**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

Отчёт

**по учебной практике**

**Тема: Визуализация алгоритма на Java**

|  |  |  |
| --- | --- | --- |
| Студент гр.5304 |  | Калганов Р.Г. |
| Студент гр.5304 |  | Круглик А.Д. |
| Студентка гр.5304 |  | Русу Я. |
| Руководитель |  |  |

Санкт-Петербург

2017

**ЗАДАНИЕ**

**на Учебную практику**

|  |  |  |
| --- | --- | --- |
| Студент Калганов Р.Г. группы 5304 | | |
| Студент Круглик А.Д. группы 5304 | | |
| Студентка Русу Я. группы 5304  Тема практики: Визуализация алгоритма на Java | | |
| Задание на практику:  Командная итеративная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: Сортировка перемешиванием | | |
| Сроки прохождения практики: 21.06.2017 – 04.07.2017 | | |
| Дата сдачи отчета: 26.06.2017 | | |
| Дата защиты отчета: 26.06.2017 | | |
|  | | |
| Студент |  | Калганов Р.Г. |
| Студент |  | Круглик А.Д. |
| Студентка |  | Русу Я. |
| Руководитель |  | . |

**Аннотация**

В ходе работы над проектом учебной практики необходимо реализовать визуализацию выбранного алгоритма на языке программирования Java. Визуализация алгоритма должна быть понятной для пользователей, а интерфейс программы удобным.

Проект выполняется в команде, что способствует развитию навыков групповой работы. Базовым условием реализации программы в команде является наличие плана разработки и разделения обязанностей.

**Summary**

In the course of work on the educational practice, it is necessary to visualize the chosen algorithm in the Java programming language. Visualization of the algorithm should be understandable for users, and the program interface is convenient.

The project is carried out in a team, which promotes the development of group work skills.The basic condition for implementing the program in the team is the availability of a plan for the development and division of duties.

**Содержание**

[Введение 5](#_Toc9361)

[1. Требования к программе 6](#_Toc30445)

[1.1. Описание основной задачи программы 6](#_Toc13078)

[1.2. Формат входных данных и результат работы программы 6](#_Toc29780)

[1.3. Эскизы интерфейса 7](#_Toc24863)

[1.4. Общий алгоритм работы программы в соответствии с эскизом интерфейса 10](#_Toc5146)

[2. План разработки и распределение ролей в бригаде 11](#_Toc11840)

[2.1. План разработки 11](#_Toc10493)

[2.2. Разделение ролей в бригаде 12](#_Toc30186)

[3. Особенности реализации 13](#_Toc19876)

[3.1. Использованные структуры данных 13](#_Toc26913)

[3.2. Описание интерфейса 14](#_Toc21532)

[4. Тестирование 15](#_Toc10890)

[4.1. Тестирование графического интерфейса 15](#_Toc12816)

[4.2. Тестирование программы с некорректными данными 17](#_Toc18490)

[4.3. Тестирование работы программы при пошаговом режиме 19](#_Toc22523)

[Заключение 21](#_Toc26799)

[Список использованных источников 22](#_Toc4768)

[Приложение A. Код программы 23](#_Toc27861)

# **Введение**

В процессе выполнения практики необходимо разработать программу на языке Java, предназначенную для осуществления сортировки перемешиванием. Также требуется реализовать пользовательский интерфейс и визуализацию алгоритма сортировки. При этом визуализация должна быть понятной для пользователя и иметь возможность пошагового представления работы алгоритма. Программа должна иметь способность выдавать результат без процесса визуализации алгоритмов сортировки, а также принимать значения заносимые пользователем или генерировать их самостоятельно, в соответствии с количеством элементов.

Алгоритм сортировки перемешиванием – это развитие [пузырьковой сортировки](http://space-base.ru/library/?book=13). Отличия от неё заключаются в том, что при прохождении части массива, происходит проверка, были ли перестановки. Если их не было, значит, эта часть массива уже упорядочена и она исключается из дальнейшей обработки. Кроме того, при прохождении массива от начала к концу, минимальные элементы перемещаются в самое начало, а максимальный элемент сдвигается к концу массива.

# **Требования к программе**

# **1.1. Описание основной задачи программы**

Программа предназначена для визуального представления сортировки перемешиванием. Необходимо реализовать алгоритм данной сортировки, которая представляет собой разновидность [пузырьковой](http://dic.academic.ru/dic.nsf/ruwiki/45663). Отличается она тем, что просмотры элементов выполняются один за другим в противоположных направлениях, при этом большие элементы стремятся к концу массива, а маленькие - к началу.

Программа должна иметь удобный интерфейс и понятный способ визуализации. Для наглядного изображения отличия работы сортировки перемешиванием от других обменных сортировок программа должна предоставлять возможность визуализации других алгоритмов, таких как:

* Пузырьковая сортировка
* Глупая сортировка
* Быстрая сортировка

# **1.2. Формат входных данных и результат работы программы**

На вход программа получает количество и массив элементов. Элементы могут быть заданы пользователем, либо образованы с помощью генерации случайных чисел.

Ограничение значении элементов: от 0 до 1000.

Ограничение количества элементов: от 1 до 65.

В результате работы, программы должна, основываясь на введённой последовательности элементов, визуально изобразить работу сортировки и предоставить отсортированный массив чисел.

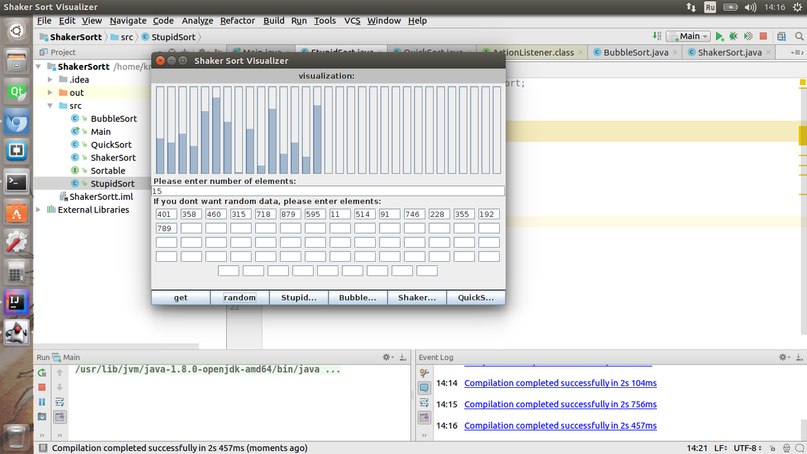
# **1.3. Эскизы интерфейса**

Интерфейс программы должен быть удобен в использовании, поэтому для достижения данной цели принимаются следующие меры:

