**Министерство образования и наук РФ**

**Федеральное государственное бюджетное образовательное**

**учреждение высшего профессионального образования**

**«Северо-Осетинский государственный университет**

**имени Коста Левановича Хетагурова»**

Факультет: Математики и информационных технологий

Направление: Прикладная математика и информатика

**Практическая работа**

На тему:

**«Шейкер сортировка»**

**Выполнил:**

Студент 1-го курса. Группы 11:

*Болотаев Владислав Георгиевич*

**Научный руководитель:**

*Молчанова Ирина Александровна*

г. Владикавказ

2020г.

Содержание

[**Описание сортировки** 3](#_Toc43560727)

[**Сложность сортировки** 4](#_Toc43560728)

[**Реализация(псевдокод)** 4](#_Toc43560729)

[**Реализация(C#)** 5](#_Toc43560730)

[**Руководство системного администратора** 7](#_Toc43560731)

[**Руководство пользователя** 7](#_Toc43560732)

[**Руководство программиста** 11](#_Toc43560733)

[**Список используемой литературы** 18](#_Toc43560734)

# **Описание сортировки**

Шейкер-сортировка является усовершенствованным методом [пузырьковой сортировки](https://prog-cpp.ru/sort-bubble/).

Проанализировав метод пузырьковой сортировки, отметим:

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

Данная модификация пузырьковой сортировки и называется Шейкер-сортировкой.

Образно алгоритм можно описать так: на каждом шаге основного цикла рассматривается часть массива, заключенная между элементами a[Left] и 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],a[k+1],..,a[Right];После сравнения а[k] с a[k+1] значение а[k] перейдет в k+1-ую ячейку, после сравнения k+1-й c k+2-й – в k+2-ую,и так далее, пока он не сместится в крайне правое положение с индексом Right, после чего граница Right сместится на одну позицию. Аналогично для минимального. После выполнения цикла по всем подмассивам он отсортируется.

## **Сложность сортировки**

|  |  |
| --- | --- |
| Лучший случай | O(n) |
| Средний случай | O(n2) |
| Худший случай | O(n2) |

## **Реализация(псевдокод)**

function A = cocktailShakerSort(A)

beginIdx = 1;

endIdx = length(A)-1;

while beginIdx <= endIdx

newBeginIdx = endIdx;

newEndIdx = beginIdx;

for ii= beginIdx:endIdx

if A(ii) > A(ii + 1)

[A(ii+1), A(ii)] = deal(A(ii), A(ii+1));

newEndIdx=ii;

end

end

endIdx = newEndIdx - 1;

for ii= endIdx:-1:beginIdx

if A(ii) > A(ii + 1)

[A(ii+1), A(ii)] = deal(A(ii), A(ii+1));

newBeginIdx = ii;

end

end

beginIdx = newBeginIdx + 1;

end

end

## **Реализация(C#)**

int left = 0, right = countGlob - 1; // левая и правая границы сортируемой области массива

int flag = 1; // флаг наличия перемещений

int t; // вспомогательная переменная

// Цикл выполняется пока левая граница не сомкнётся с правой и пока в массиве имеются перемещения

while ((left < right) && flag > 0)

{

flag = 0;

for (int i = left; i < right; i++) //двигаемся слева направо

{

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

{

t = nums[i];

nums[i] = nums[i + 1];

nums[i + 1] = t;

flag = 1;

}

}

right--; // сдвигаем правую границу на предыдущий элемент

for (int i = right; i > left; i--) //двигаемся справа налево

{

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

{

t = nums[i];

nums[i] = nums[i - 1];

nums[i - 1] = t;

flag = 1;

