**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

**отчет**

**По учебной практике**

**Тема: Визуализация алгоритма QuickSort на Java**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 6303 |  | Горбунова А.П |
| Студент гр. 6303 |  | Жахин А.А. |
| Студент гр. 6303 |  | Малышенко Ю.И. |
| Руководитель |  |  |

Санкт-Петербург

2018

**ЗАДАНИЕ**

**на Учебную практику**

|  |  |  |
| --- | --- | --- |
| Студентка Горбунова А.П. группы 6303 | | |
| Студент Жахин А.А. группы 6303 | | |
| Студент Малышенко Ю.И. группы 6303  Тема практики: Визуализация алгоритма на Java | | |
| Задание на практику:  Командная интерактивная разработка визуализатора алгоритма на Java с графическим интерфейсом.  Алгоритм: Быстрая сортировка. | | |
| Сроки прохождения практики: 27.06.2018 – 11.07.2018 | | |
| Дата сдачи отчета: \_\_.\_\_.2018 | | |
| Дата защиты отчета: \_\_.\_\_.2018 | | |
|  | | |
| Студентка |  | Горбунова А.П |
| Студент |  | Жахин А.А. |
| Студент |  | Малышенко Ю.И. |
| Руководитель |  |  |

**АННОТАЦИЯ**

В ходе работы над проектом учебной практики был рассмотрен алгоритм быстрой сортировки массива. Разработана программа для демонстрации работы этого алгоритма на языке программирования Java. Визуализация алгоритма должна быть доступной для понимания пользователем, 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 draft training practice, the algorithm quick sorting of the array was considered. A program has been developed to demonstrate the operation of this algorithm in the Java programming language. Visualization of the algorithm should be accessible for understanding the user, and the program interface is convenient.

Project implementation in a team that promotes the development of group work skills. The basic condition for the implementation of the program in the team is the availability of a development plan and a section of duties.

Содержание

[ВВЕДЕНИЕ 5](#_Toc518252417)

[1. ОПИСАНИЕ АЛГОРИТМА 7](#_Toc518252418)

[2. ТРЕБОВАНИЯ К ПРОГРАММЕ 8](#_Toc518252419)

[2.1. Исходные требования к программе 8](#_Toc518252420)

[2.1.1. Требования к входным данным 8](#_Toc518252421)

[2.1.2. Требования к визуализации 8](#_Toc518252422)

[2.1.3. Требования к интерфейсу программы 9](#_Toc518252423)

[2.2 Уточнение требований после сдачи прототипа 10](#_Toc518252424)

[3. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ 11](#_Toc518252425)

[3.1. План разработки 11](#_Toc518252426)

[3.2. Распределение ролей в бригаде 11](#_Toc518252427)

[4. ОСОБЕННОСТИ РЕАЛИЗАЦИИ 12](#_Toc518252428)

[4.1. Основной метод 12](#_Toc518252429)

[4.2. Exception в программе 13](#_Toc518252430)

[5. ТЕСТИРОВАНИЕ 14](#_Toc518252431)

[5.1. Тестирование интерфейса программы 14](#_Toc518252432)

[5.2. Тестирование алгоритма программы 22](#_Toc518252433)

[ЗАКЛЮЧЕНИЕ 24](#_Toc518252434)

[Список использованных источников 25](#_Toc518252435)

[Приложение А. Код программы 26](#_Toc518252436)

# ВВЕДЕНИЕ

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена (его варианты известны как «[Пузырьковая сортировка](https://ru.wikipedia.org/wiki/%D0%9F%D1%83%D0%B7%D1%8B%D1%80%D1%8C%D0%BA%D0%BE%D0%B2%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)» и «[Шейкерная сортировка](https://ru.wikipedia.org/wiki/%D0%A8%D0%B5%D0%B9%D0%BA%D0%B5%D1%80%D0%BD%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0" \o "Шейкерная сортировка)»), 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.

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.
* 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».
* 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 (см. ниже).

Хоар разработал этот метод применительно к [машинному переводу](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%88%D0%B8%D0%BD%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B5%D1%80%D0%B5%D0%B2%D0%BE%D0%B4); словарь хранился на [магнитной ленте](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D0%B3%D0%BD%D0%B8%D1%82%D0%BD%D0%B0%D1%8F_%D0%BB%D0%B5%D0%BD%D1%82%D0%B0), и сортировка слов обрабатываемого текста позволяла получить их переводы за один прогон ленты, без перемотки её назад. Алгоритм был придуман Хоаром во время его пребывания в [Советском Союзе](https://ru.wikipedia.org/wiki/%D0%A1%D0%A1%D0%A1%D0%A0), где он обучался в [Московском университете](https://ru.wikipedia.org/wiki/%D0%9C%D0%BE%D1%81%D0%BA%D0%BE%D0%B2%D1%81%D0%BA%D0%B8%D0%B9_%D0%B3%D0%BE%D1%81%D1%83%D0%B4%D0%B0%D1%80%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B8%D1%82%D0%B5%D1%82_%D0%B8%D0%BC%D0%B5%D0%BD%D0%B8_%D0%9C._%D0%92._%D0%9B%D0%BE%D0%BC%D0%BE%D0%BD%D0%BE%D1%81%D0%BE%D0%B2%D0%B0) компьютерному переводу и занимался разработкой русско-английского разговорника.

В процессе выполнения работы была разработана программа на языке Java, предназначенную для осуществления сортировки массива с помощью быстрой сортировки.

Требовалось реализовать пользовательский интерфейс и визуализацию алгоритма. Предусмотрена возможность ходить по решению вперед-назад, чтобы подробнее проследить ход решения.

# ОПИСАНИЕ АЛГОРИТМА

Алгоритм состоит из трёх шагов:

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

# ТРЕБОВАНИЯ К ПРОГРАММЕ

## Исходные требования к программе

### Требования к входным данным

Массив должен состоять только из целых чисел (от -99999 до 99999) количеством не более 100, элементы вводятся через пробел в специальное верхнее поле в окне программы (см. начальный прототип), либо через текстовый файл. Пользователь выбирает, каким именно способом он будет вводить массив.

Ввод массива можно осуществить двумя способами:

1. Массив задается из файла .txt (рисунок 1) (для ввода с файла необходимо нажать )



Рисунок 1 – массив в файле

2. С помощью активного поля



Для запуска решения необходимо нажать “GO”.

### Требования к визуализации

Массив визуализируется в окне в виде чисел в шариках (рисунок 2):



Рисунок 2 – Пример визуализации массива

При демонстрации поиска решения рассматривается каждый шаг, при этом элементы, которые поменяются местами выделены синим цветом. Опорный элемент выделен красным.

После того, как вершина рассмотрена, она окрашивается в красный цвет.

При демонстрации решения между 2 состояниями стрелками показаны в какие места перейдут обмениваемые элементы.

### Требования к интерфейсу программы

Интерфейс программы представлен на рисунке 3:

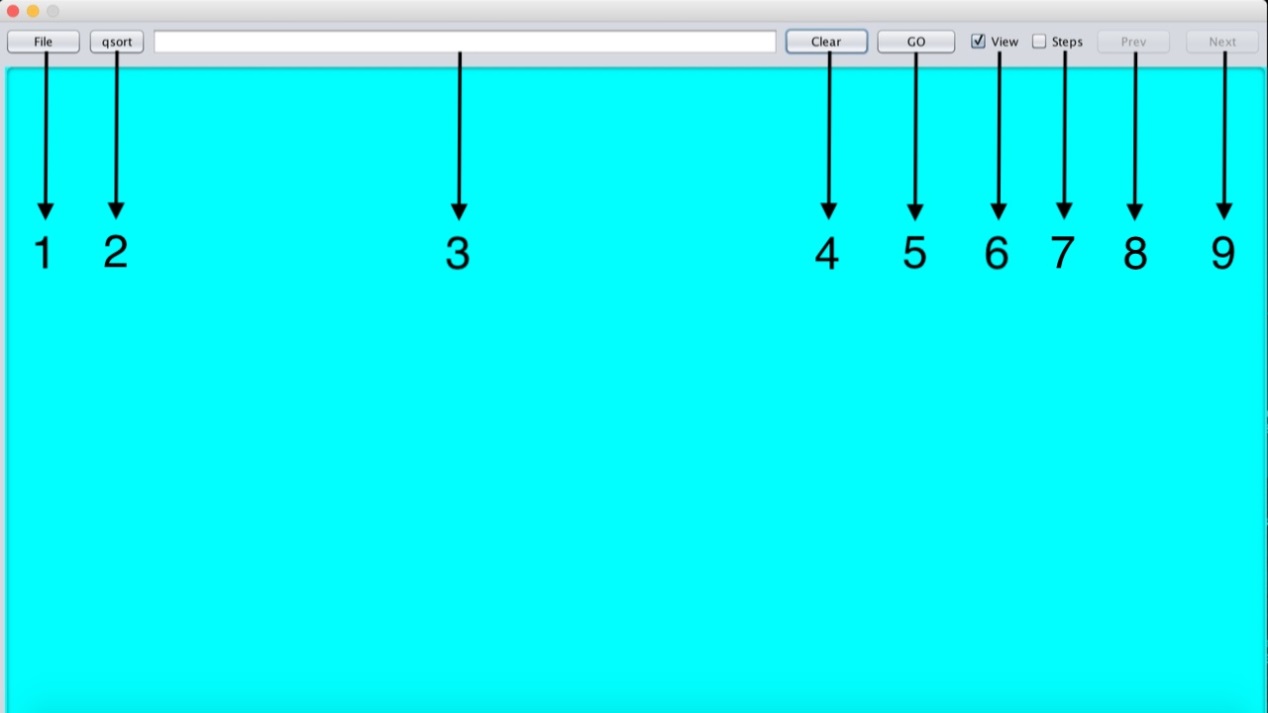


Рисунок 3 - Интерфейс программы

Описание интерфейса программы:

1. Кнопка для ввода данных из файла
2. Кнопка для вывода описания алгоритма
3. Поле для ввода входных данных
4. Кнопка для очищения данных и подготовки к сортировки нового массива
5. Кнопка для запуска программы
6. Метка для включения/выключения визуализации алгоритма
7. Метка для перехода к пошаговой демонстрации алгоритма
8. Кнопка для перехода к предыдущему шагу при пошаговой демонстрации алгоритма
9. Кнопка для перехода к следующему шагу при пошаговой демонстрации алгоритма

## Уточнение требований после сдачи прототипа

После сдачи прототипа было внесено несколько изменений в требования к программе.

Проблема: нет автоматической пошаговой визуализации алгоритма и нет кнопки с описанием алгоритма.

Решение: на рисунке 4 изображен новый интерфейс программы, который удовлетворяет новым требованиям.