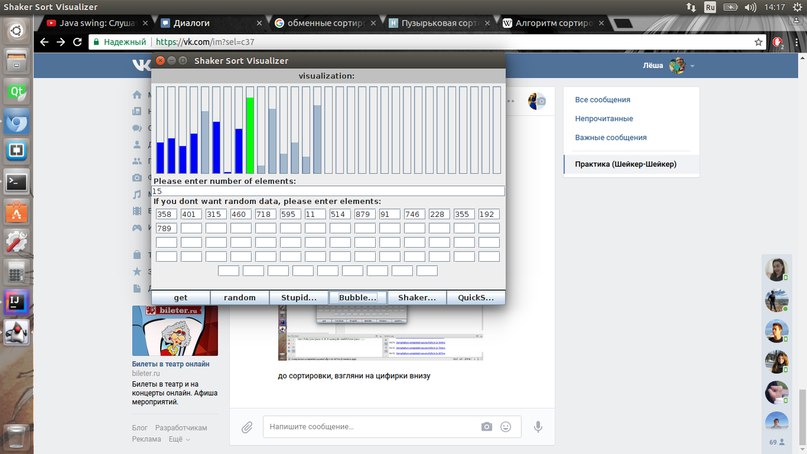
* Создание трёх режимов работы программы:
  + - Работа алгоритмов сортировки без визуализации
    - Работа алгоритмов сортировки с визуализацией без отработки по шагам
    - Работа алгоритмов сортировки по шагам (наличие возможности вернуться на шаг назад или перейти вперёд)
* Каждое окно для ввода данных подписывается по назначению
* Организовывается панель внизу, содержащая кнопки get, которая берёт введённые пользователем элементы, и random, которая отвечает за подбор случайных элементов последовательности
* На панели внизу также располагаются кнопки, запускающие алгоритмы сортировки. Данные кнопки подписаны в соответствии с названием необходимого алгоритма
* Область визуализации организовывается в верхней части основного окна программы и представлена в виде столбиков, количество которых соответствует количеству элементов, а уровень окраски -величине значения элемента на данной позиции. Кроме того, на каждом столбце прописывается значение, которое он символизирует
* При изменениях количества элементов столбцы изменяют свою ширину и ориентацию на фоне
* В процессе работы алгоритмов сортировки изменение положений элементов сопровождается наличием стрелки, которая определяет какие элементы и как меняют своё положение

Эскизы интерфейса представлены на рисунках 1 – 4:

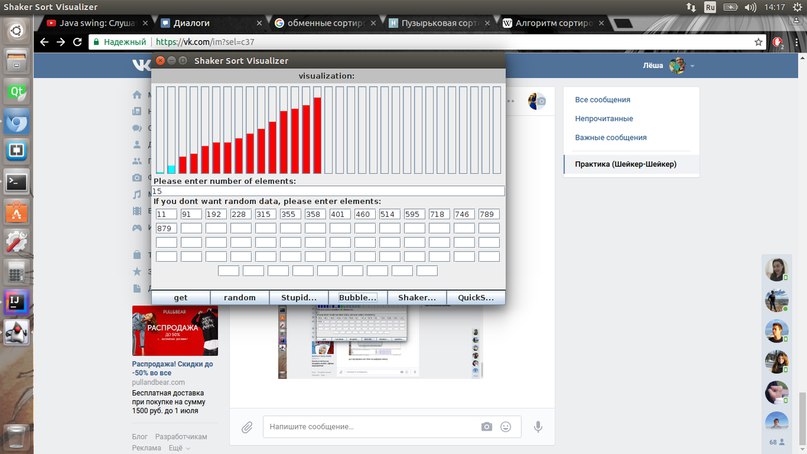
*Рисунок №1: Основное окно программы после ввода количества элементов и генерации последовательности случайных чисел*



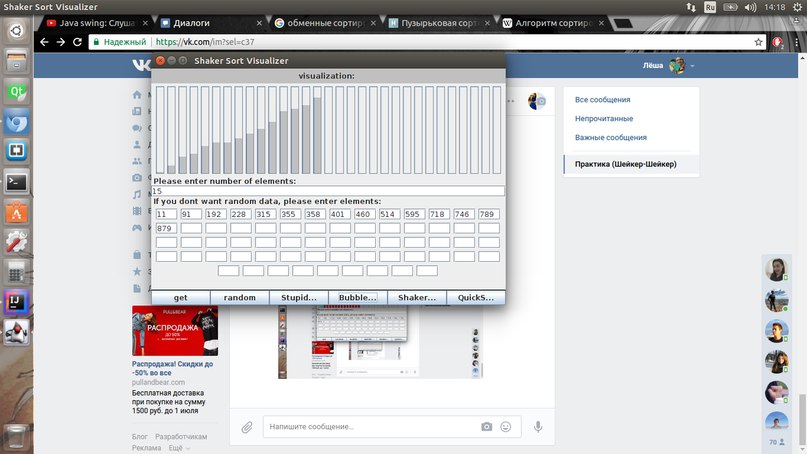
*Рисунок №2: Основное окно программы после выбора алгоритма (в процессе визуализации сортировки)*



*Рисунок №3: Основное окно программы при достижении правильной (отсортированной) последовательности*

**

*Рисунок №4: Основное окно программы после окончания работы данного алгоритма сортировки*

**

# **1.4. Общий алгоритм работы программы в соответствии с эскизом интерфейса**

* Получаем количество элементов для сортировки
* Получаем начальные значения с помощью генерации случайных чисел, либо при помощи ввода необходимых значений пользователем
* Исходя из начальных значений генерируем массив элементов и ожидаем команды от пользователя
* При нажатии пользователем кнопки:
  + - «Stupid sort» запускаем глупую сортировку
    - «Bubble sort» запускаем сортировку пузырьком
    - «Shaker sort» запускаем сортировку перемешиванием
    - «Quick sort» запускаем быструю сортировку
* После запуска одной из сортировок, производим визуализацию её алгоритма:
  + - Первым (синим) цветом подсвечиваются элементы, которые были отсортированы
    - Вторым (зелёным) цветом подсвечиваются текущие элементы, которые сортируются
    - Третьим (серым) цветом подсвечиваются элементы, которые ещё не сортировались
* После окончания работы сортировки все элементы окрашиваем в новый цвет (красный), проходимся по ним другим цветом (бирюзовым) для визуального подтверждения того, что сортировка окончена, и возвращаемся к серому цвету элементов.