}

}

left++; // сдвигаем левую границу на следующий элемент

}

# **Руководство системного администратора**

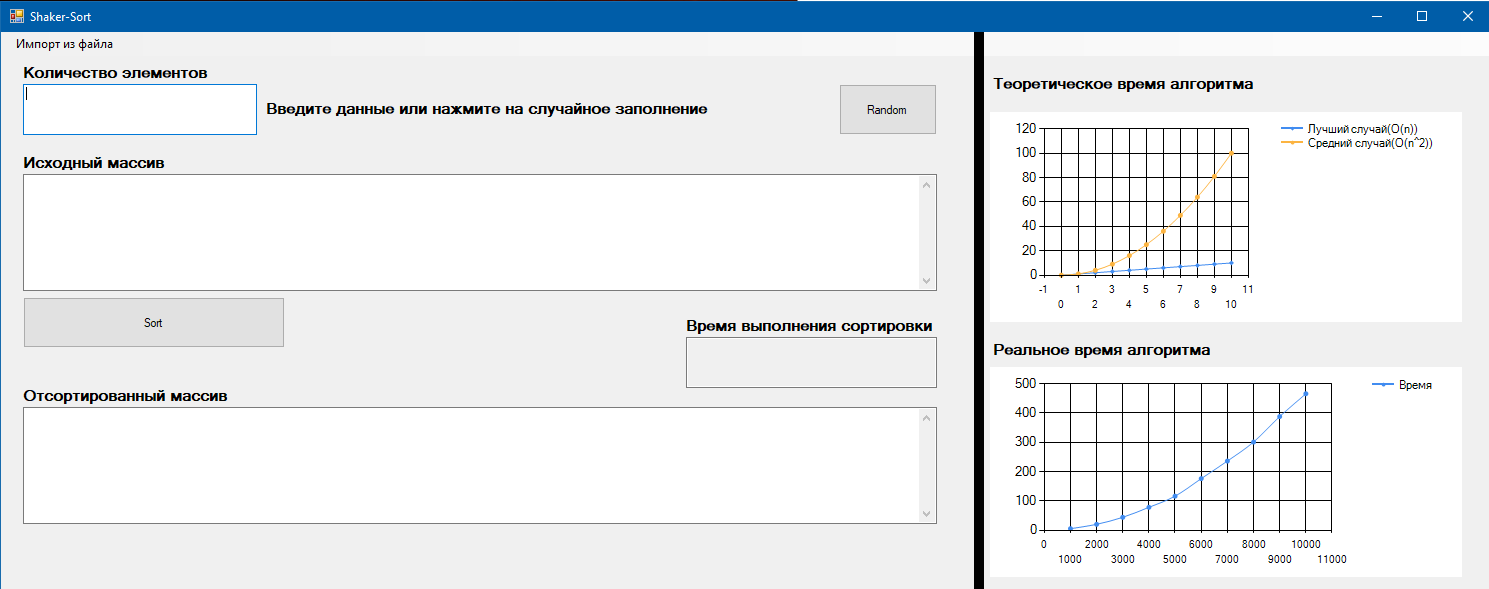
Ссылка на проект: <https://github.com/slava-vl/ShakerSort>

Разработанная система позволяет сортировать массив чисел, используя Шейкер-сортировку.

Системные требования к программному продукту: ОС Windows 7 или выше, Framework 4.0 или выше.

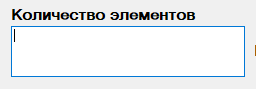
# **Руководство пользователя**

При открытии вы увидите следующее окно:



Для того чтобы отсортировать массив вам необходимо:

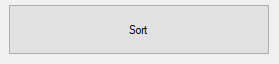
* В поле «Количество элементов» введите длину массива



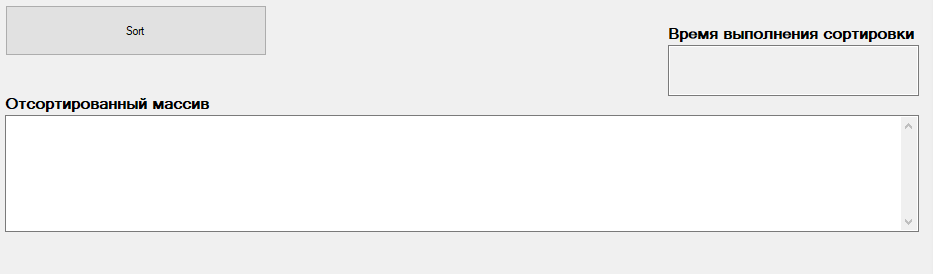
* В поле «Исходный массив» введите сам массив



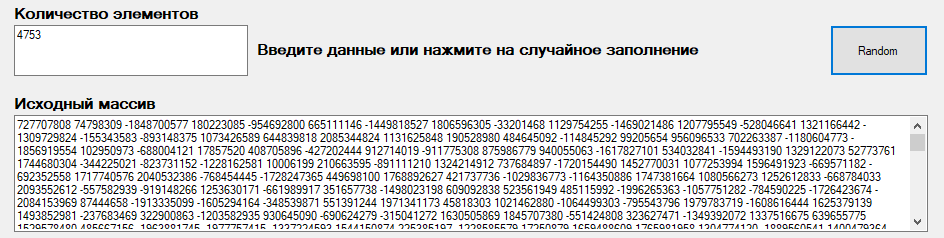
* Для того чтобы отсортировать массив Нажмите на кнопку «Sort»