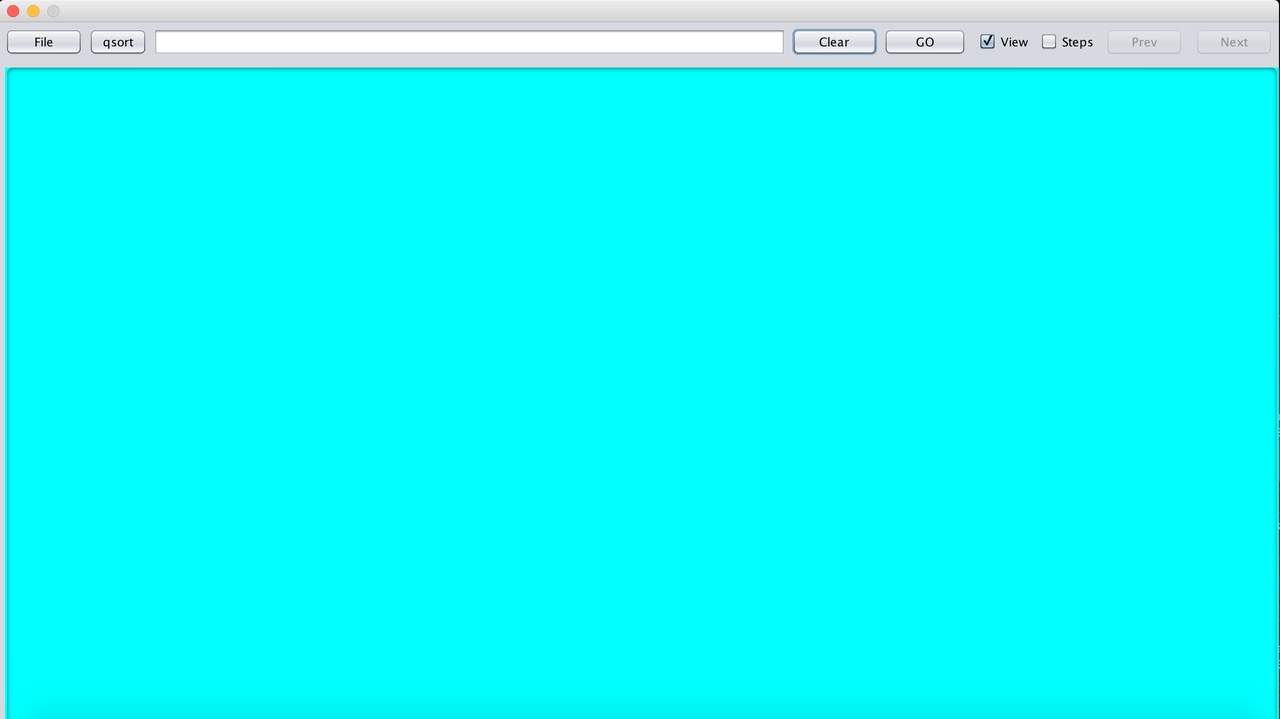


Рисунок 4 – Измененный интерфейс программы

Как можно увидеть, добавлена новая кнопка с описанием алгоритма и кнопка для очищения данных и подготовки к сортировке нового массива:

1. Добавили возможность очищать данные в строке.
2. Появилась возможность заново вводить массив, предварительно удалив предыдущий массив.
3. Появилась кнопка «qsort» для ознакомления с алгоритмом перед началом работы с программой.
4. Появился возможность пошагового вывода работы алгоритма автоматически.

# ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

## План разработки

29.06: Программа должна иметь рабочий алгоритм, а также улучшенный прототип интерфейса (с вводом/выводом элементов массива с промежуточными данными, выводом промежуточных подмассивов, выделением опорных элементов для них цветом, соблюдением отступов при рекурсии).

01.07: Доработка алгоритма визуализации, обработка различных исключительных ситуаций (например, ввода в качестве элементов массива не целых чисел, выбор пустого текстового файла), предоставление конечного варианта проекта.

## Распределение ролей в бригаде

Распределение ролей в бригаде представлено в Таблице 1.

Таблица 1 – Распределение ролей в бригаде

|  |  |
| --- | --- |
| ФИО | Обязанности |
| Малышенко Ю.И. | * Реализация алгоритма * Оформление спецификации * Тестирование программы |
| Жахин А.А. | * Визуализация алгоритма * Разработка архитектуры проекта * Оформление пояснительной записки |
| Горбунова А.П. | * Реализация окончательного варианта визуализации * Разработка тестов программы * Оформление пояснительной записки |

# ОСОБЕННОСТИ РЕАЛИЗАЦИИ

## Основной метод

Быстрая сортировка относится к алгоритмам «разделяй и властвуй».

Алгоритм состоит из трёх шагов:

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

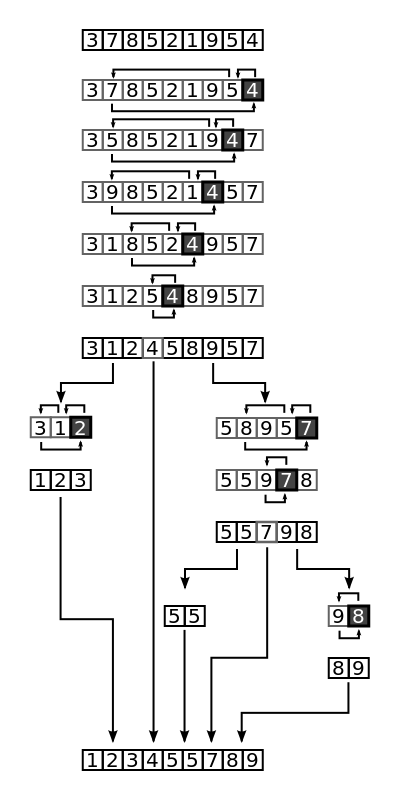


Рисунок 5 – Пример быстрой сортировки.

Здесь опорным является последний элемент массива (ячейка чёрного цвета), что в отсортированных массивах может приводить к ухудшению производительности.

## Exception в программе

Исключение — это событие при выполнении программы, которое приводит к её ненормальному или неправильному поведению.

В нашей программе такие события связаны с вводом данных пользователем. При вводе некорректных данных будет выводиться сообщение об ошибке. В таблице 2 представлены такие события и выводимые сообщения.

Таблица 2 – Вывод сообщения при вводе некорректных данных

|  |  |
| --- | --- |
| Ввод некорректного данного | Сообщение |
| В поле для ввода массива не введен ни один символ. | Пустой массив! |
| В поле для ввода массива введены дробные числа.  В поле для ввода массива введены недопустимые символы. | Введите только целые числа! |

# ТЕСТИРОВАНИЕ

## Тестирование интерфейса программы

При тестировании интерфейса программы будет проверяться корректная работа программы при работе с пользователем. Также в тестировании программы можно увидеть описание доработанного интерфейса программа.

При запуске программы открывается окно, представленное на рисунке 6.

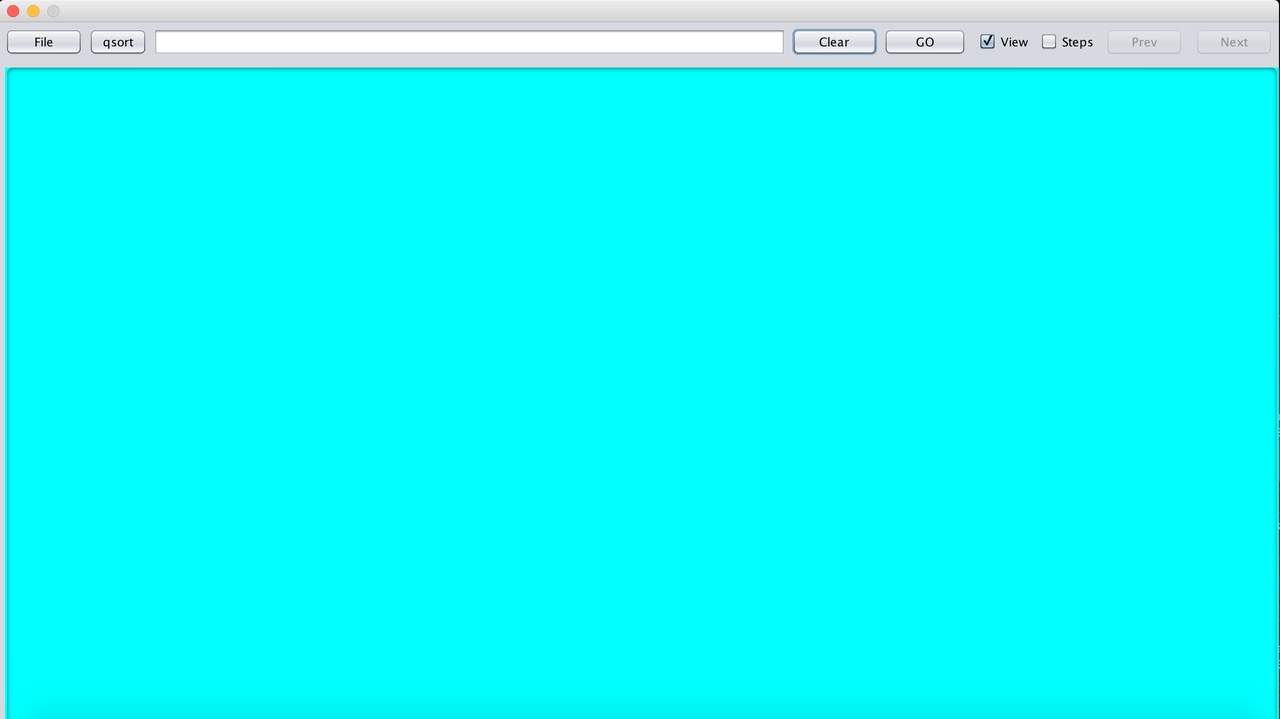


Рисунок 6

Как видно на рисунке 6, несколько кнопок неактивны, потому что в данный момент Пользователь не может начать поиск решения, так как не введен массив.

Перед началом работы с программой нажмем на кнопку «qsort» и прочитаем описание алгоритма (рисунок 7).

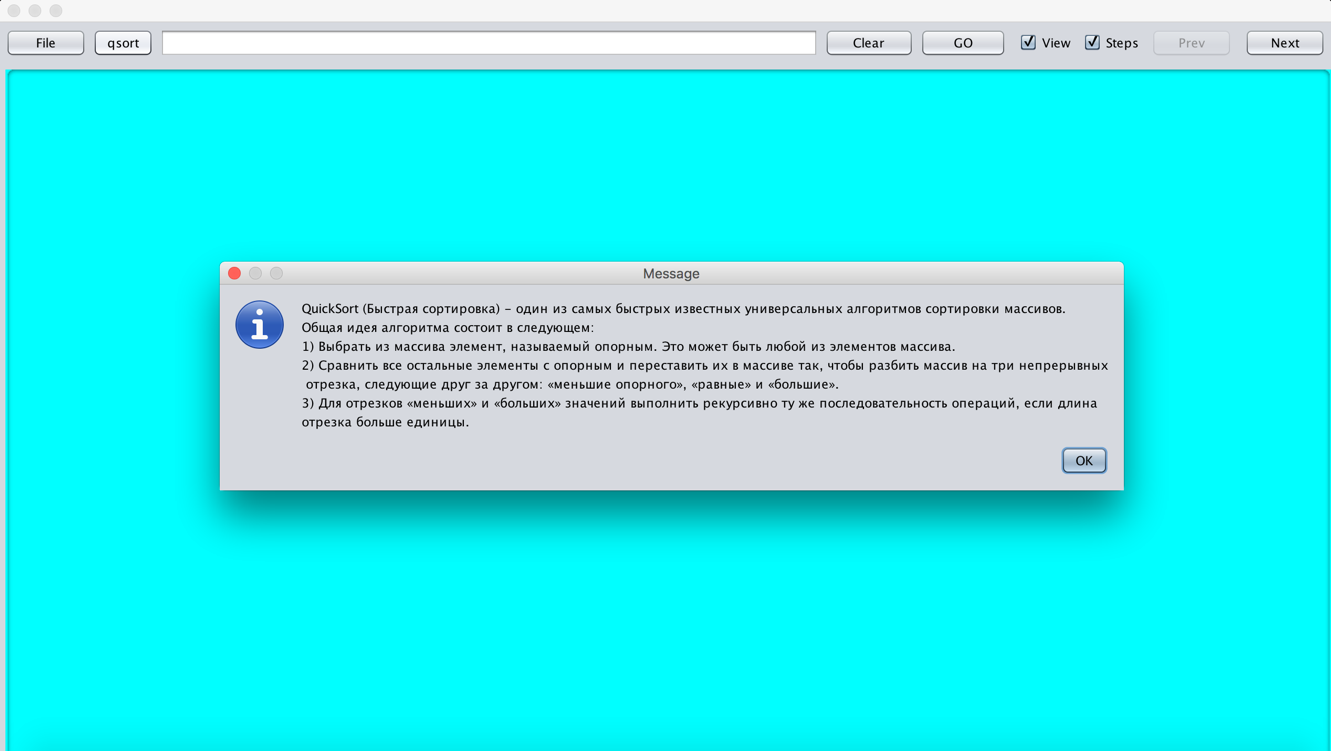


Рисунок 7 – Описание алгоритма.