# **План разработки и распределение ролей в бригаде**

# **2.1. План разработки**

План разработки и реализации программы представлен в таблице 2:

*Таблица №2: План разработки*

|  |  |  |  |
| --- | --- | --- | --- |
| Номер этапа работы | Дата начала этапа | Дата окончание этапа | Назначение этапа (описание работа, проводимых в данный промежуток времени) |
| 1 | 21.06.17 | 23.06.17 | * Определение темы проекта * Написание спецификации * Составление плана разработки * Распределение ответственности |
| 2 | 23.06.17 | 26.06.17 | * Создание прототипа программы, демонстрирующего пользовательский интерфейс |
| 3 | 26.06.17 | 28.06.17 | * Создание рабочей программы для визуализации сортировок:   + Программа должна работать, на оттестированных данных   + Должны быть реализованы все классы и методы, прописан интерфейс * Поиск проблем и ошибок в работе программы, в том числе проблемы отсутствия обработки некорректных данных |
| 4 | 28.06.17 | 30.06.17 | * Исправление всех ошибок * Создание финальной программы |

# **2.2. Разделение ролей в бригаде**

Разделение обязанностей по членам команды представлено в таблице 3:

*Таблица №3: Разделение обязанностей*

|  |  |
| --- | --- |
| ФИО | Обязанности |
| Калганов Р.Г. | Создание интерфейса и визуализации |
| Круглик А.Д. | Разработка архитектуры проекта, координация действии, проверка деятельности других членов команды |
| Русу Я. | Реализация алгоритмов сортировки |

# **Особенности реализации**

# **3.1. Использованные структуры данных**

Для создания GUI использовалась библиотека Swing, а именно следующие графические виджеты:

* JFrame – графическое полотно для виджетов и элементов отрисовки
* JPanel – графическая группировка отдельных виджетов и элементов на полотне
* JTextField – текстовое поле
* JButton - кнопка
* JComboBox – выпадающий список

Для создания собственной графики использовались примитивы для отрисовки одиночных линий и наследуемый от JPanel собственный класс GraphicsPanel. Для программного взаимодействия с созданным GUI используются ряд наследуемых от ActionListener классов:

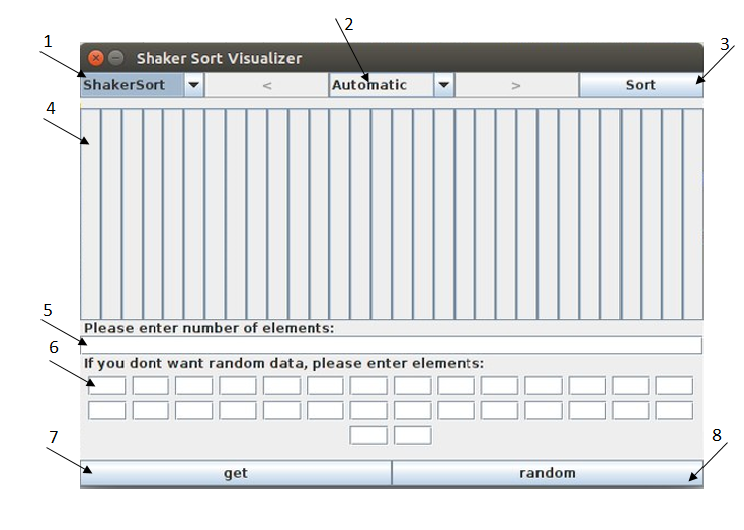
* AnotherSortButtonActionListener – взаимодействие с кнопкой начала сортировки
* GetElButtonActionListener – взаимодействие с кнопкой, задающей группу чисел для сортировки
* RandomButtonActionListener – взаимодействие с кнопкой случайной генерации чисел
* nextStepActionListener и prevStepActionListener – взаимодействие с кнопками пошаговой сортировки

Помимо графических компонент использовались обыкновенные массивы чисел, строк, и.т.п.

# **3.2. Описание интерфейса**

Описание интерфейса программы представлено в таблице 1:

*Рисунок №5: Основное окно программы*



*Таблица №1: Описание интерфейса*

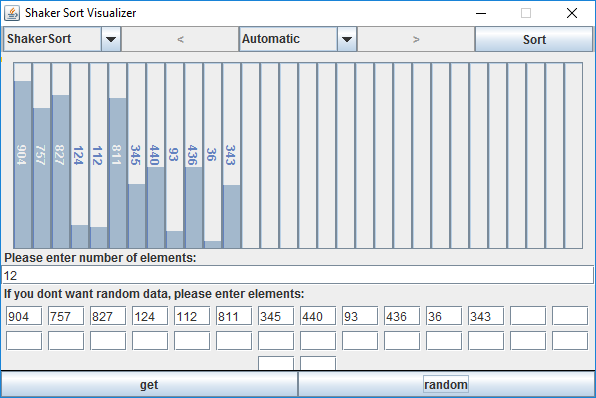
|  |  |
| --- | --- |
| Номер | Назначение |
| 1 | Кнопка для выбора алгоритма сортировки |
| 2 | Кнопка для выбора режима визуализации |
| 3 | Кнопка для запуска выбранного способа сортировки |
| 4 | Область визуализации сортировки |
| 5 | Окно для ввода количества элементов |
| 6 | Окно для ввода значений элементов |
| 7 | Кнопка, которая берёт введённые пользователем элементы |
| 8 | Кнопка, которая отвечает за подбор случайных элементов последовательности |

# **Тестирование**

# **4.1. Тестирование графического интерфейса**

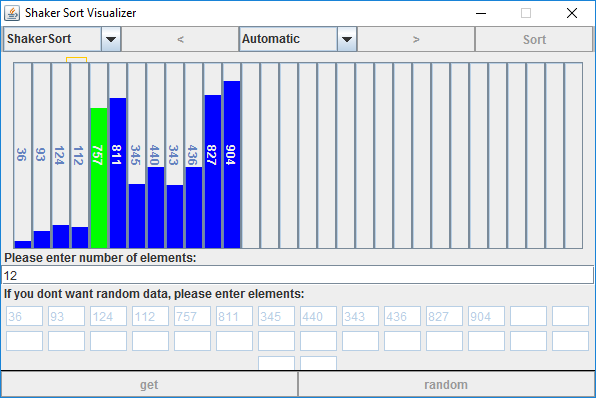
Откроем программу и зададим случайный набор из 12 чисел при помощи кнопки random и текстовой строки:

*Рисунок №6: Основное окно программы*

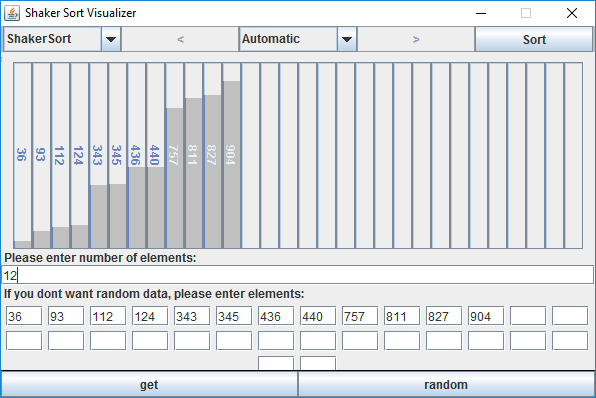


Запускаем процесс сортировки:

*Рисунок №7: Процесс сортировки перемешиванием*



*Рисунок №8: Окончание сортировки*



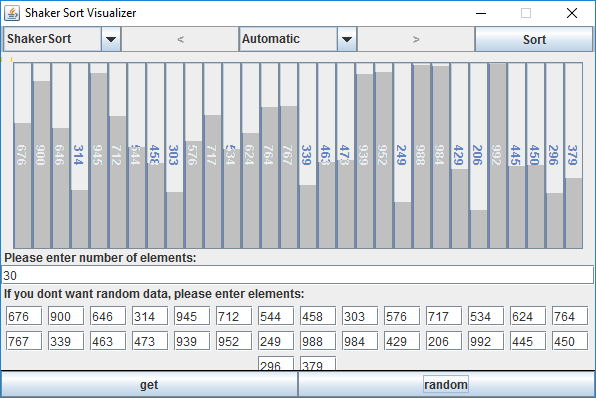
# **4.2. Тестирование программы с некорректными данными**

Попытаемся задать больше 30 чисел и запустим алгоритм «глупой» сортировки:

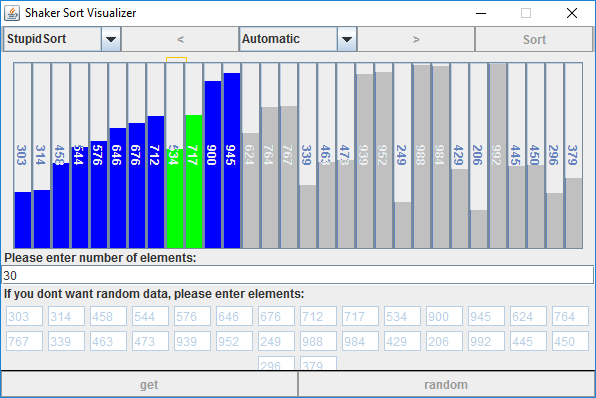
*Рисунок №9: Попытка ввода некорректных данных (программа*

*автоматически изменяет некорректные данные на максимально*

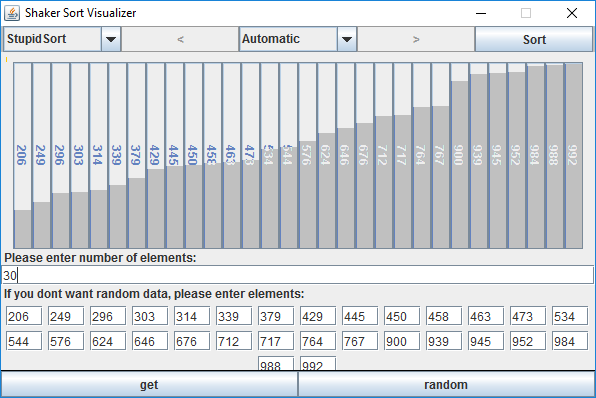
*возможные)*



*Рисунок №10: «Глупая» сортировка*



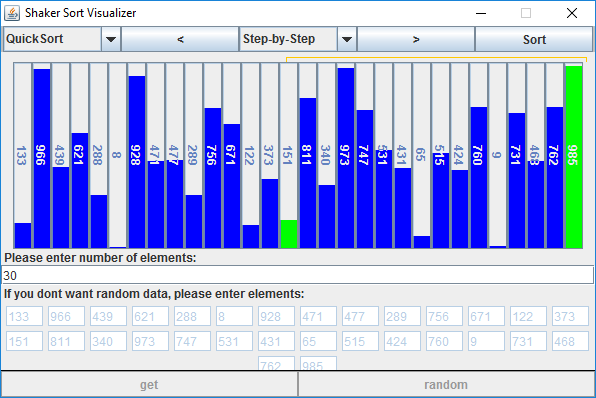
*Рисунок №11: Окончание сортировки*



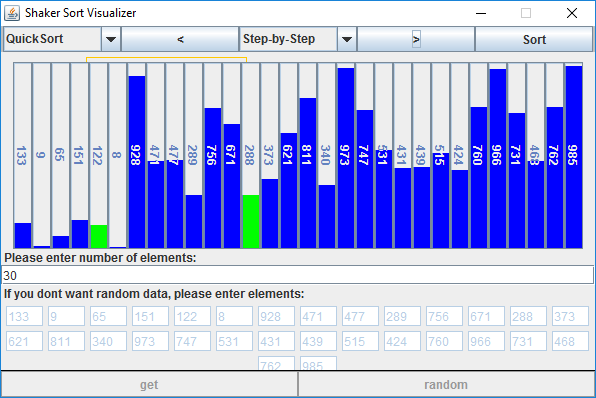
# **4.3. Тестирование работы программы при пошаговом режиме**

Протестируем пошаговый режим сортировки на 30 элементах для быстрой сортировки:

*Рисунок №12: Первый шаг пошаговой быстрой сортировки*

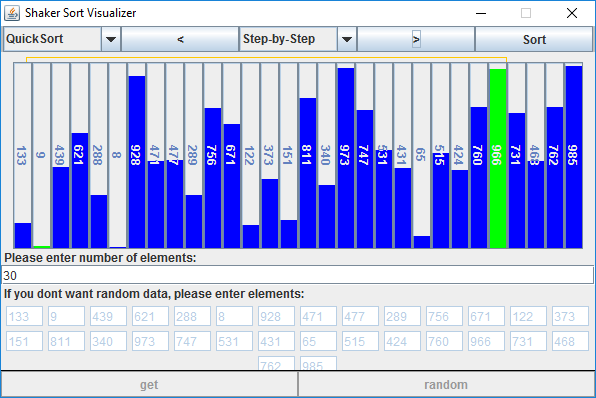


*Рисунок №13: Четвёртый шаг пошаговой быстрой сортировки*



Вернёмся к второму шагу:

*Рисунок №14: Второй шаг пошаговой быстрой сортировки*



# **Заключение**

В результате работы над данным проектом была написана программа, успешно справляющаяся со своей основной задачей. Были реализованы алгоритм сортировки перемешиванием и его визуализация.

Дополнительно программа реализует ещё семь распространённых алгоритмов сортировки для наглядного изображения отличия сортировки перемешиванием от других.

В ходе разработки программы были получены навыки совместной работы и разделения обязанностей.

# **Список использованных источников**

1. Герберт Шилдт: Java 8. Полное руководство. 9-е издание :: Издательство: Вильямс, 2015.1300 с.
2. Хабрахабр. URL:<https://habrahabr.ru/post/204600/> (Дата обращения: 26.06.2017 - «Пузырьковая сортировка и все-все-все»)
3. Java документация. URL:<https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html> (Дата обращения: 25.06.2017 - «Package javax.swing»)
4. idea intellij документация. URL:http://www.jetbrains.com/idea/documentation/ (Дата обращения: 24.06.2017 - «Learn and Support»)

# **Приложение A. Код программы**

**BubbleSort.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class BubbleSort implements Sortable {

protected int numb;

protected int[] array;

BubbleSort(int n,int[] arr){

numb = n;

array = arr;

}

@Override

public Vector<Integer> sort(int step){

int nowStep=0;

Vector<Integer> result = new Vector<Integer>();

if (step!=0) {

for (Integer i = numb - 1; i >= 0; i--)

for (Integer j = 0; j < i; ++j)

if (array[j] > array[j + 1]) {

swap(array, j, j + 1);

result.add(j);

result.add(j + 1);

++nowStep;

if (nowStep == step) return result;

}

}

return result;

}

@Override

public void swap(int[] array,int left,int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**GraphicPanel.java**

import javax.swing.\*;

import java.awt.Color;

import java.awt.Graphics;

import java.awt.\*;

/\*\*

\* Created by root on 23.06.17.

