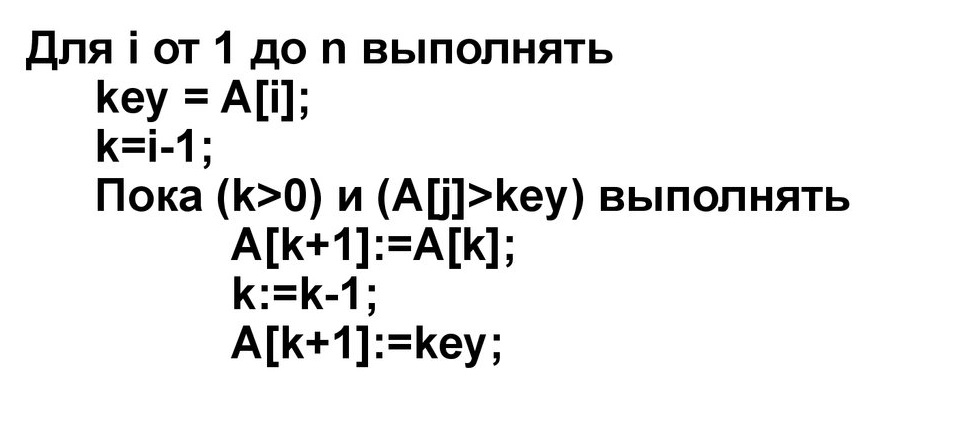


Рисунок 2. Псевдокод алгоритма вставками

# СОРТИРОВКА ПЕРЕМЕШИВАНИЕМ

Сортировка перемешиванием (англ. Cocktail sort) [3] — 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. Лучший случай для этой сортировки — отсортированный массив О(n), худший — отсортированный в обратном порядке О(n2).

Образно алгоритм можно описать так: на каждом шаге основного цикла рассматривается массив a[Left] ÷ a[Right], после выполнения двух внутренних циклов минимальный и максимальный элемент в исходном массиве перетекают к краям, минимальный в — a[Left], максимальный — в a[Right]. Пусть максимальный элемент имеет индекс k, тогда массив можно изобразить так: a[Left], a[1], .., a[k-1], a[k], a[k+1], ..,a[Right]. После сравнения a[k] с a[k+1] значение a[k] перейдет в k+1-ую ячейку, после сравнения k+1-й c k+2-й — в k+2‑eю, и так далее, пока он не сместится в крайне правое положение с индексом Right. Аналогично для минимального. После выполнения цикла по всем подмассивам он отсортируется [4].

Псевдокод алгоритма:

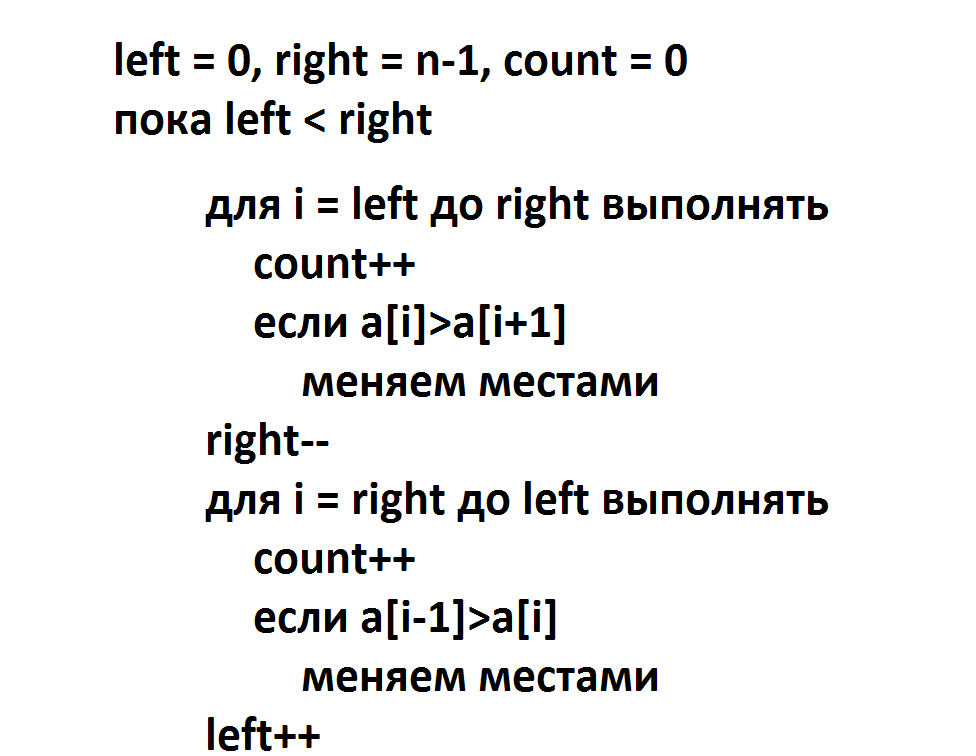


Рисунок 3. Псевдокод алгоритма перемешиванием

# ОСОБЕННОСТИ РЕАЛИЗАЦИИ ПРОГРАММЫ

Программа написана на языке C# в среде Microsoft Visual Studio. Основная часть реализована в классе: MainWindow. Для задержки сортировки можно воспользоваться экземпляром класса Thread для отдельного потока.

1. private int[] ArrayGeneration() В этом методе происходит генерация массива случайными числами от 5 до 175. *Результатом работы метода* является массива заполненный случайными числами.
2. private void InitializeLines(int[] array) В этом методе происходит построение линий на форму где число из массива обозначает угол наклона линии по отношению к горизонту формы. *Исходные данные метода*: array – массив значений угла наклона.
3. private int[] BulbSort(int[] input, int SelectColor) В этом методе происходить пузырьковая сортировка. *Исходные данные метода*: input - массив по которому будет произведена сортировка, SelectColor – переменная, которая хранит цвет линии. *Результатом работы метода* является отсортированный массив.
4. private int[] InsertionSort(int[] input, int SelectColor) Аналогично методу BulbSort, этот метод сортировки вставками.
5. private int[] ShakerSort(int[] input, int SelectColor) Этот метод сортировки смешиванием.
6. private void SetLine(int index) В этом методе происходит установка линии на нужное место в соответствии с индексом в массиве. *Исходные данные метода*: index – индекс для определения места в массиве.
7. private void SwapLines(int index1, int index2) В этом методе происходит изменение положения двух палочек и отображение на форму.
8. private void PrintArray(int[] array) В этом методе происходит печать массива со значениями градуса наклона линии на форму.

# РУКОВОДСТВО ДЛЯ ПОЛЬЗОВАТЕЛЯ

В данной программе происходит сортировка массива с пошаговой визуализацией.

Исходные данные, набор значений, должны быть записаны в текстовом файле формата «txt». Где значения должны быть целыми числами и принадлежать промежутку от 5 до 175, количество этих значений не должно превышать 50 и не быть меньше 10. Разделяющим знаком между цифрами является пробел.

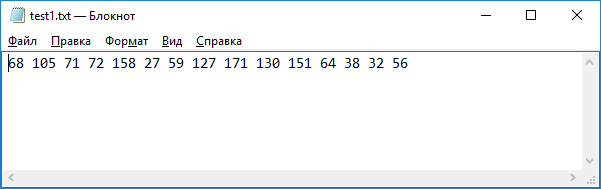


Рисунок 4. Исходные данные

Так же ввод данных можно осуществлять через нажатие на кнопку «Генерировать» для генерирования случайных значений в заданном количестве.