Оставим строку пустой и запустим алгоритм с помощью кнопки «GO!». Появляется диалоговое окно с ошибкой. Результат представлен на рисунке 8.

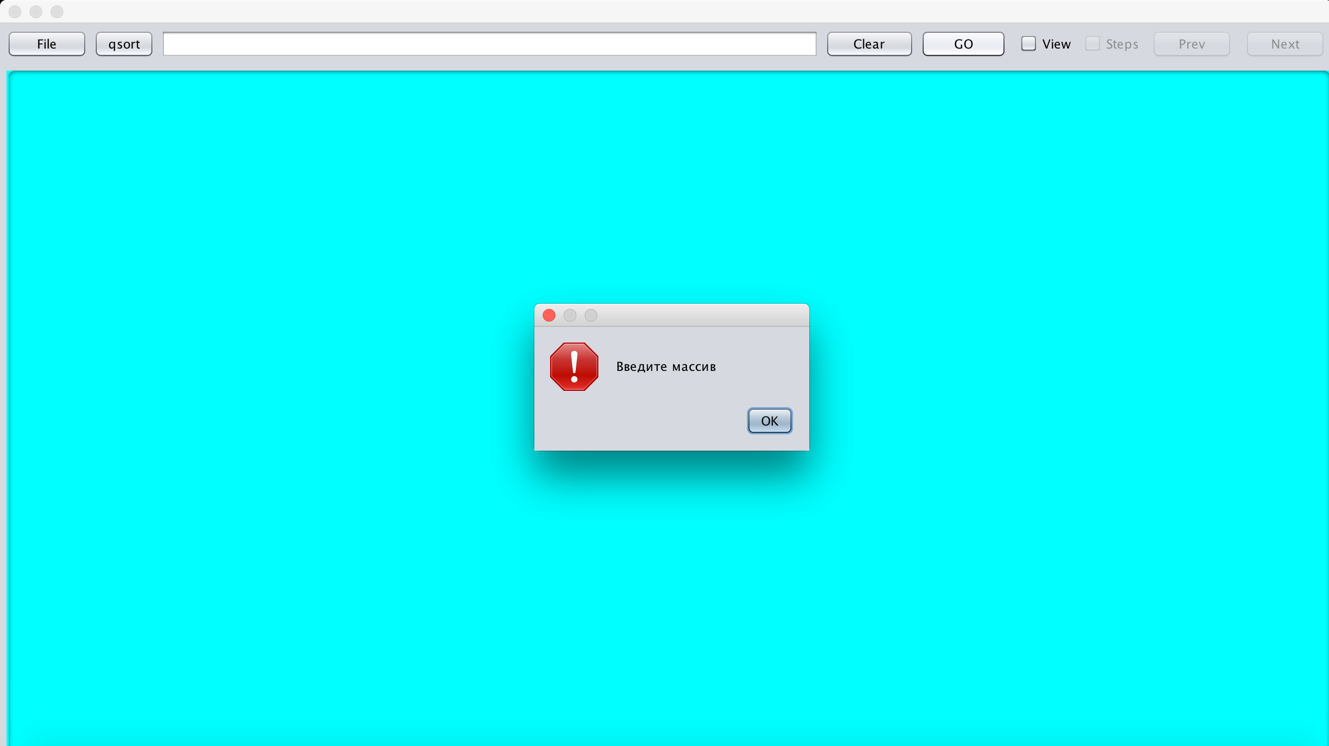


Рисунок 8 – Диалоговое окно с ошибкой.

Снимем выделение «View» и добавим в строку массив, который нужно отсортировать. При нажатии кнопки «GO» массив отсортируется в этой же строке. Результат представлен на рисунке 9.1 и 9.2.

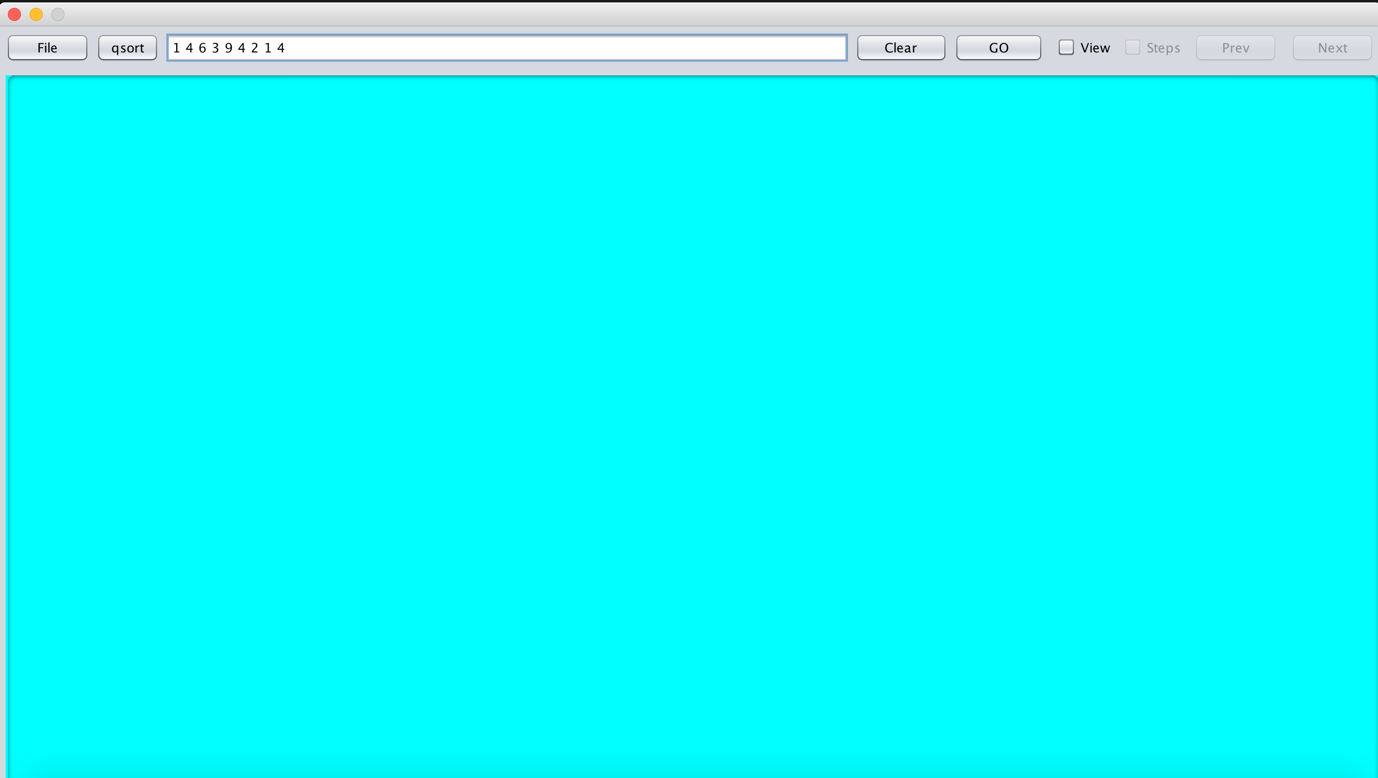


Рисунок 9.1 – Массив до сортировки.

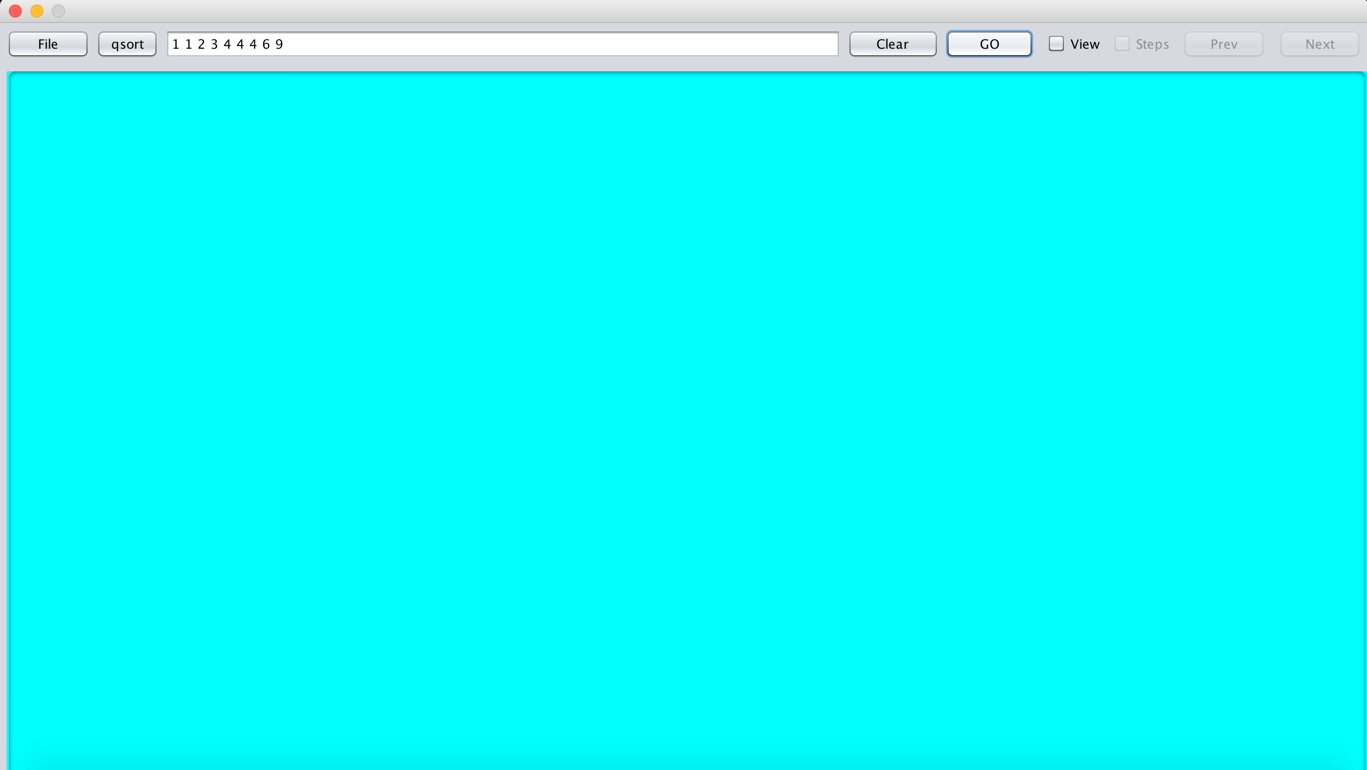


Рисунок 9.1 – Массив после сортировки.