\*/

public class GraphicPanel extends JPanel{//это рисует

Graphics2D g2;

int xStart = -5;

int xEnd =-5;

public void setData(int startX,int endX){

xStart = startX+10-(startX/19);

xEnd = endX+10-(endX/19);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

g2 = (Graphics2D) g;

g2.setColor(Color.ORANGE);

xStart += 10;

xEnd += 10;

g2.drawLine(xStart,5,xEnd,5);

g2.drawLine(xStart,5,xStart,10);

g2.drawLine(xEnd,5,xEnd,10);

}

};

InsertionSort.java

import java.util.Vector;

/\*\*

\* Created by root on 23.06.17.

\*/

public class InsertionSort implements Sortable {

protected int numb;

protected int[] array;

InsertionSort(int n,int[] arr){

numb = n;

array = arr;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

@Override

public Vector<Integer> sort(int step) {

Vector<Integer> result =new Vector<Integer>();

int nowStep = 0;

if (step!=0){

int j;

for (int i = 1;i<numb;++i){

j = i;

while (j>0&&array[j-1]>array[j]){

result.add(j-1);

result.add(j);

swap(array,j-1,j);

++nowStep;

if (nowStep==step) return result;

--j;

}

}}

return result;

}

}

**MergeSort.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class MergeSort implements Sortable {

protected int numb;

protected int[] array;

protected int[] buffer;

MergeSort(int n,int[] arr){

numb = n;

array = arr;

buffer = new int[n];

}

private void merge(int left,int right,Vector<Integer> result,int step,int nowStep){

if (nowStep==step) return;

if (left==right) return;

if ((right - left)==1) {

if (array[left]>array[right]){

result.add(array[right]);

result.add(left);

++nowStep;

if (nowStep==step) return;

result.add(array[left]);

result.add(right);

++nowStep;

if (nowStep==step) return;

swap(array,left,right);

return;}

}

int middle = (left+right)/2;

merge(left,middle,result,step,nowStep);

merge(middle+1,right,result,step,nowStep);

int xLeft = left;

int xRight = middle+1;

int current = 0;

while ((right-left+1)!=current){

if (xLeft>middle){

buffer[current++] = array[xRight++];

} else if (xRight>right){

buffer[current++] = array[xLeft++];

} else if (array[xLeft]>array[xRight]){

buffer[current++] = array[xRight++];

} else {

buffer[current++] = array[xLeft++];

}

}

for (int i = 0; i<current;++i){

result.add(buffer[i]);

result.add(left+i);

array[left+i] = buffer[i];

++nowStep;

if (nowStep==step) return;

}

return;

}

@Override

public Vector<Integer> sort(int step) {

Vector<Integer> result = new Vector<Integer>();

if (step!=0){

merge(0,numb-1,result,step,0);}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**QuickSort.java**

import com.sun.xml.internal.bind.v2.model.annotation.Quick;

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class QuickSort implements Sortable {

protected int numb;

protected int[] array;

QuickSort(int n,int[] arr){

numb = n;

array = arr;

}

public void QuickSort(int left, int right,Vector<Integer> result,int step,int nowStep){

if (nowStep==step) return;

int middle = array[(left+right)/2];

int k = 0;

int i = left;

int j = right;

while (i<=j){

while (array[i]<middle) ++i;

while (array[j]>middle) --j;

if (i<=j){

result.add(i);

result.add(j);

swap(array,i,j);

++nowStep;

if (nowStep==step) return;

++i;

--j;

}

}

if (i<right) QuickSort(i,right,result,step,nowStep);

if (left<j) QuickSort(left,j,result,step,nowStep);

return;

}

@Override

public Vector<Integer> sort(int step) {

Vector<Integer> result = new Vector<Integer>();

if (step!=0){

QuickSort(0,numb-1,result,step,0);}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**SelectionSort.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class SelectionSort implements Sortable {

protected int numb;

protected int[] array;

SelectionSort(int n,int[] arr){

numb = n;

array = arr;

}

@Override

public Vector<Integer> sort(int step) {

int nowStep = 0;

Vector<Integer> result = new Vector<Integer>();

if (step!=0) {

for (int i = 0; i < numb - 1; ++i) {

int least = i;

for (int j = i + 1; j < numb; j++) {

if (array[j] < array[least]) {

least = j;

}

}

result.add(i);

result.add(least);

swap(array, i, least);

++nowStep;

if (nowStep == step) return result;

}

}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**ShakerSort.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class ShakerSort implements Sortable {

protected int numb;

protected int[] array;

ShakerSort(int n,int[] arr){

numb = n;

array = arr;

}

@Override

public Vector<Integer> sort(int step){

int nowStep = 0;

Vector<Integer> result = new Vector<Integer>();

int left = 0;

int right = numb - 1;

if (step!=0) {

while (left <= right) {

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

if (array[i] > array[i + 1]) {

result.add(i);

result.add(i + 1);

swap(array, i, i + 1);

++nowStep;

if (nowStep == step) return result;

}

}

--right;

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

if (array[i - 1] > array[i]) {

result.add(i - 1);

result.add(i);

swap(array, i - 1, i);

++nowStep;

if (nowStep == step) return result;

}

}

left++;

}

}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**ShellSort.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class ShellSort implements Sortable {

protected int numb;

protected int[] array;

ShellSort(int n,int[] arr){

numb = n;

array = arr;

}

@Override

public Vector<Integer> sort(int step) {

int nowStep = 0;

Vector<Integer> result = new Vector<Integer>();

int flag = 0,i=0,t=0,j = 0;

if (step!=0){

for (flag = numb;flag>0;flag/=2) {

for (i = flag;i<numb;++i){

t = array[i];

for (j = i; j >= flag && t < array[j - flag]; j -= flag){

result.add(array[j-flag]);

result.add(j);

array[j] = array[j-flag];

}

result.add(t);

result.add(j);

array[j]=t;

++nowStep;

if (nowStep==step) return result;

}

}}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**Sortable.java**

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public interface Sortable {

Vector<Integer> sort(int step);

void swap(int[] array, int left, int right);

}

**StupidSort.java**

import org.apache.tools.ant.types.resources.Sort;

import java.util.Vector;

/\*\*

\* Created by root on 22.06.17.