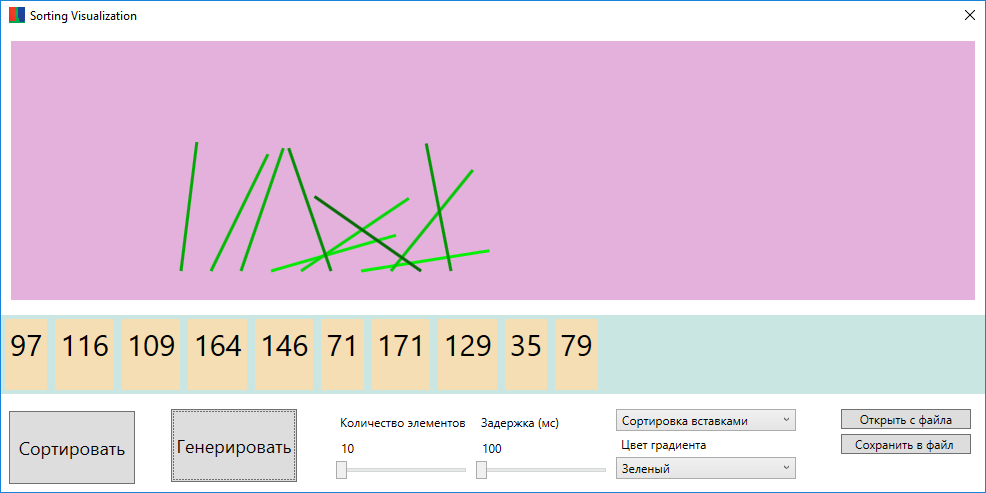


Рисунок 5. Ввод данных

Для ввода данных с файла вначале в главном окне программы следует выбрать «Открыть с файла» (см. рис. 5). После этого появится окно, где следует выбрать файл, в котором записаны данные (в примере это файл test1.txt – см. рис. 6).

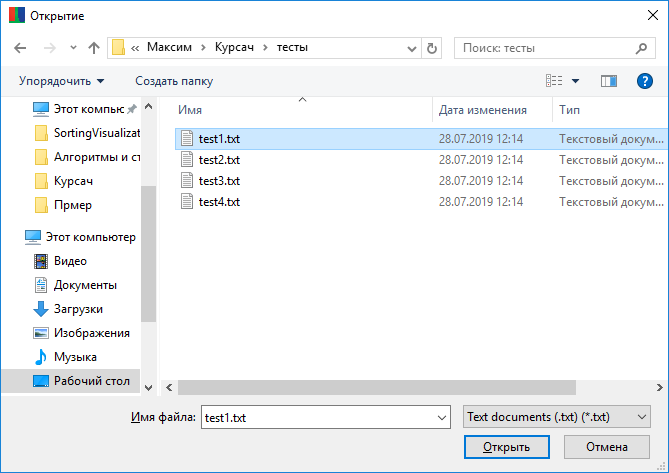


Рисунок 6. Выбор файла

Далее нажимаем кнопку «Сортировать» (см. рис. 7). При нажатии этой кнопки выполнится пошаговая сортировка массива выбранным способом.

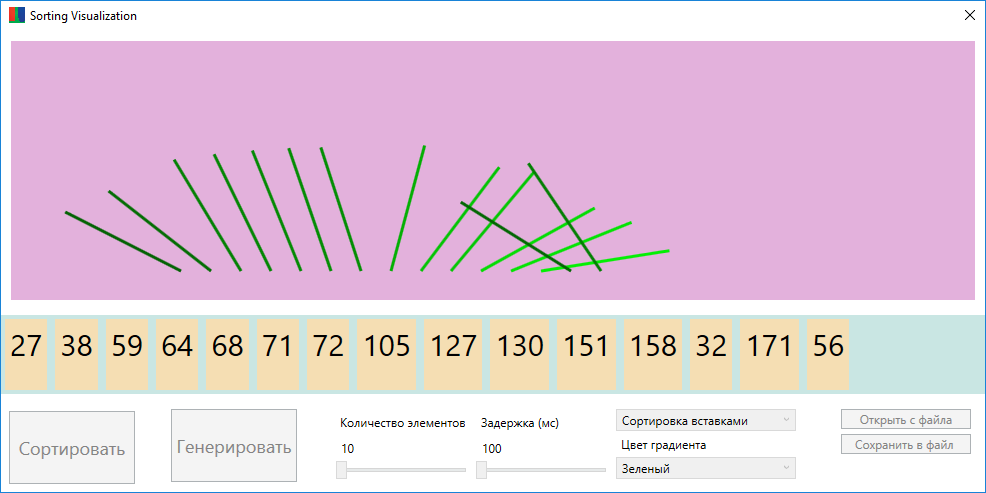


Рисунок 7. Решение задачи

Для вывода результата в файл следует выбрать «Сохранить в файл». После этого появится окно, где следует сохранить файл (см. рис. 8).