Выделим «View», добавим в строку массив, который нужно отсортировать. При нажатии кнопки «GO» массив отсортируется в этой же строке и выведется в окне визуализации. Результат представлен на рисунке 10.

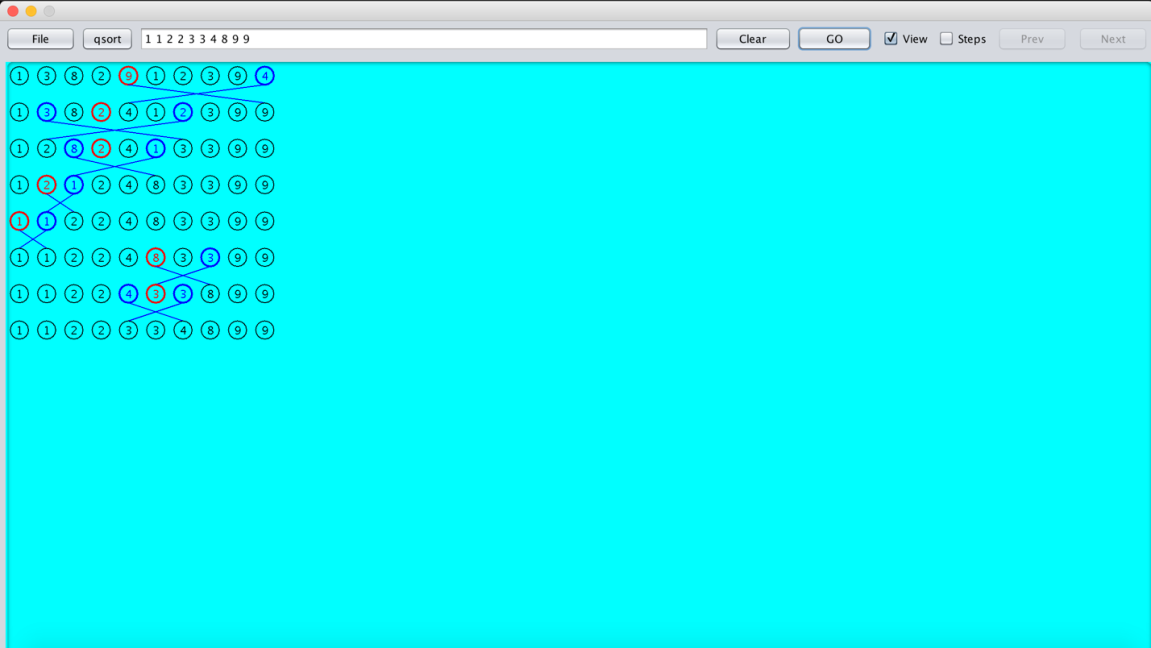


Рисунок 10 – Пошаговая работа алгоритма выведена в окне визуализации.

Выделим «View» и  «Steps», добавим в строку массив, который нужно отсортировать. При нажатии кнопки «GO» массив отсортируется в этой же строке и выведется в окне визуализации пошагово автоматически. Результат представлен на рисунке 11.1 и 11.2.

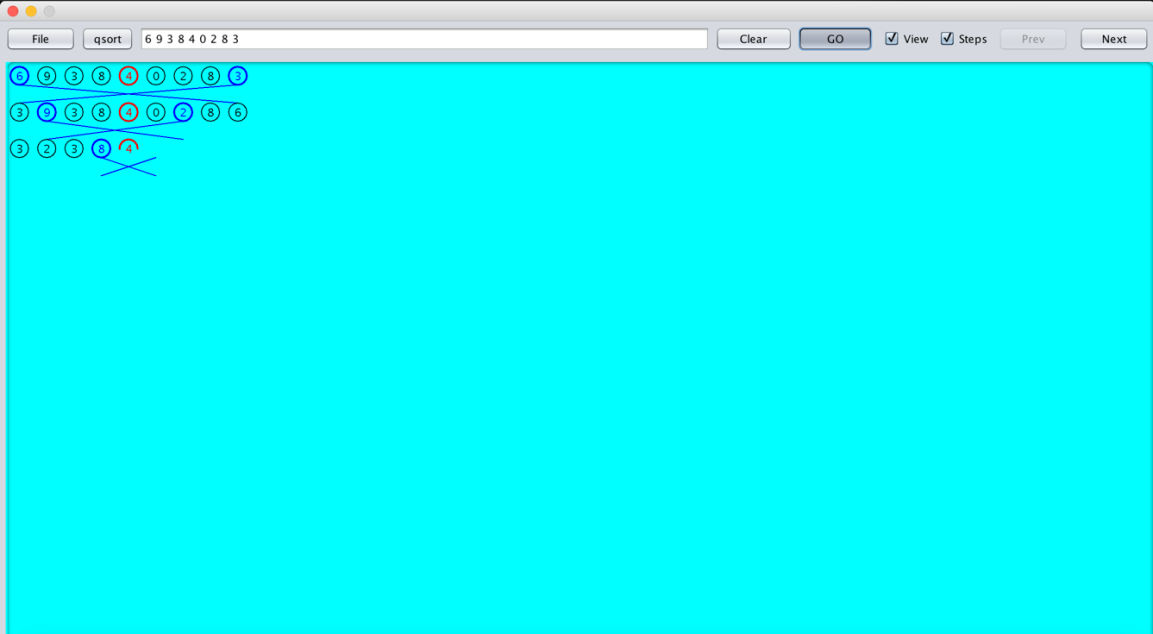


Рисунок 11.1 – Пошаговая работа алгоритма выводится в окне визуализации автоматически.

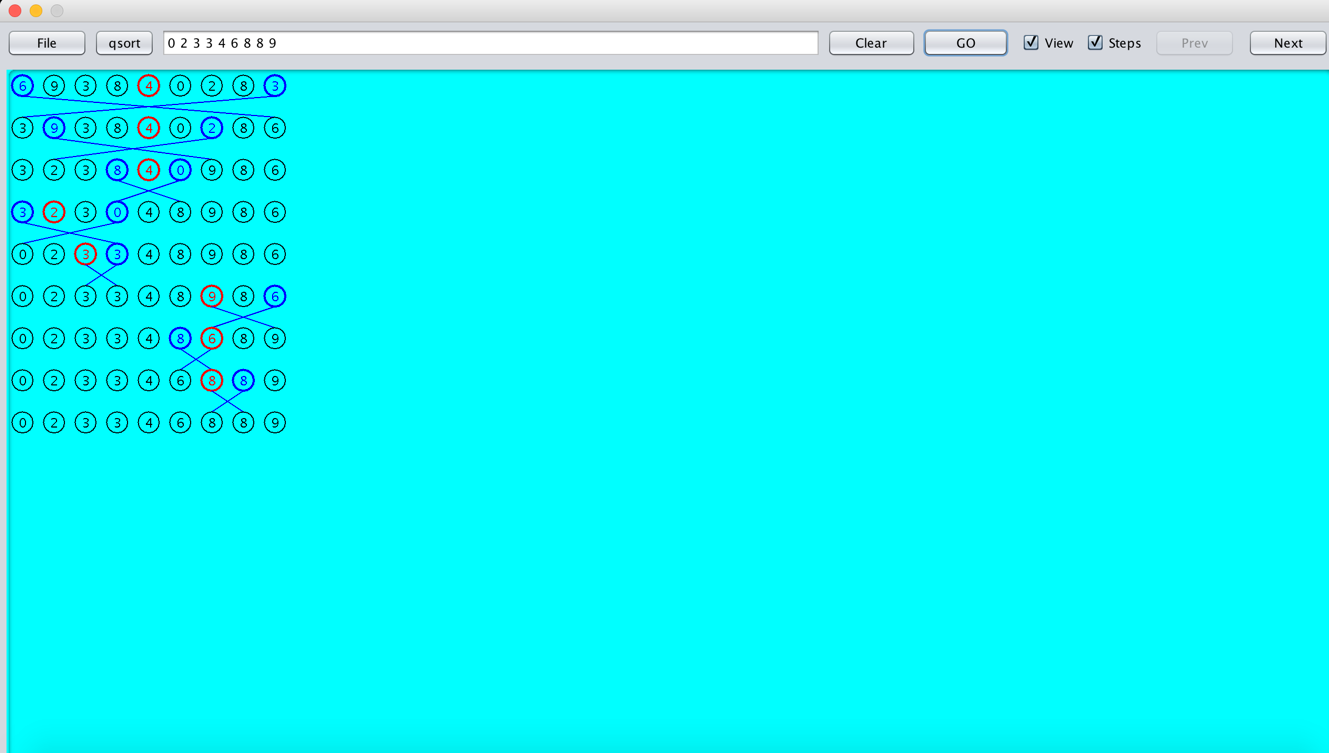


Рисунок 11.2 – Автоматическая пошаговая работа алгоритма завершила свою работу.

При активных кнопках «View» и «Steps» активизируются кнопки «Prev» и «Next», которые позволяют управлять пошаговым выводом. Добавим в строку массив, который нужно отсортировать. При нажатии кнопки «Next» массив отсортируется в этой же строке и выведется в окне визуализации пошагово. При нажатии кнопки «Prev» происходит возврат на предыдущий шаг.

Результат представлен на рисунке 12.1, 12.2 и 12.3.

