МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

Тема: Сортировка слабой кучей

Студент гр. 9381 Птичкин С.А.

Студент гр. 9381 Прибылов Н.А.

Студент гр. 9383 Ноздрин В.Я.

Руководитель Фирсов М.А.

Санкт-Петербург
2021

ЗАДАНИЕ

НА УЧЕБНУЮ ПРАКТИКУ

Студент Птичкин С.А. группы 9381

Студент Прибылов Н.А. группы 9381

Студент Ноздрин В.Я. группы 9383

Тема практики: сортировка слабой кучей

Задание на практику:

Командная итеративная разработка визуализатора алгоритма(ов) на Java с графическим интерфейсом.

Алгоритм: сортировка слабой кучей.

Сроки прохождения практики: 01.07.2021 – 14.07.2021

Дата сдачи отчета: 14.07.2020

Дата защиты отчета: 14.07.2020

Студент		Птичкин С.А.
Студент		Прибылов Н.А.
Студент		Ноздрин В.Я.
Руководитель		Фирсов М.А.

АННОТАЦИЯ

Целью текущей учебной практики является разработка GUI приложения реализующего алгоритм сортировки слабой кучей. Программа разрабатывается на языке Java бригадой из трех человек, каждый из которых выполняет свою роль.

Сдача и показ проекта на определенном этапе выполнения осуществляется согласно плану разработки.

SUMMARY

The goal of the current training practice is to develop a GUI application that implements the weak heap sorting algorithm. The program is developed in Java by a team of three people, each of whom plays a role.

The delivery and display of the project at a certain stage of implementation is carried out according to the development plan.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью текущей учебной практики является разработка приложения с графическим интерфейсом (GUI), которое выполняет сортировку введенного массива чисел алгоритмом сортировки слабой кучей. Программа предоставляет пользователю интерфейс для ввода и вывода данных, а также для запуска основного алгоритма. Так как алгоритм сортировки основан на свойствах слабой кучи, а слабая куча сама по себе является бинарным деревом, в программе реализовано графическое представление структуры в виде дерева с текстовыми подсказками о шагах алгоритма и выделениями разными цветами для наглядности. Имеется возможность выполнения алгоритма в пошаговом режиме.

1. ТРЕБОВАНИЯ К ПРОГРАММЕ

1.1. Исходные Требования к программе

Программа визуализирует алгоритм сортировки массива чисел с помощью структуры слабой кучи.

Приложение должно выполнять следующие функции:

1. Задание входных данных с помощью текстового поля.
2. Задание входных данных с помощью элементов графического интерфейса путём нажатия на кнопки, которые будут отражать массив элементов, и выбора соответствующей функции в контекстном меню, добавление элемента через отдельную кнопку с символом “+”.
3. Удаление и обмен элементов, с помощью контекстного меню кнопок.
4. Применение алгоритма сортировки к введённым данным путём нажатия соответствующей кнопки.
5. Визуализация элементов массива набором кнопок, пошаговая визуализация самого алгоритма. На каждом шаге будет отражаться текущее состояние массива и бинарного дерева. Отсортированная часть массива будет выделена специальным цветом. На первом этапе, построении кучи визулиализируется каждое перемещение элементов, при этом элементы подсвечиваются специальным цветом. На втором этапе, непосредственно сортировки визуализируются операции удаления и перемещения элементов, которые также окрашиваются в специальный цвет. На экране рядом с деревом будет текст при построении кучи "Этап 1. Построение кучи" а при непосредственно сортировке "Этап 2. Сортировка".
6. Считывание входных данных из файла и запись результатов в файл.
7. Взаимодействие с пользователем посредством текстовых сообщений, поясняющих управление приложением и шаги работы алгоритма, указывающие, какие элементы удаляются, перемещаются и сравниваются на каждой операции.

1.2. Уточнение требований после сдачи прототипа.

Замечания:

1. (Выполнено) При изменении размеров окна должны растягиваться/сужаться те элементы, для которых это имеет смысл.

2. (Выполнено) Нужно добавить поле с текстовыми пояснениями, выводимыми по ходу алгоритма.

1.3. Уточнение требований после сдачи второй версии.

Замечания:

1.1. (Выполнено) При каждом нажатии на "Шаг" должно быть видно какое-то изменение (на рисунке, и/или в массиве, и/или в пояснениях).

1.2. (Выполнено) Если элементы нужно будет обменять, то не следует писать "но не меняем". На тех шагах, на которых происходит сравнение, следует писать именно про сравнение, а на тех шагах, где происходят обмены, должны быть сообщения "...не меняем..." или "...меняем...".

2. (Выполнено) Массив над рисунком должен располагаться в одну линию при возможности (в том числе если пользователь увеличил ширину окна).

3. (Выполнено) На рисунке должно явно показываться, где установлены биты вращения. Это должно быть видно не только по индексам под узлами.