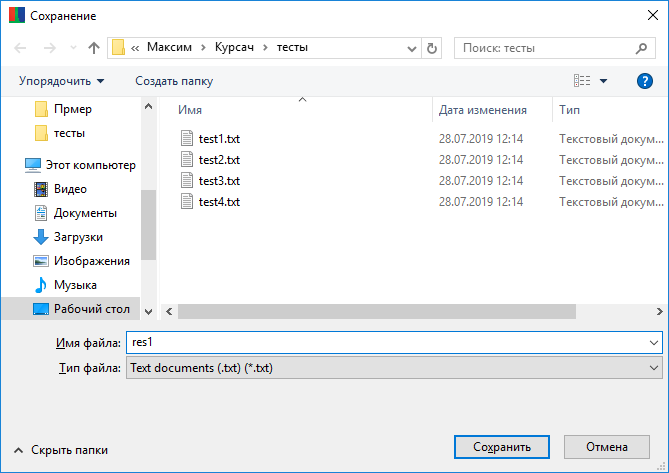


Рисунок 8. Вывод результата

Замечание. По окончанию сортировки выскачет окно с оповещением о времени работы сортировки.

# ЧИСЛЕННЫЙ ЭКСПЕРИМЕНТ

Для проверки правильности работы сортировки достаточно просто просмотреть итоговый результат.

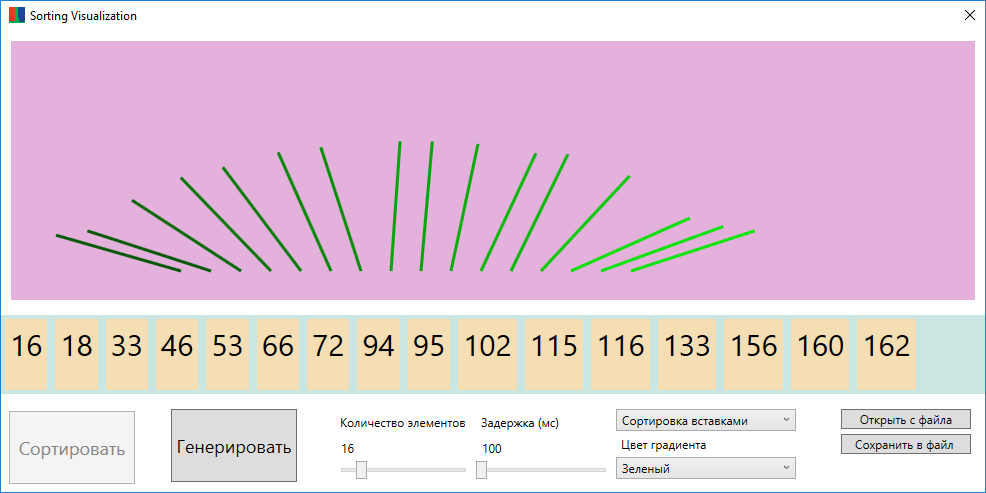


Рисунок 9. итоговый результат сортировки

В следствии чего можно провести эксперимент, проверить скорость сортировок на разном количестве случайных значений. Таким образом можно построить таблицу и график, зависимость работы программы по времени от и количества случайных значений. Тесты проводились с одинаковой задержкой 100 мс. Содержимое файлов:

test1.txt - 68 105 71 72 158 27 59 127 171 130 151 64 38 32 56

test2.txt - 128 16 126 168 111 39 39 50 16 117 100 30 126 110 150 112 95 163 60 131 106 37 32 169 85

test3.txt - 125 169 79 37 63 24 29 5 146 63 58 42 43 160 132 38 99 42 131 82 56 108 56 29 15 39 172 45 55 159 54 64 134 94 122 136 158 146 19 114

|  |  |  |  |
| --- | --- | --- | --- |
|  | Пузырек | Вставками | Смешиванием |
| test1 | 10:56 сек | 06:30 сек | 03:03 сек |
| test2 | 30:17 сек | 14:60 сек | 07:39 сек |
| test3 | 78:40 сек | 33:90 сек | 17:06 сек |

Таблица 1. Сравнение сортировок

График 1. Сравнение сортировок

По графику видно, что сортировка смешиванием работает в разы быстрее чем пузырьковая сортировка и сортировка вставками, чего и следовало ожидать ведь алгоритмическая сложность сортировки смешиванием O(n) в лучшем случае, где n это количество сортируемых значений.

# ЗАКЛЮЧЕНИЕ

При написании данной курсовой работы были рассмотрены различные теоретические сведения. Основываясь на источники, которые были изучены, я могу сделать вывод, что сортировки очень важны и количество их разнообразия очень велико, теоретическая часть подтвердилась на практике.