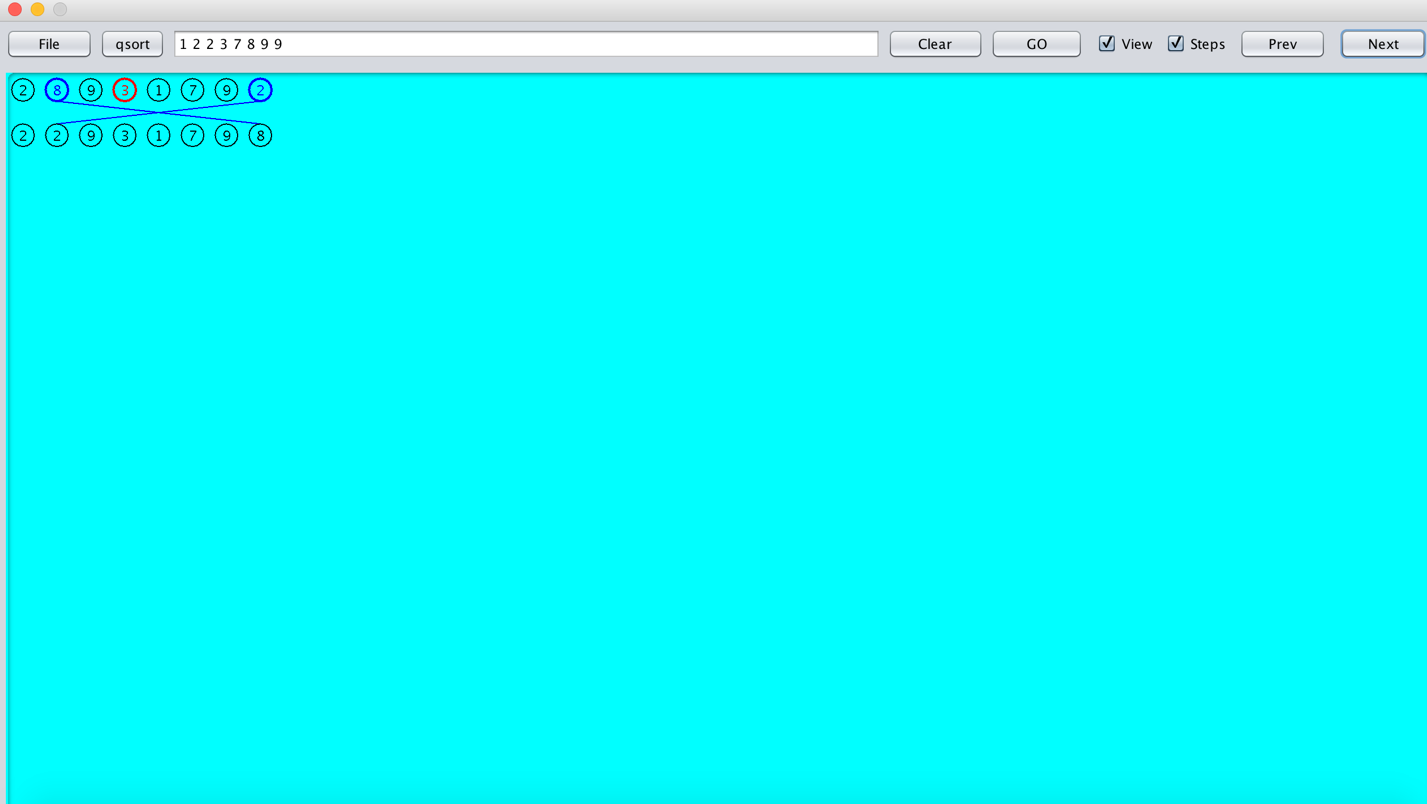


Рисунок 12.1 – Начало пошаговой работы алгоритма.

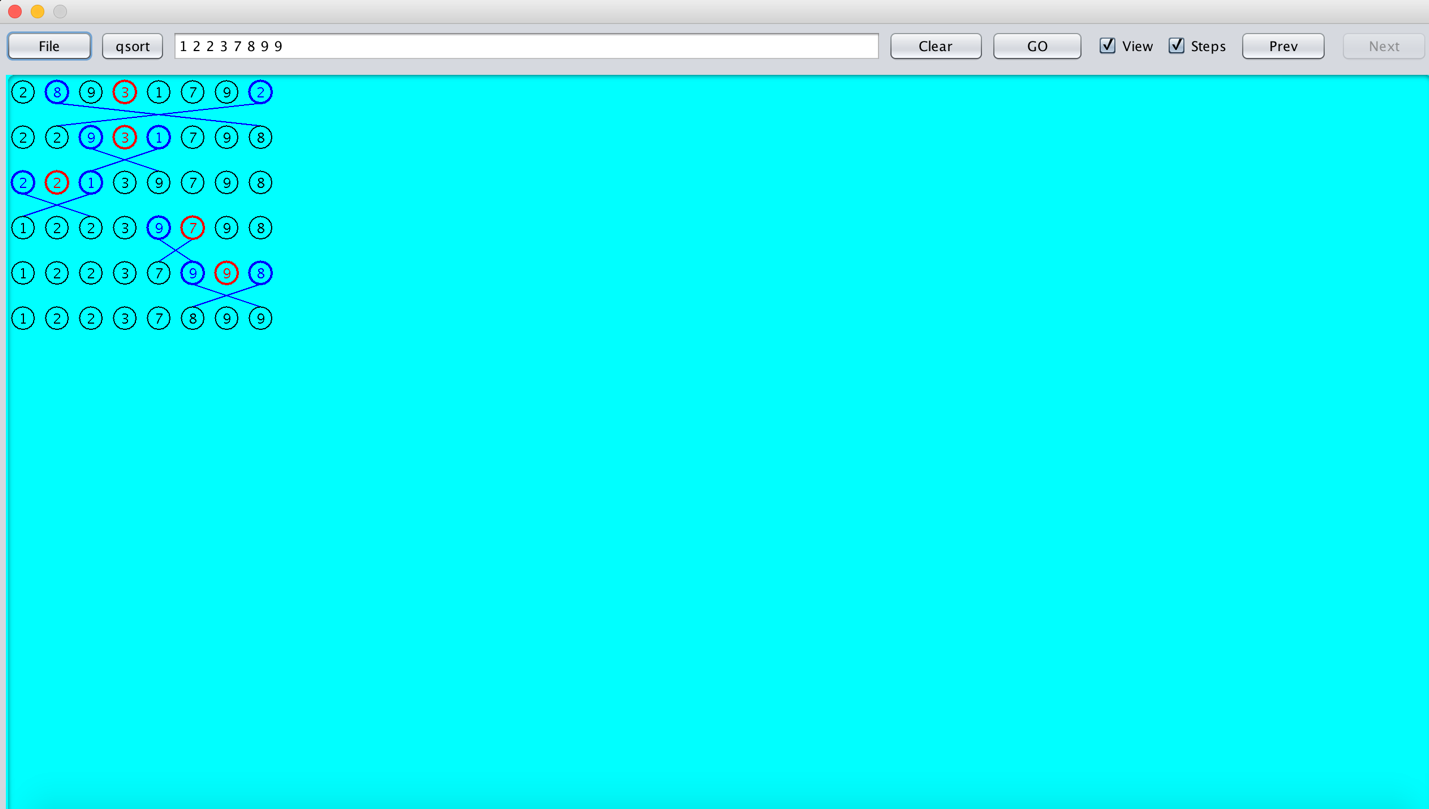


Рисунок 12.2 – Конец пошаговой работы алгоритма.

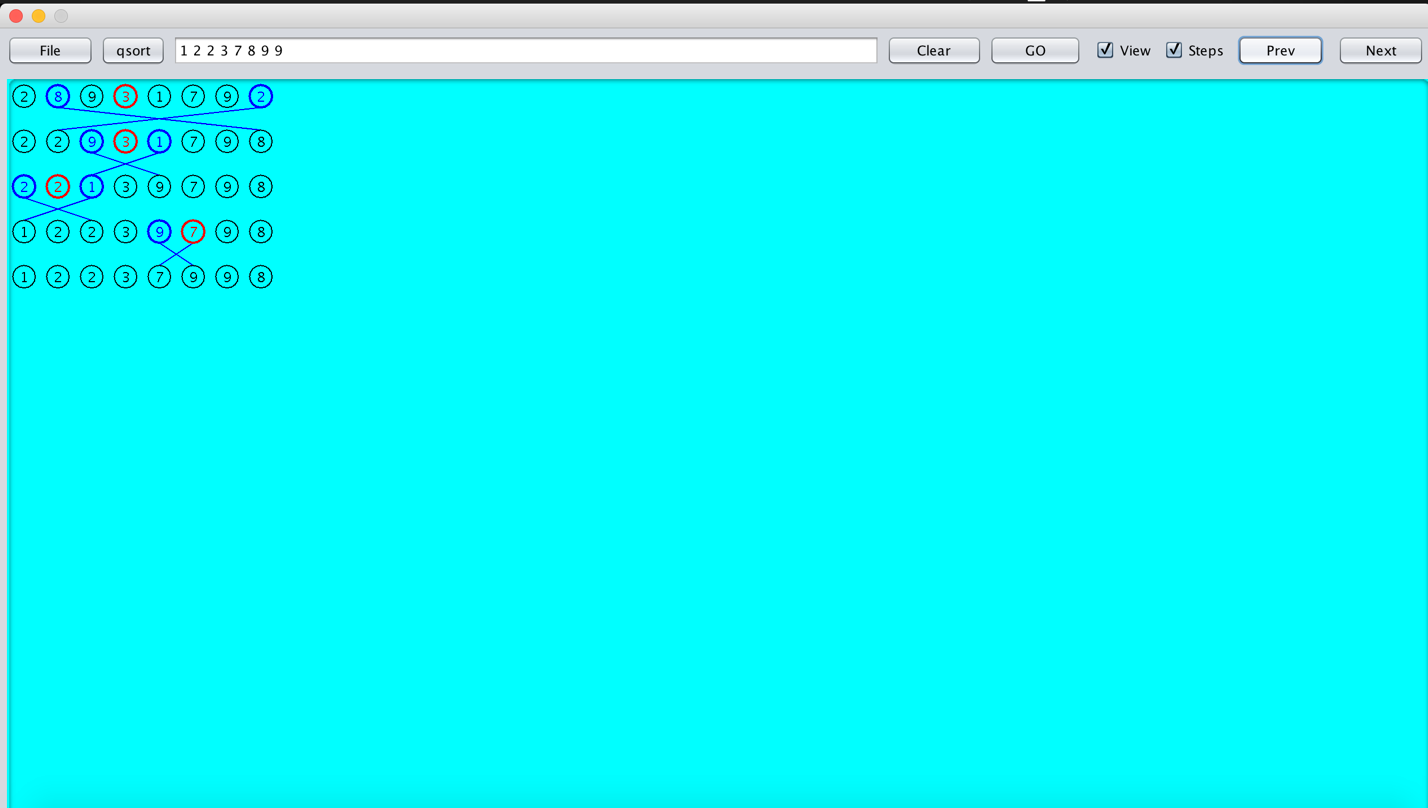


Рисунок 12.3 – Возврат на предыдущий шаг.