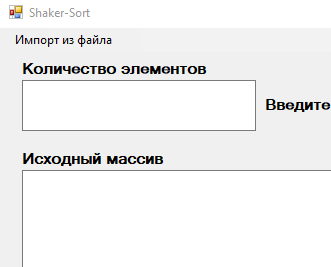
* После нажатия отсортированный массив появится в поле «Отсортированный массив», а время выполнения сортировки в поле «Время выполнения сортировки»



* Если вы хотите отсортировать случайное количество элементов, нажмите на кнопку «Random», после чего поля «Количество элементов» и «Исходный массив» заполнятся случайными числами



* Если необходимо отсортировать данные из файла, нажмите на кнопку «Импорт из файла», после чего откроется окно проводника



* В проводнике выберите нужный файл формата .txt, после чего содержимое файла будет внесено в форму

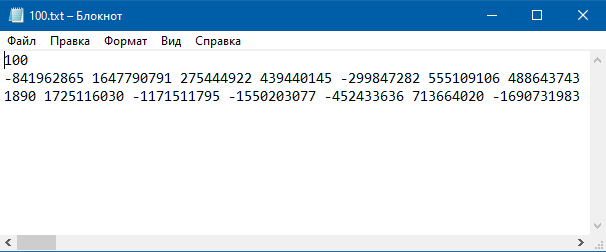
**Важно**:

Для корректного отображения содержимого файла формат внутренних данных должен быть следующий:

1 Строка - Количество элементов

2 Строка - Исходный массив

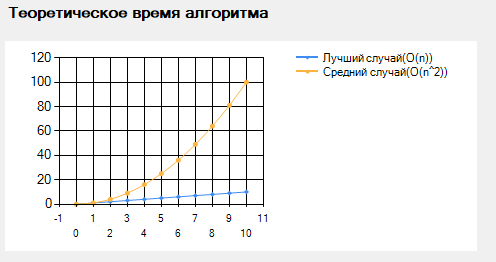
Пример:



Справа от основной части программы расположены два графика:

1. График теоретического времени работы сортировки
2. График реального времени работы сортировки

Первый график неизменен. Он показывает время работы сортировки в лучшем и среднем случаях.



Второй график изменяется и уточняется в зависимости от введенных данных.



# **Руководство программиста**

При разработке ПО отдельные классы не создавались, поэтому весь функционал прописан в основном классе MainForm.

Глобальные переменные:

* int countGlob - длина исходного массива;
* int[] nums – исходный массив;
* Dictionary<int,int> realChart = new Dictionary<int, int> {} – словарь реального времени сортировки данных, в котором key – количество элементов, value – время сортировки.

При запуске формы вызывается метод MainFor\_Load. В нем вызываются следующие функции:

1. TeoreticChartFill() – отвечающая за заполнение графика теоретического времени работы сортировки:

private void TeoreticChartFill()

{

for (int i = 0; i <= 10; i+=1)

{

chartTeoreticTime.Series["Средний случай(O(n^2))"].Points.AddXY(i, i\*i);

chartTeoreticTime.Series["Лучший случай(O(n))"].Points.AddXY(i, i);

}

}

1. RealChartFill() – отвечающая за заполнение графика реального времени работы сортировки:

private void RealChartFill()

{

var sortedDict = new SortedDictionary<int, int>(realChart);

chartRealTime.Series["Время"].Points.Clear();

foreach (var item in sortedDict)

{

chartRealTime.Series["Время"].Points.AddXY(item.Key, item.Value);

}

}

После заполнения полей «Количество элементов» и «Исходный массив» и нажатия на кнопку Sort вызывается метод buttonSort\_Click, в котором происходят следующие действия:

* countGlob = int.Parse(textBoxCount.Text) – присваивание переменной countGlob содержимого textBoxCount;
* TransformToIntMass() – вызов функции отвечающей за изъятие данных из textBoxNums в массив nums:

private void TransformToIntMass()

{

nums = new int[countGlob];

string[] textmas = textBoxNums.Text.Split();

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

{

nums[i] = int.Parse(textmas[i]);

}

}

* shakerSort() – вызов функции сортировки, подсчет времени ее выполнения и вывод времени в textBoxTime:

private void shakerSort()

{

Stopwatch stopWatch = new Stopwatch();//Таймер, замеряющий время выполнения сортировки

stopWatch.Start(); //Запуск таймера

int left = 0, right = countGlob - 1; // левая и правая границы сортируемой области массива

int flag = 1; // флаг наличия перемещений

int t;// вспомогательная переменная

// Цикл выполняется пока левая граница не сомкнётся с правой и пока в массиве имеются перемещения

while ((left < right) && flag > 0)