В результате выполнения курсовой работы была написана программа, выполняющая пошаговую сортировку массива разными способами. Программа обладает высокими параметрами быстродействия, маленькими размерами и не требовательны к системным ресурсам компьютера.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

|  |  |
| --- | --- |
| [1] | Сортировка пузырьком *Wikipedia* : веб-сайт. URL: https://en.wikipedia.org/wiki/Bubble\_sort  (дата обращения: 20.08.2019). |
| [2] | Сортировка вставками *Wikipedia* : веб-сайт. URL: https://en.wikipedia.org/wiki/Insertion\_sort  (дата обращения: 20.08.2019) |
| [3] | Сортировка перемешиванием *Wikipedia* : веб-сайт. URL: https://en.wikipedia.org/wiki/Cocktail\_shaker\_sort  (дата обращения: 20.08.2019) |
| [4] | Visualizing Algorithms *bost.ocks*: веб-сайт. URL: https://bost.ocks.org/mike/algorithms/#sorting  (дата обращения: 20.08.2019) |

[5] Структуры данных и алгоритмы - Джеффри Ульман, Джон Хопкрофт, Альфред Ахо,

2010 г. - 400 с.

[6] Дискретная математика и комбинаторика - Джеймс Андерсон 2019 г. - 960 с.

[7] Основы дискретной математики. Учебное пособие - В. А. Осипова 2016 г - 160 с.

# ПРИЛОЖЕНИЕ А

MainWindow.cs

using System;

using System.Windows.Shapes;

using System.Windows;

using System.Windows.Media;

using System.Threading;

using System.Windows.Threading;

using System.Windows.Controls;

using System.IO;

using Microsoft.Win32;

using System.Diagnostics;

namespace sorting\_visualization

{

public partial class MainWindow : Window

{

private int d = 100;//минимальная задержка

private double r = 130;//длина палочки

private double a = 140, b = 230;//отступы от краёв формы для построения палочек

private int[] numbers;//массив значений

private Line[] lines;//массив линий

private Thread SortingThread;//экземпляр класса Thread для отдельного потока

public MainWindow()

{

InitializeComponent();

Closed += EndProgram;

InitializeRandom();

}

private int[] ArrayGeneration()

{

//Генерирование массива значений от 5 до 175

Random random = new Random();

int[] numbers = new int[(int)InputN.Value];

for (int i = 0; i < (int)InputN.Value; i++)

numbers[i] = random.Next(5, 175);

return numbers;

}

private void InitializeLines(int[] array)

{

//построение палочек на форму

DrawingCanvas.Children.Clear();

lines = new Line[array.Length];

for (int i = 0; i < array.Length; i++)

{

//создание палочки с нужными характеристиками и добавление на форму

Line line = new Line();

line.X1 = a + (double)(i + 1) \* 30;

line.Y1 = b;

line.X2 = -Math.Cos(((double)array[i] / 180) \* Math.PI) \* r + line.X1;

line.Y2 = -Math.Sin(((double)array[i] / 180) \* Math.PI) \* r + line.Y1;

line.StrokeThickness = 3;

SolidColorBrush brush = null;

switch (SelectColor.SelectedIndex)

{

case 0:

{

brush = new SolidColorBrush(Color.FromArgb(250, (byte)(array[i] + 70), 0, 0));

break;

}

case 1:

{

brush = new SolidColorBrush(Color.FromArgb(250, 0, (byte)(array[i] + 70), 0));

break;

}

case 2:

{

brush = new SolidColorBrush(Color.FromArgb(250, 0, 0, (byte)(array[i]+70)));

break;

}

default:

break;

}

line.Stroke = brush;

lines[i] = line;

DrawingCanvas.Children.Add(line);

}

}

private void EndProgram(object sender, EventArgs e)

{

//обработка закрытия программы

if (SortingThread != null)

SortingThread.Abort();

Close();

}

private void RunSorting()

{

//вызов сортировки и отлючение кнопок

RunButton.IsEnabled = false;

RandomButton.IsEnabled = false;

InputN.IsEnabled = false;

InputDelay.IsEnabled = false;

OpenBtn.IsEnabled = false;

SaveBtn.IsEnabled = false;

SelectBox.IsEnabled = false;

SelectColor.IsEnabled = false;

int selectSort = SelectBox.SelectedIndex;

int selectColor = SelectColor.SelectedIndex;