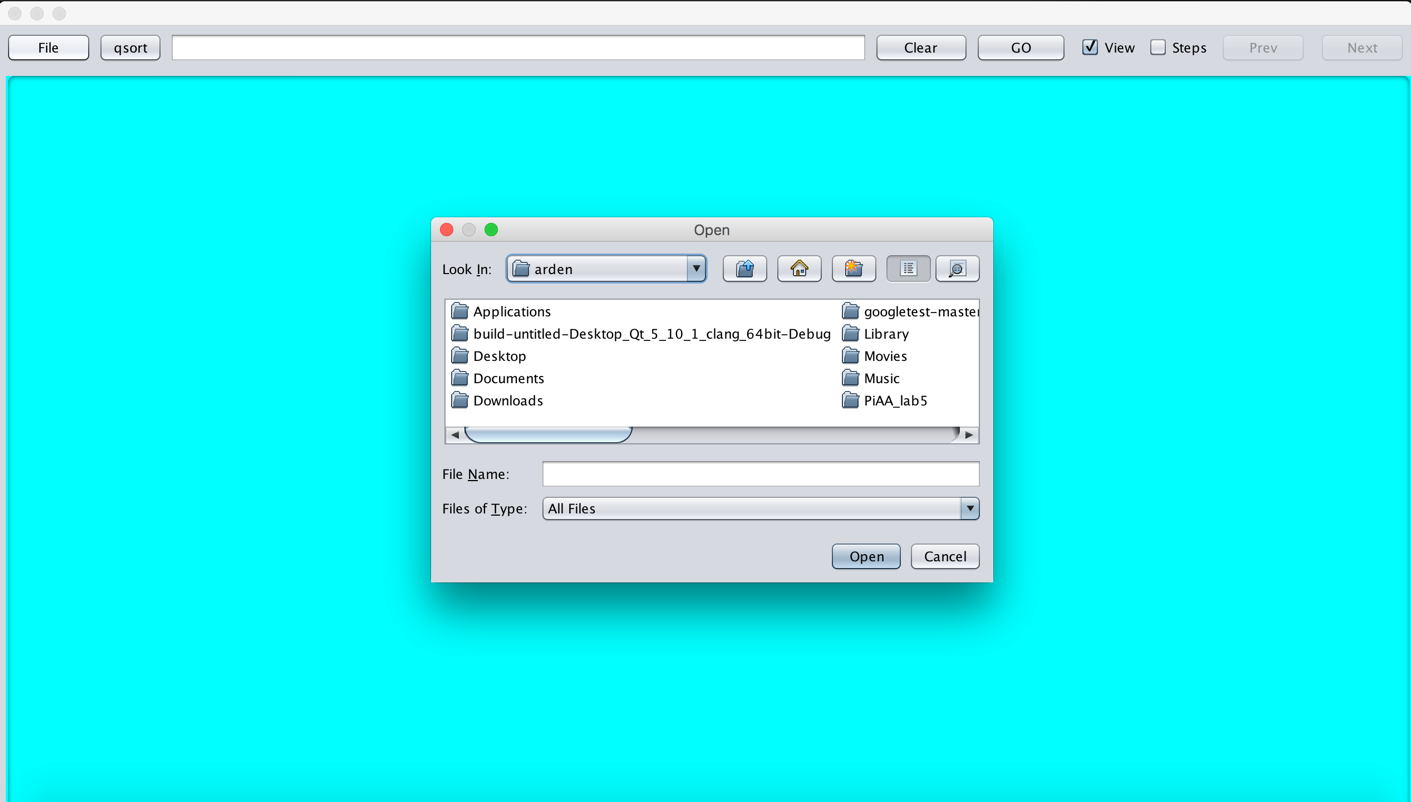
Проверим сортировку массива взятого из файла. Нажмем на кнопку «File», выберем заведомо созданный массив в файле и отсортируем его. Результат представлен на рисунке 13.1, 13.2 и 13.3.

Рисунок 12.1 – Выбираем файл с массивом.

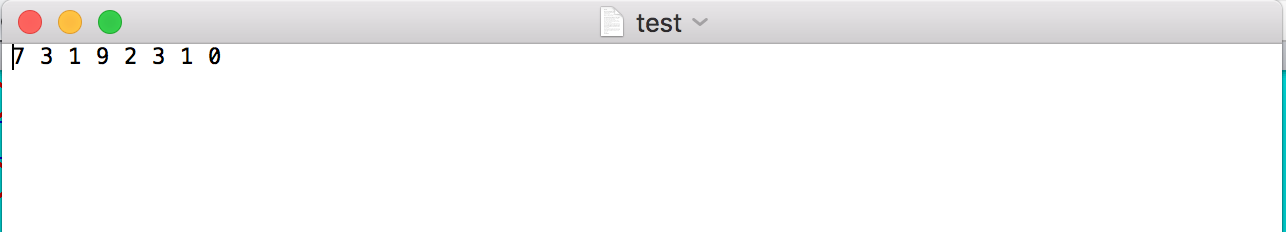


Рисунок 12.2 – Массив в файле.

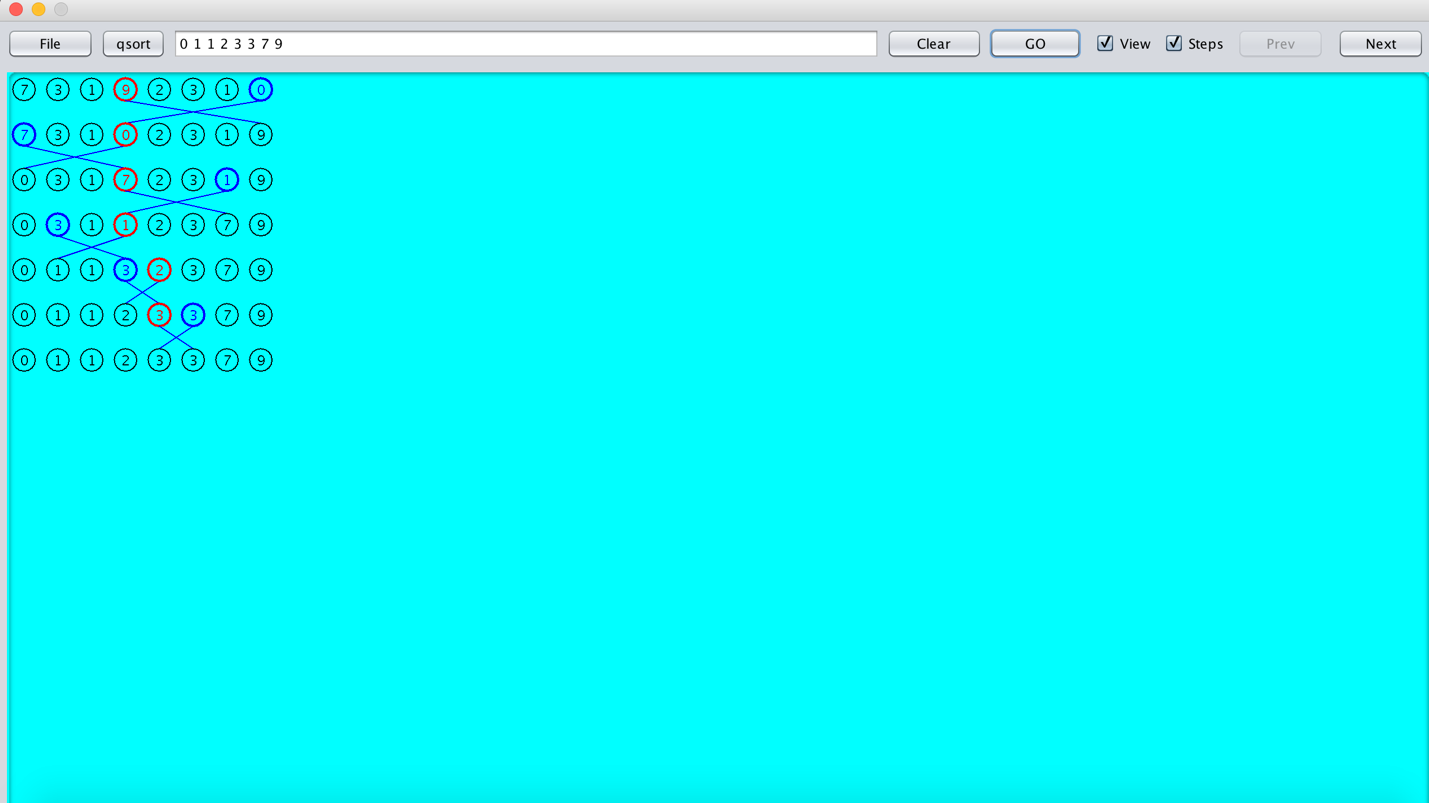
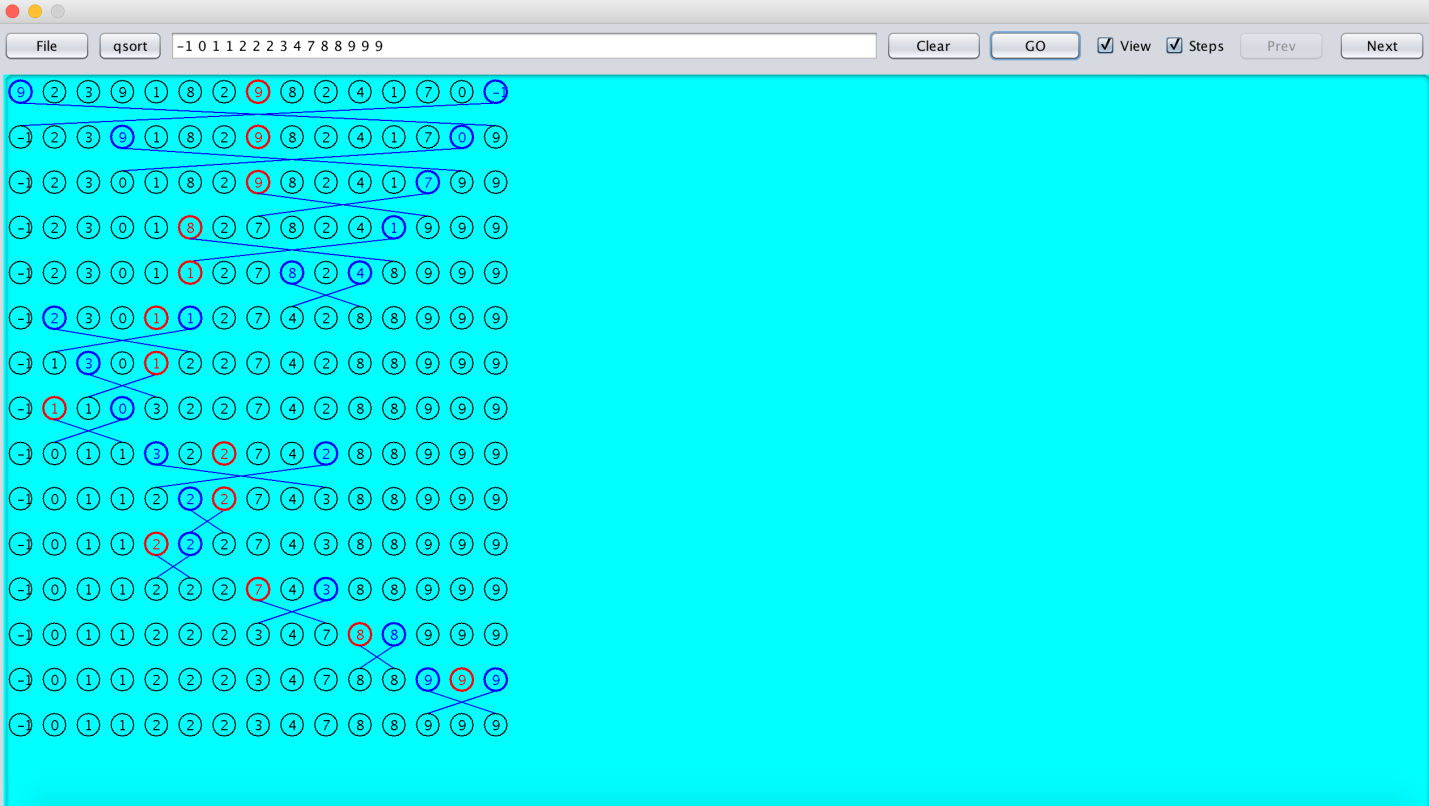


Рисунок 12.3 – Отсортированный массив из файла.