\*/

public class StupidSort implements Sortable{

protected int numb;

protected int[] array;

StupidSort(int numberOfEl,int[] arr){

numb = numberOfEl;

array = arr;

}

@Override

public Vector<Integer> sort(int step) {

int nowStep = 0;

Vector<Integer> result = new Vector<Integer>();

int flag = 0;

int k = 0;

if (step!=0){

while (flag!=(numb-1)){

if (array[flag]>array[flag+1]){

result.add(flag);

result.add(flag+1);

swap(array,flag,flag+1);

++nowStep;

if (nowStep==step) return result;

flag = 0;

} else ++flag;

}}

return result;

}

@Override

public void swap(int[] array, int left, int right) {

int k = 0;

k = array[left];

array[left] = array[right];

array[right] = k;

}

}

**Main.java**

import javax.sound.sampled.Line;

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.Random;

import java.util.Vector;

/\*\*

\* Created by root on 21.06.17.

\*/

public class Main {

static int numberOfSortedEl;

static final int speed = 5;

static final int speedB = 5;

static final int maxElem = 30;

static final int minRange = 0;

static final int maxRange = 1000;

static final int height = 600;

static final int width = 400;

static final int lengthElem = 20;

static int step = 1;

static JFrame mainWindow = new JFrame();

static JPanel visualizePanel = new JPanel();

static JPanel secVisPanel = new JPanel();

static GraphicPanel uppSecVisPanel = new GraphicPanel();

static JPanel centerSecVisPanel = new JPanel();

static JPanel firstVisPanel = new JPanel();

static JPanel secondWorkPanel = new JPanel();

static JPanel thirdWorkPanel = new JPanel();

static JPanel fourWorkPanel = new JPanel();

static JPanel fiveWorkPanel = new JPanel();

static JPanel sixWorkPanel = new JPanel();

static JPanel workPanel = new JPanel();

static JPanel upperPanel = new JPanel();

static JTextField numbOfElement = new JTextField(2);

static JTextField[] ourArray = new JTextField[maxElem];

static JProgressBar[] sortedEl = new JProgressBar[maxElem];

static int[] sortedArray = new int[maxElem];

static JButton anotherSortButt = new JButton("Sort");

static JButton nextStepSortButt = new JButton(">");

static JButton prevStepSortButt = new JButton("<");

static JComboBox<String> typeOfSort = new JComboBox<String>();

static JComboBox<String> sortBox = new JComboBox<String>();

static int[] firstSafedArray = new int[maxElem];

static JButton randomButt = new JButton("random");

static JButton getButt = new JButton("get");

public static void main(String[] args) {

mainWindow.setSize(height, width);

mainWindow.setTitle("Shaker Sort Visualizer");

mainWindow.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

mainWindow.setMaximumSize(new Dimension(height, width));

mainWindow.setMinimumSize(new Dimension(height, width));

mainWindow.setResizable(false);

mainWindow.setLayout(new BorderLayout());

workPanel.setLayout(new GridLayout());

workPanel.setBackground(Color.LIGHT\_GRAY);

visualizePanel.setBackground(Color.black);

sortBox.addItem("ShakerSort");

sortBox.addItem("MergeSort");

sortBox.addItem("ShellSort");

sortBox.addItem("SelectionSort");

sortBox.addItem("InsertionSort");

sortBox.addItem("BubbleSort");

sortBox.addItem("QuickSort");

sortBox.addItem("StupidSort");

typeOfSort.addItem("Automatic");

typeOfSort.addItem("Step-by-Step");

typeOfSort.addItem("Result");

nextStepSortButt.setEnabled(false);

prevStepSortButt.setEnabled(false);

anotherSortButt.addActionListener(new AnotherSortButtonActionListener());

nextStepSortButt.addActionListener(new nextStepActionListener());

prevStepSortButt.addActionListener(new prevStepActionListener());

centerSecVisPanel.setLayout(new BorderLayout());

centerSecVisPanel.add(secVisPanel, BorderLayout.CENTER);

centerSecVisPanel.add(uppSecVisPanel, BorderLayout.PAGE\_START);

secVisPanel.setLayout(new GridLayout());

getButt.addActionListener(new GetElButtonActionListener());

randomButt.addActionListener(new RandomButtonActionListener());

JLabel enterNumbOfEl = new JLabel(" Please enter number of elements: ");

workPanel.add(getButt);

workPanel.add(randomButt);

secVisPanel.add(prevStepSortButt);

for (int i = 0; i < maxElem; ++i) {

ourArray[i] = new JTextField(3);

sortedEl[i] = new JProgressBar();

sortedEl[i].setMinimum(minRange);

sortedEl[i].setMaximum(maxRange);

sortedEl[i].setValue(minRange);

sortedEl[i].setOrientation(JProgressBar.VERTICAL);

sortedEl[i].setStringPainted(true);

sortedEl[i].setString("");

sortedEl[i].setPreferredSize(new Dimension(20,130));

secVisPanel.add(sortedEl[i]);

}

secVisPanel.add(nextStepSortButt);

visualizePanel.setLayout(new BoxLayout(visualizePanel, BoxLayout.PAGE\_AXIS));

visualizePanel.add(centerSecVisPanel);

JLabel enterArrEl = new JLabel(" If you dont want random data, please enter elements: ");

firstVisPanel.setLayout(new BorderLayout());

secondWorkPanel.setLayout(new BorderLayout());

thirdWorkPanel.setLayout(new BorderLayout());

fourWorkPanel.setLayout(new BorderLayout());

fiveWorkPanel.setLayout(new FlowLayout());

sixWorkPanel.setLayout(new BorderLayout());

sixWorkPanel.add(fiveWorkPanel);

fourWorkPanel.add(sixWorkPanel, BorderLayout.CENTER);

thirdWorkPanel.add(fourWorkPanel, BorderLayout.CENTER);

secondWorkPanel.add(thirdWorkPanel, BorderLayout.CENTER);

firstVisPanel.add(secondWorkPanel, BorderLayout.CENTER);

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

fiveWorkPanel.add(ourArray[i]);

thirdWorkPanel.add(enterNumbOfEl, BorderLayout.NORTH);

fourWorkPanel.add(numbOfElement, BorderLayout.NORTH);

sixWorkPanel.add(enterArrEl, BorderLayout.NORTH);

visualizePanel.add(firstVisPanel, BorderLayout.CENTER);

JLabel upper = new JLabel(" visualization: ");

upperPanel.setLayout(new GridLayout());

upperPanel.setBackground(Color.LIGHT\_GRAY);

upperPanel.add(sortBox);

upperPanel.add(prevStepSortButt);

upperPanel.add(typeOfSort);

upperPanel.add(nextStepSortButt);

upperPanel.add(anotherSortButt);

mainWindow.add(visualizePanel, BorderLayout.CENTER);

mainWindow.add(workPanel, BorderLayout.SOUTH);

mainWindow.add(upperPanel, BorderLayout.NORTH);

mainWindow.setVisible(true);

}

public static void move(Vector<Integer> result) {

Thread myThready = new Thread(new Runnable() {

@Override

public void run() {

synchronized (this) {

lock();

int k = 0;

int flag = 1;