SortingThread = new Thread(delegate ()

{

Sorting(selectSort, selectColor);

});

SortingThread.Start();

}

private void Sorting(int SelectSort, int selectColor)

{

//запуск сортировки, по окончанию включение кнопок

//подсчет времени работы сортировки

//выбор сортировки

Stopwatch stopWatch = new Stopwatch();

stopWatch.Start();

switch (SelectSort)

{

case 0:

{

numbers = BulbSort(numbers, selectColor);

break;

}

case 1:

{

numbers = InsertionSort(numbers, selectColor);

break;

}

case 2:

{

numbers = ShakerSort(numbers, selectColor);

break;

}

default:

InitializeRandom();

return;

}

Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate ()

{

RunButton.IsEnabled = false;

RandomButton.IsEnabled = true;

InputN.IsEnabled = true;

InputDelay.IsEnabled = true;

OpenBtn.IsEnabled = true;

SaveBtn.IsEnabled = true;

SelectBox.IsEnabled = true;

SelectColor.IsEnabled = true;

stopWatch.Stop();

TimeSpan ts = stopWatch.Elapsed;

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:00}", ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);

MessageBox.Show(elapsedTime,"Время работы");

});

}

private int[] BulbSort(int[] input, int SelectColor)

{

//сортировка пузырьком

for (int i = 0; i < input.Length; i++)

{

for (int j = i + 1; j < input.Length; j++)

{

if (input[j] < input[i])

{

int temp = input[j];

input[j] = input[i];

input[i] = temp;

SwapLines(i, j); //изменение положения палочек

PrintArray(numbers); //печать массива

}

Thread.Sleep(d); //задержка

}

SetColor(i, numbers[i], SelectColor);//установка цвета

}

return input;

}

private int[] InsertionSort(int[] input, int SelectColor)

{

//сортировка вставками

int counter = 0;

for (int i = 1; i < input.Length; i++)

{

for (int j = i; j > 0 && input[j - 1] > input[j]; j--)

{

counter++;

int tmp = input[j - 1];

input[j - 1] = input[j];

input[j] = tmp;

Thread.Sleep(d); //задержка

PrintArray(numbers); //печать массива

SwapLines(j-1,j);

}

}

return input;

}

private int[] ShakerSort(int[] input, int SelectColor)

{

//сортировка смешиванием

int left = 0,

right = input.Length - 1,

count = 0;

while (left < right)

{

for (int i = left; i < right; i++)

{

count++;

if (input[i] > input[i + 1])

{

int tmp = input[i + 1];

input[i + 1] = input[i];

input[i] = tmp;

Thread.Sleep(d/2); //задержка

SwapLines(i+1, i); //изменение положения палочек

}

}

PrintArray(numbers); //печать массива

SetLine(right);

SetColor(right,numbers[right], SelectColor);//установка цвета

right--;

SetColor(right, numbers[right], SelectColor);//установка цвета

for (int i = right; i > left; i--)

{

count++;

if (input[i - 1] > input[i])

{

int tmp = input[i - 1];

input[i - 1] = input[i];

input[i] = tmp;

Thread.Sleep(d/2); //задержка

SwapLines(i - 1, i); //изменение положения палочек

}

}

PrintArray(numbers); //печать массива

SetLine(left);

SetColor(left, numbers[left], SelectColor);//установка цвета

left++;

SetColor(left, numbers[left], SelectColor);//установка цвета

}

return input;

}

private void SetColor(int index, int gr, int selectColor)

{

//изменение цвета палочки, которые на своём месте

gr += 70;

Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate ()

{

SolidColorBrush brush = null;

switch (selectColor)

{

case 0: {

brush = new SolidColorBrush(Color.FromArgb(250, (byte)gr, 0 , 0));

break;

}

case 1:

{

brush = new SolidColorBrush(Color.FromArgb(250, 0, (byte)gr, 0));

break;

}

case 2:

{

brush = new SolidColorBrush(Color.FromArgb(250, 0, 0, (byte)gr));

break;

}

default:

break;

}

lines[index].Stroke = brush;

});

}

private void SetLine(int index)

{

//изменение положения палочки

Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate ()

{

for (int i = 0; i <= index; i++)