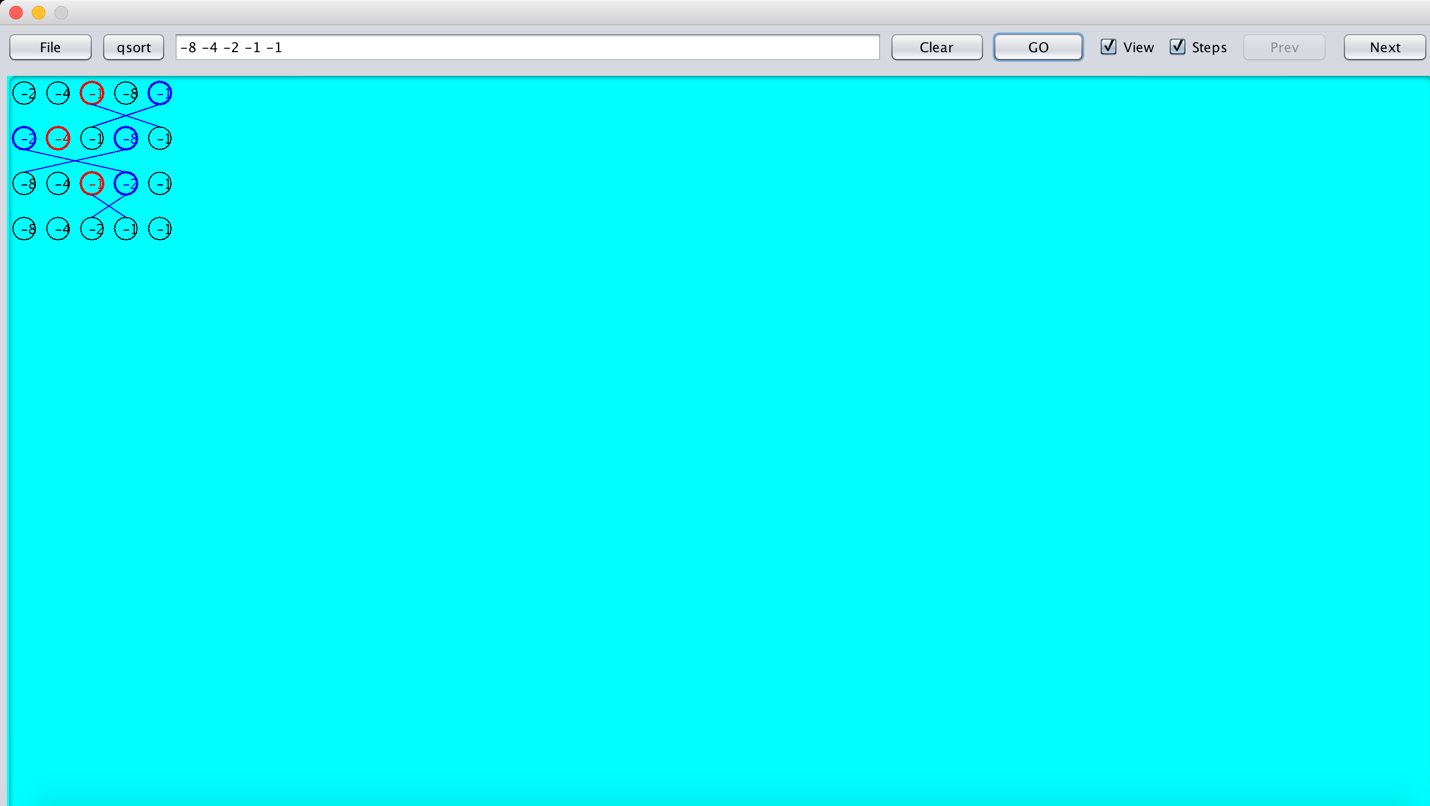
## Тестирование алгоритма программы

Проведем несколько тестов программы, причем массивы будут с некоторыми особенностями.

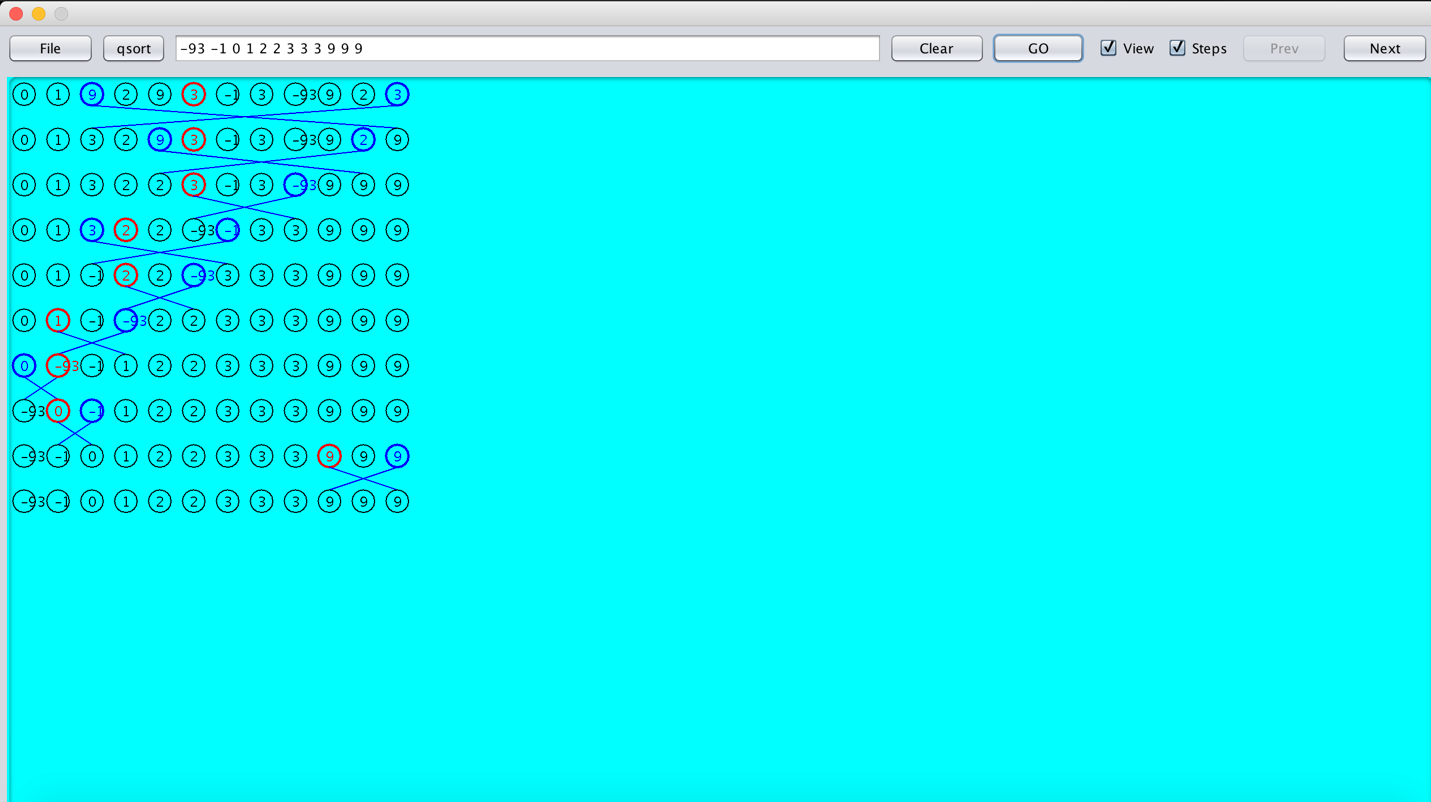
**Тест 1.**

****

**Тест 2.**

****

**Тест 3.**

****

# ЗАКЛЮЧЕНИЕ

В результате работы над данным проектом была написана программа, которая успешно справляется со своей основной задачей: помочь понять пользователю принцип работы алгоритма быстрой сортировки.

Было проведено тестирование на работоспособность полученной программы. В ходе тестирования было выявлено, что полученная программа работает корректно и предупреждает ввод некорректных данных со стороны пользователя.

При реализации этой программы, в течение отведенного времени, было разработано несколько версий графического интерфейса, поскольку первоначально поставленные требования оказались избыточны и были изменены (при этом программа сохранила свой функционал).

На данном этапе программа, т.е. реализация алгоритма и интерфейс, оптимальна и корректно работающая.

Работа по созданию программы позволила приобрести практические навыки не только по использованию методов программирования при решении задачи, но и навыки совместной работы и разделения обязанностей.

# Список использованных источников

1. «Алгоритмы. Построение и анализ» / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн, Издательство: Вильямс, 2013 год, 1324 стр.
2. Документация // URL: <http://docs.oracle.com/javase/8/javase-clienttechnologies.htm> (даты обращения: 27.06.2017-29.06.2017)
3. Паттерны для новичков: MVC vs MVP vs MVVM // Хабрахабр. URL: <https://habrahabr.ru/post/215605/> (дата обращения: 28.06.2017)
4. Эккель Брюс Философия Java [4-е издание] Издательство: Питер, 2015 год, 1170 стр.