Integer j = 0;

for (Integer i : result) {

if (flag % 2 == 0) {

uppSecVisPanel.setData(i \* lengthElem, j \* lengthElem);

uppSecVisPanel.repaint();

++flag;

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedEl[j].setForeground(Color.green);

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedEl[i].setForeground(Color.green);

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

k = sortedArray[j];

sortedArray[j] = sortedArray[i];

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedArray[i] = k;

sortedEl[j].setValue(sortedArray[j]);

sortedEl[j].setString(Integer.toString(sortedArray[j]));

sortedEl[i].setValue(sortedArray[i]);

sortedEl[i].setString(Integer.toString(sortedArray[i]));

ourArray[j].setText(Integer.toString(sortedArray[j]));

ourArray[i].setText(Integer.toString(sortedArray[i]));

sortedEl[j].setForeground(Color.blue);

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedEl[i].setForeground(Color.blue);

} else {

j = i;

++flag;

}

}

}

for (int i = 0; i < maxElem; ++i) {

sortedEl[i].setForeground(Color.RED);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

for (int i = 0; i < maxElem; ++i) {

sortedEl[i].setForeground(Color.cyan);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

for (int i = 0; i < maxElem; ++i) {

sortedEl[i].setForeground(Color.lightGray);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}

}

unlock();

uppSecVisPanel.setData(-5, -5);

uppSecVisPanel.repaint();

}

});

myThready.start();

}

public static void moveNextStep(Vector<Integer> result, int[] arr) {

int k = 0;

int flag = 1;

Integer j = 0;

for (int i = 0; i < numberOfSortedEl; ++i) {

sortedArray[i] = arr[i];

sortedEl[i].setValue(arr[i]);

sortedEl[i].setString(Integer.toString(arr[i]));

ourArray[i].setText(Integer.toString(arr[i]));

sortedEl[i].setForeground(Color.BLUE);

}

for (Integer i : result) {

if (flag % 2 == 0) {

uppSecVisPanel.setData(i \* lengthElem, j \* lengthElem);

uppSecVisPanel.repaint();

++flag;

sortedEl[j].setForeground(Color.green);

sortedEl[i].setForeground(Color.green);

k = sortedArray[j];

sortedArray[j] = sortedArray[i];

sortedArray[i] = k;

sortedEl[j].setValue(sortedArray[j]);

sortedEl[j].setString(Integer.toString(sortedArray[j]));

sortedEl[i].setValue(sortedArray[i]);

sortedEl[i].setString(Integer.toString(sortedArray[i]));

ourArray[j].setText(Integer.toString(sortedArray[j]));

ourArray[i].setText(Integer.toString(sortedArray[i]));

for (int p = 0; p < numberOfSortedEl; ++p) {

System.out.println(sortedArray[p]);

}

sortedEl[j].setForeground(Color.blue);

sortedEl[i].setForeground(Color.blue);

} else {

j = i;

++flag;

}

}

sortedEl[j].setForeground(Color.green);

sortedEl[result.get(result.size() - 1)].setForeground(Color.green);

}

public static void moveNotChoiceSortStepByStep(Vector<Integer> result,int[] arr) {

int k = 0;

int flag = 1;

Integer j = 0;

for (int i = 0; i < numberOfSortedEl; ++i) {

sortedArray[i] = arr[i];

sortedEl[i].setValue(arr[i]);

sortedEl[i].setString(Integer.toString(arr[i]));

ourArray[i].setText(Integer.toString(arr[i]));

sortedEl[i].setForeground(Color.BLUE);

}

for (Integer i : result) {

if (flag % 2 == 0) {

++flag;

sortedEl[i].setForeground(Color.green);

sortedArray[i] = j;

sortedEl[i].setValue(j);

sortedEl[i].setString(Integer.toString(j));

ourArray[i].setText(Integer.toString(j));

sortedEl[i].setForeground(Color.blue);

} else {

j = i;

++flag;

}

}

sortedEl[result.get(result.size() - 1)].setForeground(Color.green);

sortedEl[result.get(result.size() - 2)].setForeground(Color.green);

}

private static void lock(){

anotherSortButt.setEnabled(false);

getButt.setEnabled(false);

randomButt.setEnabled(false);

sortBox.setEnabled(false);

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

ourArray[i].setEnabled(false);

}

private static void unlock(){

anotherSortButt.setEnabled(true);

getButt.setEnabled(true);

randomButt.setEnabled(true);

sortBox.setEnabled(true);

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

ourArray[i].setEnabled(true);

}

public static void moveNotChoiceSort(Vector<Integer> result){

Thread mySecondThready = new Thread(new Runnable() {

@Override

public void run() {

synchronized (this) {

lock();

int k = 0;

int flag = 1;

Integer j = 0;

for (Integer i : result) {

if (flag%2==0){

++flag;

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedEl[i].setForeground(Color.green);

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedArray[i] = j;

sortedEl[i].setValue(j);

sortedEl[i].setString(Integer.toString(j));

ourArray[i].setText(Integer.toString(j));

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

sortedEl[i].setForeground(Color.blue);

try {

Thread.sleep(speed);

} catch (InterruptedException e) {

e.printStackTrace();

}

} else {j = i;

++flag;}}

}

for (int i = 0; i < maxElem; ++i){

sortedEl[i].setForeground(Color.RED);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}}

for (int i = 0; i < maxElem; ++i){

sortedEl[i].setForeground(Color.cyan);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}}

for (int i = 0; i < maxElem; ++i){

sortedEl[i].setForeground(Color.lightGray);

try {

Thread.sleep(speedB);

} catch (InterruptedException e) {

e.printStackTrace();

}}

unlock();

}

}); mySecondThready.start();

}

public static void automaticSort(int[] safeArr, Vector<Integer> result){

nextStepSortButt.setEnabled(false);

prevStepSortButt.setEnabled(false);

String choiceSort = (String) sortBox.getSelectedItem();

switch (choiceSort) {

case "MergeSort":

MergeSort merge = new MergeSort(numberOfSortedEl, safeArr);

result = merge.sort(-1);

moveNotChoiceSort(result);

break;

case "ShellSort":

ShellSort shell = new ShellSort(numberOfSortedEl,safeArr);

result = shell.sort(-1);

moveNotChoiceSort(result);

break;

case "SelectionSort":

SelectionSort sel = new SelectionSort(numberOfSortedEl,safeArr);

result = sel.sort(-1);

move(result);

break;

case "InsertionSort":

InsertionSort ins = new InsertionSort(numberOfSortedEl,safeArr);

result = ins.sort(-1);

move(result);

break;

case "BubbleSort":

BubbleSort bubSort = new BubbleSort(numberOfSortedEl, safeArr);

result = bubSort.sort(-1);

move(result);

break;

case "QuickSort":

QuickSort quickSortt = new QuickSort(numberOfSortedEl,safeArr);

result = quickSortt.sort(-1);

move(result);

break;

case "ShakerSort":