{

double Onex1 = lines[i].X1;

double Oney1 = lines[i].Y1;

lines[i].X2 = -Math.Cos(((double)numbers[i] / 180) \* Math.PI) \* r + lines[i].X1;

lines[i].Y2 = -Math.Sin(((double)numbers[i] / 180) \* Math.PI) \* r + lines[i].Y1;

}

});

}

private void SwapLines(int index1, int index2)

{

//изменение положения двух палочек

Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate ()

{

double Onex1 = lines[index1].X1;

double Oney1 = lines[index1].Y1;

double Twox1 = lines[index2].X1;

double Twoy1 = lines[index2].Y1;

lines[index2].X1 = Onex1;

lines[index2].Y1 = Oney1;

lines[index2].X2 = -Math.Cos(((double)numbers[index2] / 180) \* Math.PI) \* r + lines[index2].X1;

lines[index2].Y2 = -Math.Sin(((double)numbers[index2] / 180) \* Math.PI) \* r + lines[index2].Y1;

lines[index1].X1 = Twox1;

lines[index1].Y1 = Twoy1;

lines[index1].X2 = -Math.Cos(((double)numbers[index1] / 180) \* Math.PI) \* r + lines[index1].X1;

lines[index1].Y2 = -Math.Sin(((double)numbers[index1] / 180) \* Math.PI) \* r + lines[index1].Y1;

});

Line temp = lines[index1];

lines[index1] = lines[index2];

lines[index2] = temp;

}

private void InitializeRandom()

{

//инициализация палочек и значений

d = (int)InputDelay.Value;

numbers = ArrayGeneration();

InitializeLines(numbers);

OutputTextBox.Children.Clear();

foreach (int i in numbers)

{

Label label = new Label();

label.Background = Brushes.Wheat;

label.FontSize = 30;

label.Content = i;

label.BorderThickness = new Thickness(4);

OutputTextBox.Children.Add(label);

}

RunButton.IsEnabled = true;

}

private void RandomButton\_Click(object sender, RoutedEventArgs e)

{

InitializeRandom();

}

private void InputN\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

Quantity.Content = ((int)InputN.Value).ToString();

}

private void InputDelay\_ValueChanged(object sender, RoutedPropertyChangedEventArgs<double> e)

{

Delay.Content = ((int)InputDelay.Value).ToString();

}

private void PrintArray(int[] array)

{

//печать массива значений

Dispatcher.BeginInvoke(DispatcherPriority.Normal, (ThreadStart)delegate ()

{

OutputTextBox.Children.Clear();

foreach (int i in array)

{

Label label = new Label();

label.Background = Brushes.Wheat;

label.FontSize = 30;

label.Content = i;

label.BorderThickness = new Thickness(4);

OutputTextBox.Children.Add(label);

}

});

}

private void OpenBtn\_Click(object sender, RoutedEventArgs e)

{

//Считывание с файла

try

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "Text documents (.txt)|\*.txt";

openFileDialog.ShowDialog();

string temp = File.ReadAllText(openFileDialog.FileName);

string[] Lines = temp.Split(new char[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);

if (Lines.Length > InputN.Maximum || Lines.Length < InputN.Minimum)

throw new Exception("Ошибка");

numbers = new int[Lines.Length];

for (int i = 0; i < Lines.Length; i++)

numbers[i] = Int32.Parse(Lines[i]);

InitializeLines(numbers);

OutputTextBox.Children.Clear();

foreach (int i in numbers)

{

Label label = new Label();

label.Background = Brushes.Wheat;

label.FontSize = 30;

label.Content = i;

label.BorderThickness = new Thickness(4);

OutputTextBox.Children.Add(label);

}

RunButton.IsEnabled = true;

}

catch (Exception exce)

{

MessageBox.Show(exce.Message);

}

}

private void SaveBtn\_Click(object sender, RoutedEventArgs e)

{

try

{

//сохранение результата в файл

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Text documents (.txt)|\*.txt";

saveFileDialog.ShowDialog();

string SaveResult="";

for (int i = 0; i < numbers.Length; i++)

SaveResult += numbers[i].ToString() + " ";

File.WriteAllText(saveFileDialog.FileName, SaveResult);

MessageBox.Show("Файл сохранён", "Система");

}

catch (Exception exce)

{

MessageBox.Show(exce.Message);

}

}

private void RunSortingButton\_Click(object sender, RoutedEventArgs e)

{

RunSorting();

}

}

}