4. (Выполнено) Чи́сла в массиве над рисунком после запуска алгоритма серые, в результате они плохо видны, когда элемент окрашивается. Делайте чи́сла белыми или чёрными (чтобы были хорошо видны).

5. (Выполнено) В текстовых пояснениях должны быть сообщения об установках битов вращения и соответствующих изменениях в куче.

6. (Выполнено) Сохранение текущего массива должно быть доступно на любом шаге.

7. (Выполнено) Добавить кнопку "Предыдущее состояние", при нажатии на которую открывается окошко, в котором будет показываться массив и рисунок в том виде, в каком они были на предыдущем шаге.

1.4. Пример работы визуализации алгоритма.

Пусть на текущем шаге минимальный элемент (34) находится в корне кучи. Следующим шагом должна быть смена элемента в корне (34) и последнего элемента в куче (42).

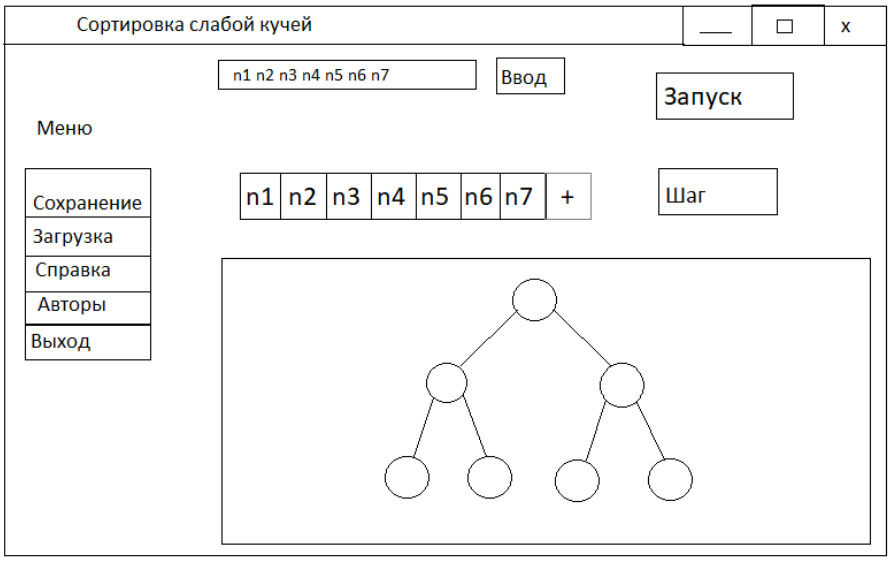
Красим эти два элемента (например, в красный) в куче и массиве, после чего производим следующий шаг (обмен):

В куче последний элемент 42 становится корнем, а элемент 34 исчезает из кучи; в массиве элементы 34 и 42 меняются местами. Элемент 34 становится частью отсортированного куска массива и красится в цвет, соответствующий отсортированному куску (например, серый).