ShakerSort shakeSort = new ShakerSort(numberOfSortedEl,safeArr);

result = shakeSort.sort(-1);

move(result);

break;

case "StupidSort":

StupidSort stupSort = new StupidSort(numberOfSortedEl,safeArr);

result = stupSort.sort(-1);

move(result);

break;

}

}

static void stepByStepSort(int step,Vector<Integer> result,int[] safeArr){

nextStepSortButt.setEnabled(true);

prevStepSortButt.setEnabled(true);

String choiceSort = (String) sortBox.getSelectedItem();

for (int i = 0;i<maxElem;++i){

safeArr[i] = firstSafedArray[i];

}

switch (choiceSort) {

case "MergeSort":

MergeSort merge = new MergeSort(numberOfSortedEl, safeArr);

result = merge.sort(step);

moveNotChoiceSortStepByStep(result,firstSafedArray);

break;

case "ShellSort":

ShellSort shell = new ShellSort(numberOfSortedEl,safeArr);

result = shell.sort(step);

moveNotChoiceSortStepByStep(result,firstSafedArray);

break;

case "SelectionSort":

SelectionSort sel = new SelectionSort(numberOfSortedEl,safeArr);

result = sel.sort(step);

moveNextStep(result,firstSafedArray);

break;

case "InsertionSort":

InsertionSort ins = new InsertionSort(numberOfSortedEl,safeArr);

result = ins.sort(step);

moveNextStep(result,firstSafedArray);

break;

case "BubbleSort":

BubbleSort bubSort = new BubbleSort(numberOfSortedEl, safeArr);

result = bubSort.sort(step);

moveNextStep(result,firstSafedArray);

break;

case "QuickSort":

QuickSort quickSortt = new QuickSort(numberOfSortedEl,safeArr);

result = quickSortt.sort(step);

moveNextStep(result,firstSafedArray);

break;

case "ShakerSort":

ShakerSort shakeSort = new ShakerSort(numberOfSortedEl,safeArr);

result = shakeSort.sort(step);

moveNextStep(result,firstSafedArray);

break;

case "StupidSort":

StupidSort stupSort = new StupidSort(numberOfSortedEl,safeArr);

result = stupSort.sort(step);

moveNextStep(result,firstSafedArray);

break;

}

}

public static class AnotherSortButtonActionListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent actionEvent) {

Vector<Integer> result = new Vector<>();

int[] safeArr = new int[maxElem];

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

safeArr[i] = sortedArray[i];

switch ((String)typeOfSort.getSelectedItem()){

case "Automatic":

automaticSort(safeArr,result);

break;

case "Step-by-Step":

lock();

anotherSortButt.setEnabled(true);

step = 1;

for (int i = 0;i<maxElem;++i){

firstSafedArray[i] = safeArr[i];

}

stepByStepSort(step,result,safeArr);

break;

case "Result":

step = -1;

for (int i = 0;i<maxElem;++i){

firstSafedArray[i] = safeArr[i];

}

stepByStepSort(step,result,safeArr);

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

sortedEl[i].setForeground(Color.lightGray);

nextStepSortButt.setEnabled(false);

prevStepSortButt.setEnabled(false);

unlock();

break;

}

}

}

public static class GetElButtonActionListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent actionEvent) {

Integer numb = maxElem, el = 0;

for (int i = 0; i < maxElem; ++i) {

sortedEl[i].setValue(minRange);

// ourArray[i].setText("");

sortedEl[i].setVisible(true);

sortedEl[i].setString("");

}

try {

numb = Integer.valueOf(numbOfElement.getText());

} catch (NumberFormatException e) {

numbOfElement.setText("Неверный формат строки");

numb = 0;

}

if (numb > maxElem) {

numb = maxElem;

numbOfElement.setText(Integer.toString(maxElem));

}

numberOfSortedEl = numb;

for (Integer i = 0; i < numb; ++i) {

sortedArray[i] = 0;

try {

el = Integer.valueOf(ourArray[i].getText());

} catch (NumberFormatException e) {

numbOfElement.setText("Неверный формат элементов");

return;

}

if (el > maxRange) {

el = maxRange;

ourArray[i].setText(Integer.toString(maxRange));

} else if (el<minRange){

el = minRange;

ourArray[i].setText(Integer.toString(minRange));

}

sortedArray[i] = el;

}

for (int i = numb; i < maxElem; ++i)

ourArray[i].setText("");

for (Integer i = 0; i < numb; ++i) {

sortedEl[i].setString(Integer.toString(sortedArray[i]));

sortedEl[i].setValue(sortedArray[i]);

}

//расскоментировать при необходимости

//for (int i = numb;i<=maxElem;++i)

// sortedEl[i].setVisible(false);

}

}

public static class RandomButtonActionListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent actionEvent) {

Random random = new Random();

Integer numb = maxElem;

for (int i = 0;i<maxElem;++i){

ourArray[i].setText("");

sortedEl[i].setValue(minRange);

sortedEl[i].setVisible(true);

sortedEl[i].setString("");

}

try {

numb = Integer.valueOf(numbOfElement.getText());

}catch (NumberFormatException e) {

numbOfElement.setText("Неверный формат строки");

numb=0;

}

if (numb>maxElem) {

numb = maxElem;

numbOfElement.setText(Integer.toString(maxElem));

} else if (numb<0){ numbOfElement.setText(Integer.toString(0));

numb = 0;}

numberOfSortedEl = numb;

Integer sch = 0;

for (Integer i = 0;i<numb;++i){

sch = random.nextInt(maxRange);

sortedEl[i].setValue(sch);

sortedArray[i] = sch;

sortedEl[i].setString(sch.toString());

ourArray[i].setText(sch.toString());

}

//расскомментировать, при необходимости

//for (int i = numb;i<maxElem;++i)

// sortedEl[i].setVisible(false);

}

}

public static class nextStepActionListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent actionEvent) {

boolean flag = true;

for (int i = 1;i<numberOfSortedEl;++i){

if (Integer.valueOf(ourArray[i-1].getText())>Integer.valueOf(ourArray[i].getText())){

flag = false;

break;}

}

if (flag==false) {

Vector<Integer> result = new Vector<>();

int[] safeArr = new int[maxElem];

if (step == -1) step = 0;

++step;

for (int i = 0; i < maxElem; ++i) {

safeArr[i] = firstSafedArray[i];

}

stepByStepSort(step, result, safeArr);

}

}

}

public static class prevStepActionListener implements ActionListener {

@Override

public void actionPerformed(ActionEvent actionEvent) {

Vector<Integer> result = new Vector<>();

int[] safeArr = new int[maxElem];

if(step>0){

--step;

for (int i = 0;i<maxElem;++i){

safeArr[i] = firstSafedArray[i];

}

stepByStepSort(step,result,safeArr);}

else if(step==0){

stepByStepSort(0,result,safeArr);

}

}

}

}