{

flag = 0;

for (int i = left; i < right; i++) //двигаемся слева направо

{

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

{

t = nums[i];

nums[i] = nums[i + 1];

nums[i + 1] = t;

flag = 1;

}

}

right--; // сдвигаем правую границу на предыдущий элемент

for (int i = right; i > left; i--) //двигаемся справа налево

{

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

{

t = nums[i];

nums[i] = nums[i - 1];

nums[i - 1] = t;

flag = 1;

}

}

left++; // сдвигаем левую границу на следующий элемент

}

stopWatch.Stop();//Остановка таймера

// Задаю формат и вывожу время с таймера

TimeSpan ts = stopWatch.Elapsed;

string elapsedTime = String.Format("{0:00}:{1:00}:{2:00}.{3:000}",

ts.Hours, ts.Minutes, ts.Seconds,

ts.Milliseconds);

textBoxTime.Text = elapsedTime;

//Перевожу время с таймера в миллисекунды

int milisec = ts.Seconds \* 1000 + ts.Milliseconds;

//Добавляю полученные данные в словарь realChart

if (!realChart.ContainsKey(countGlob))

realChart.Add(countGlob, milisec);

else realChart[countGlob] = milisec;

}

* AddNumsToTextBoxAfterSort() – вывод отсортированного массива на textBoxAfterSort:

private void AddNumsToTextBoxAfterSort()

{

textBoxAfterSort.Clear();

string s="";

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

{

s += nums[i] + " ";

}

textBoxAfterSort.Text = s;

}

* RealChartFill() – изменение графика реального времени сортировки, в соответствии с новыми данными, полученными после последней сортировки

Реализация случайного заполнения:

При нажатии на кнопку Random вызывается метод buttonRandom\_Click. В нем заполняются textBoxCount и textBoxNums случайными числами:

private void buttonRandom\_Click(object sender, EventArgs e)

{

textBoxNums.Clear();

Random rd = new Random();

textBoxCount.Text = rd.Next(100,20000).ToString();

countGlob = int.Parse(textBoxCount.Text);

string s="";

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

{

s+= rd.Next(int.MinValue, int.MaxValue) +" ";

}

textBoxNums.Text = s;

}

Реализация импорта данных из файла:

При нажатии на кнопку «Импорт из файла» вызывается метод ImportFromFileToolStripMenuItem\_Click.

Описание метода:

* Открывается окно проводника(настроен фильтр на файлы формата .txt)

var excelOpen = new OpenFileDialog

{

Filter = "|\*.txt"

};

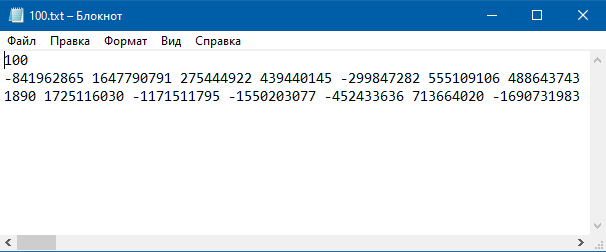
**Важно**:

Для корректного отображения содержимого файла формат внутренних данных должен быть следующий:

1 Строка - Количество элементов

2 Строка - Исходный массив

Пример:



* После открытия необходимого файла считываем данные, заполняя textBoxCount и textBoxNums соответственно

if (excelOpen.ShowDialog() == DialogResult.OK)

{

using (StreamReader sr = new StreamReader(excelOpen.FileName))

{

textBoxCount.Text = sr.ReadLine();

textBoxNums.Text = sr.ReadToEnd();

}

}

Реализация графиков:

В приложении реализованы два графика:

1. – график теоретического времени работы сортировки

2. – график реального времени работы сортировки

Первый график реализован на chartTeoreticTime. В нем две коллекции series "Средний случай(O(n^2))" и "Лучший случай(O(n))". График заполняется один раз при запуске программы в методе TeoreticChartFill.

Второй график реализован на chartRealTime. В нем одна коллекция "Время". График строится по Dictionary<int,int> realChart. Словарь обновляется при каждой сортировке, добавляя или изменяя содержимое.

# **Список используемой литературы**

1. Д.Кнут: Искусство программирования. Том 3. Сортировка и поиск. 2007г.
2. <https://metanit.com/sharp/tutorial/> Полное руководство по языку программирования С# 8.0 и платформе .NET Core 3
3. <https://kvodo.ru/shaker-sort.html> Шейкерная сортировка(Перемешиванием)