# Приложение А. Код программы

|  |
| --- |
| import java.awt.BasicStroke;  import java.awt.Color;  import javax.swing.JOptionPane;  import java.util.ArrayList;  import java.util.List;  import java.awt.Graphics2D;  import java.awt.event.ActionEvent;  import java.awt.event.WindowEvent;  import java.awt.geom.Ellipse2D;  import java.io.File;  import java.io.IOException;  import java.util.Scanner;  import javax.swing.JFileChooser;  import javax.swing.Timer;  import java.util.\*;  import java.lang.Thread;  import java.util.logging.Level;  import java.util.logging.Logger;  public class Form extends javax.swing.JFrame {  public Form() {  initComponents();  if (g2 == null)  g2 = (Graphics2D) jPanel1.getGraphics();  jPanel1.update(g2);  }    private Ellipse2D circbuffer;  private Graphics2D g2;    //параметры шаров  final int radius = 20;  final int step = 25;  int numArr[], curr\_pos = 0;  boolean draw = true;    //Список состояний массива  private List<int[]> mainList = new ArrayList<>();  //Обводка  private final List<int[]> linesList = new ArrayList<>();    //@SuppressWarnings("unchecked");  // <editor-fold defaultstate="collapsed" desc="Generated Code">  private void initComponents() {  buttonGroup1 = new javax.swing.ButtonGroup();  jTextField1 = new javax.swing.JTextField();  jButton1 = new javax.swing.JButton();  jPanel1 = new javax.swing.JPanel();  jButton2 = new javax.swing.JButton();  jCheckBox1 = new javax.swing.JCheckBox();  jButton3 = new javax.swing.JButton();  jButton4 = new javax.swing.JButton();  jCheckBox2 = new javax.swing.JCheckBox();  jButton5 = new javax.swing.JButton();  jButton6 = new javax.swing.JButton();  setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  setResizable(false);  addWindowListener(new java.awt.event.WindowAdapter() {  public void windowOpened(java.awt.event.WindowEvent evt) {  formWindowOpened(evt);  }  });  jTextField1.addMouseListener(new java.awt.event.MouseAdapter() {  public void mouseClicked(java.awt.event.MouseEvent evt) {  jTextField1MouseClicked(evt);  }  });  jButton1.setText("GO");  jButton1.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton1ActionPerformed(evt);  }  });  jPanel1.setBackground(new java.awt.Color(0, 255, 255));  jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(""));  javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  jPanel1.setLayout(jPanel1Layout);  jPanel1Layout.setHorizontalGroup(  jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGap(0, 0, Short.MAX\_VALUE)  );  jPanel1Layout.setVerticalGroup(  jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGap(0, 713, Short.MAX\_VALUE)  );  jButton2.setText("File");  jButton2.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton2ActionPerformed(evt);  }  });  jCheckBox1.setText("Steps");  jCheckBox1.addChangeListener(new javax.swing.event.ChangeListener() {  public void stateChanged(javax.swing.event.ChangeEvent evt) {  jCheckBox1StateChanged(evt);  }  });  jButton3.setText("Prev");  jButton3.setEnabled(false);  jButton3.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton3ActionPerformed(evt);  }  });  jButton4.setText("Next");  jButton4.setEnabled(false);  jButton4.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton4ActionPerformed(evt);  }  });  jCheckBox2.setSelected(true);  jCheckBox2.setText("View");  jCheckBox2.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jCheckBox2ActionPerformed(evt);  }  });  jButton5.setText("qsort");  jButton5.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton5ActionPerformed(evt);  }  });  jButton6.setText("Clear");  jButton6.addActionListener(new java.awt.event.ActionListener() {  public void actionPerformed(java.awt.event.ActionEvent evt) {  jButton6ActionPerformed(evt);  }  });  javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  getContentPane().setLayout(layout);  layout.setHorizontalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(layout.createSequentialGroup()  .addContainerGap()  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)  .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addGroup(layout.createSequentialGroup()  .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED\_SIZE, 77, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(jButton5)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, 627, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(jButton6, javax.swing.GroupLayout.PREFERRED\_SIZE, 85, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED\_SIZE, 82, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(jCheckBox2)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(jCheckBox1)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(jButton3, javax.swing.GroupLayout.PREFERRED\_SIZE, 77, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(jButton4, javax.swing.GroupLayout.PREFERRED\_SIZE, 77, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addContainerGap())))  );  layout.setVerticalGroup(  layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addGroup(layout.createSequentialGroup()  .addContainerGap()  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  .addComponent(jButton5, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  .addComponent(jButton1)  .addComponent(jButton2)  .addComponent(jCheckBox1)  .addComponent(jButton3)  .addComponent(jButton4)  .addComponent(jCheckBox2)  .addComponent(jButton6)))  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  );  pack();  }// </editor-fold>  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {  if(jTextField1.getText().length() == 0){  JOptionPane.showMessageDialog(null, new String[] {"Введите массив"}, null, JOptionPane.ERROR\_MESSAGE);  return;  }    mainList.clear();  linesList.clear();  jPanel1.removeAll();  jPanel1.update(g2);  jPanel1.revalidate();    GetNums();  mainList.add(numArr.clone());  quickSort(0, numArr.length - 1);  if(jCheckBox1.isSelected()){  for(int i=0; i < mainList.size()+1; i++)  {    try {  jPanel1.removeAll();  jPanel1.revalidate();  jPanel1.update(g2);  //jPanel1.repaint();    //i++;  Draw(i);  Thread.sleep(750);  } catch (InterruptedException ex) {  Logger.getLogger(Form.class.getName()).log(Level.SEVERE, null, ex);  }  SetNums();  }  }  else  {  if(draw) Draw(mainList.size());  SetNums();  }  }    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {  JFileChooser chooser = new JFileChooser();  chooser.showOpenDialog(null);  File f = chooser.getSelectedFile();    try(Scanner in = new Scanner(f))  {  jTextField1.setText(in.nextLine());  }  catch(IOException ex){  System.out.println(ex.getMessage());  }  }  private void jCheckBox1StateChanged(javax.swing.event.ChangeEvent evt) {  if(jCheckBox1.isSelected()){  //jButton3.setEnabled(true);  jButton4.setEnabled(true);  }else{  jButton3.setEnabled(false);  jButton4.setEnabled(false);  }  }  private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {  if(curr\_pos == 0){  draw = false;  if(jTextField1.getText().length() == 0){  JOptionPane.showMessageDialog(null, new String[] {"Введите массив"}, null, JOptionPane.ERROR\_MESSAGE);  return;  }    mainList.clear();  linesList.clear();  jPanel1.removeAll();  jPanel1.update(g2);  jPanel1.revalidate();    GetNums();  mainList.add(numArr.clone());  quickSort(0, numArr.length - 1);  if(draw) Draw(mainList.size());  SetNums();  }else{  jButton3.setEnabled(true);  }    //JOptionPane.showMessageDialog(null, mainList.size());    if(curr\_pos + 1> mainList.size())  {  jButton4.setEnabled(false);  jButton3.setEnabled(true);  SetNums();  return;  }    //Нужно удалить всё и отрисовать заново  jPanel1.removeAll();  jPanel1.revalidate();  jPanel1.update(g2);  //jPanel1.repaint();    curr\_pos++;  Draw(curr\_pos);    }  private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {  if(curr\_pos - 2 < 0)  {  jButton3.setEnabled(false);  return;  }  jButton4.setEnabled(true);    //Нужно удалить всё и отрисовать заного  jPanel1.removeAll();  jPanel1.revalidate();  jPanel1.update(g2);  //jPanel1.repaint();    curr\_pos--;  Draw(curr\_pos);  }  private void jTextField1MouseClicked(java.awt.event.MouseEvent evt) {  curr\_pos = 0;  }  private void jCheckBox2ActionPerformed(java.awt.event.ActionEvent evt) {  if(!jCheckBox2.isSelected())  {  jCheckBox1.setEnabled(false);  jCheckBox1.setSelected(false);  }  else  {  jCheckBox1.setEnabled(true);  }  draw = jCheckBox2.isSelected();  }  private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {  JOptionPane.showMessageDialog(null, "QuickSort (Быстрая сортировка) - один из самых быстрых известных универсальных алгоритмов сортировки массивов.\n" + "Общая идея алгоритма состоит в следующем:\n"+  "1) Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива.\n" + "2) Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных\n" +  " отрезка, следующие друг за другом: «меньшие опорного», «равные» и «большие». \n" + "3) Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина\n" +  "отрезка больше единицы.\n");  }  private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {  mainList.clear();  linesList.clear();  jPanel1.removeAll();  jPanel1.update(g2);  jPanel1.revalidate();  jTextField1.setText("");  }  private void GetNums(){    String strArr[] = jTextField1.getText().split(" ");  if(jTextField1.getText().contains(".") || jTextField1.getText().contains(","))  {  JOptionPane.showMessageDialog(null, new String[] {"Введите только целые числа!"}, null, JOptionPane.ERROR\_MESSAGE);  }    numArr = new int[strArr.length];  try{  for(int i = 0; i < strArr.length; i++){  numArr[i] = Integer.parseInt(strArr[i]);  }  }  catch(NumberFormatException e)  {  ActionEvent evt = null;  if(jTextField1.getText().contains(".") || jTextField1.getText().contains(","))  {  jButton6ActionPerformed(evt);  return;  }  JOptionPane.showMessageDialog(null, new String[] {"Введите только целые числа!"}, null, JOptionPane.ERROR\_MESSAGE);  jButton6ActionPerformed(evt);  }  }    private void SetNums(){  String result = "";    for(int i = 0; i < numArr.length; i++)  result += Integer.toString(numArr[i]) + " ";  jTextField1.setText(result);    //JOptionPane.showMessageDialog(null, "Массив отсортирован, наслаждайся!");  }    private void quickSort(int left, int right) {  int i = left, j = right;  int tmp, tmp\_line[] = new int[3];  int pivot = numArr[(left + right) / 2];  /\* partition \*/  while (i <= j) {  System.out.println("Зашли в while");  System.out.print("i = ");  System.out.println(i);  System.out.print("j = ");  System.out.println(j);  while (numArr[i] < pivot){  i++;  System.out.print("i:");  System.out.print(i);  System.out.println(" numArr[i] < pivot");  }  while (numArr[j] > pivot){  j--;  System.out.print("j:");  System.out.print(j);  System.out.println(" numArr[j] > pivot");  }  if (i <= j) {  tmp = numArr[i];  numArr[i] = numArr[j];  numArr[j] = tmp;    if(i != j){  tmp\_line[0] = i;  tmp\_line[1] = j;  tmp\_line[2] = (left + right) / 2;  System.out.println("TMP-LINE:" + i + j + pivot);  System.out.println(numArr.toString());  linesList.add(tmp\_line.clone());  mainList.add(numArr.clone()); //ADD    }  i++;  j--;      }  }  System.out.println("Конец итерации");  /\* recursion \*/  if (left < j){  quickSort(left, j);  }  if (i < right){  quickSort(i, right);  }  }    private void Draw (int step) {  if(!draw && !jCheckBox2.isSelected()) return;    for(int i = 0; i < step; i++){  for(int q = 0; q < numArr.length; q++){  if(i < step - 1 && (q == linesList.get(i)[0] || q == linesList.get(i)[1] || q == linesList.get(i)[2])){  if(q == linesList.get(i)[2]){  g2.setColor(Color.red);  g2.setStroke(new BasicStroke(2));  }else{  g2.setColor(Color.blue);    g2.drawLine(-5 + linesList.get(i)[0]\*(radius+10) + radius, 5 + 40\*i + radius, -5 + linesList.get(i)[1]\*(radius+10) + radius, 5 + (i+1)\*40);  g2.drawLine(-5 + linesList.get(i)[1]\*(radius+10) + radius, 5 + 40\*i + radius, -5 + linesList.get(i)[0]\*(radius+10) + radius, 5 + (i+1)\*40);    g2.setStroke(new BasicStroke(2));  }  }  g2.drawString(Integer.toString(mainList.get(i)[q]), 12 + 30\*q, 20 + 40\*i);  g2.draw(new Ellipse2D.Float(5 + q\*(radius+10), 5 + 40\*i , radius, radius));    g2.setStroke(new BasicStroke(1));  g2.setColor(Color.BLACK);  }  }    jPanel1.revalidate();  }    public static void main(String args[]) {  try {  for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  if ("Nimbus".equals(info.getName())) {  javax.swing.UIManager.setLookAndFeel(info.getClassName());  break;  }  }  } catch (ClassNotFoundException ex) {  java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (InstantiationException ex) {  java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (IllegalAccessException ex) {  java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  } catch (javax.swing.UnsupportedLookAndFeelException ex) {  java.util.logging.Logger.getLogger(Form.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  }    java.awt.EventQueue.invokeLater(new Runnable() {  public void run() {  new Form().setVisible(true);  }  });  }  // Variables declaration - do not modify  private javax.swing.ButtonGroup buttonGroup1;  private javax.swing.JButton jButton1;  private javax.swing.JButton jButton2;  private javax.swing.JButton jButton3;  private javax.swing.JButton jButton4;  private javax.swing.JButton jButton5;  private javax.swing.JButton jButton6;  private javax.swing.JCheckBox jCheckBox1;  private javax.swing.JCheckBox jCheckBox2;  private javax.swing.JPanel jPanel1;  private javax.swing.JTextField jTextField1;  // End of variables declaration  } |