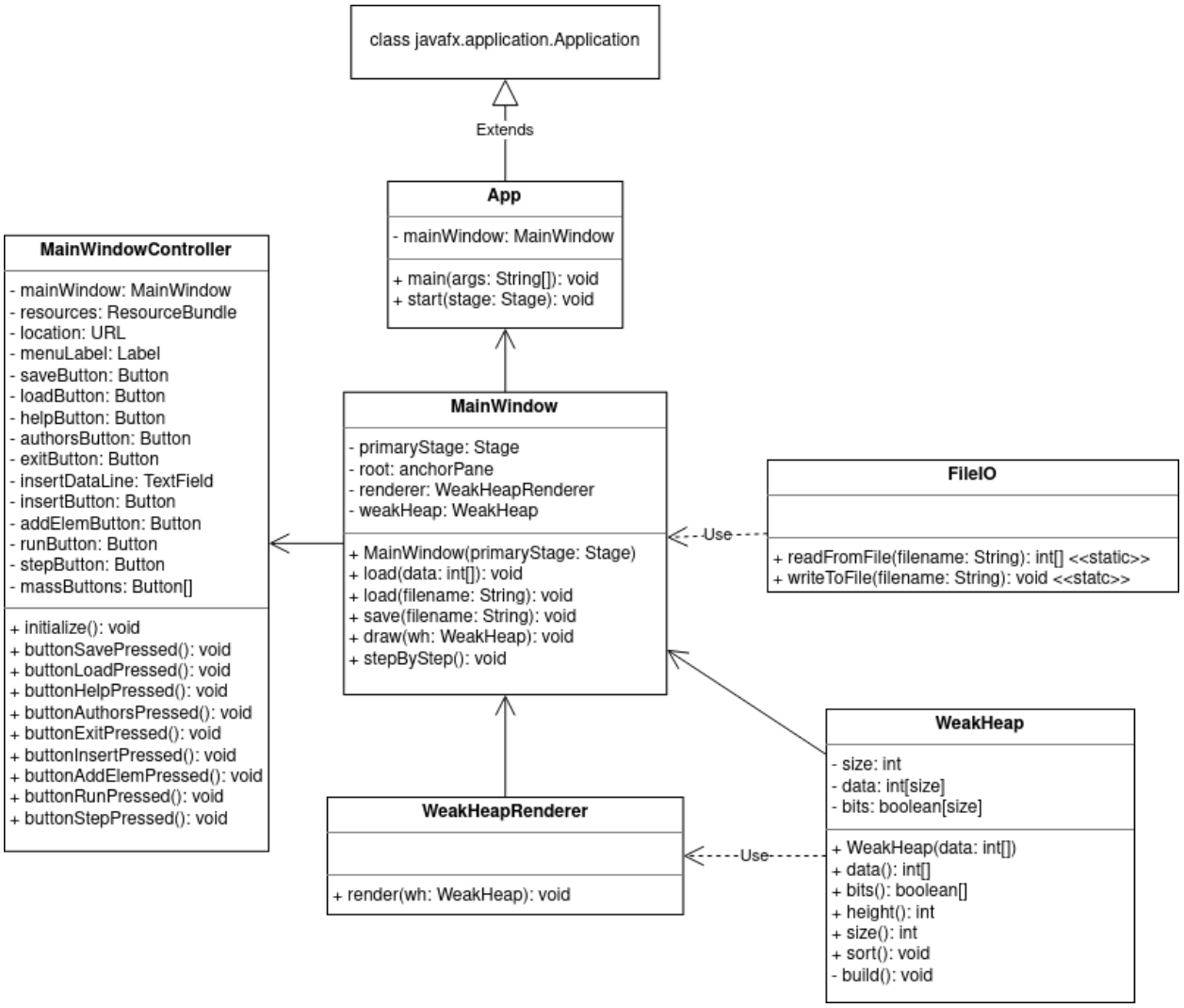
В качестве текстового пояснения данного шага пользователю выводится сообщение по типу:

"В корне находится минимальный элемент 34. Меняем его местами с последним элементом неотсортированной части массива 42. Удаляем элемент 34 из корня. Теперь элемент 34 принадлежит отсортированной части массива."

1.5. Эскиз интерфейса.



1.6. UML диаграмма классов.



2. ПЛАН РАЗРАБОТКИ И РАСПРЕДЕЛЕНИЕ РОЛЕЙ В БРИГАДЕ

2.1. План разработки.

1) 5 июля - согласование спецификации и плана.
2) 7 июля - прототип (интерфейс)
3) 9 июля - 1 версия (реализация алгоритма, некоторые функции графического интерфейса, визуализация результата работы алгоритма)
4) 11 июля - 2 версия (загрузка, сохранение данных в файловой системе, пошаговая визуализация работы алгоритма, добавление текстового описания шагов алгоритма, защита приложения от некорректных действий пользователя)
5) 14 июля - финальная версия (обработка программой исключительных ситуаций, доработка уже реализованных возможностей приложения).

2.2. Распределение ролей в бригаде.

1. Птичкин Сергей - реализация графического интерфейса.
2. Ноздрин Василий - реализация алгоритма сортировки, считывание данных, работа с файлами.
3. Прибылов Никита - визуализация алгоритма, тестирование, сборка проекта.

3. ОСОБЕННОСТИ РЕАЛИЗАЦИИ

3.1. Описание алгоритма

Куча — это структура данных, в которой значение в каждом узле меньше значений в правом и левом потоках.

Слабая куча — это структура данных типа дерево с правилом. Для каждой вершины выполнено, что значение в этой вершине меньше или равно значениям в правом поддереве исходящем из нее. Для того, чтобы куча была кучей, у корня не делают левого поддерева.

Главное свойство слабой кучи — минимальный элемент в ней лежит в корне.

Сортировка основана на том, чтобы удалять минимум из кучи, помещать его в последний элемент массива, и уменьшать “активную” часть массива на единицу. Далее выполняется просеивание и слабая куча становится корректной. Так выполняется операция delMin кучи (стандартная операция сортирующих деревьев). Действие повторяется до тех пор, пока весь массив не будет отсортирован.

Существует несколько операций в куче.

Операция Join(v, w) сравнивает элементы v и w и пытается протолкнуть минимум из v в w. Если элементы меняются, у нижнего из них меняются правое и левое поддеревья местами. Это нужно для сохранения структуры.

Операция SiftDown меняет местами элементы с индексом 0 и length-1 местами и уменьшает length на 1. При этом сам массив остается прежнего размера. Таким образом отделяется отсортированная часть данных от кучи.

Операция Up(v) возвращает первую вершину-предка, для которой вершина v является правым потомком.

Операция Build() вызывает метод Join() передавая в первый аргумент все вершины по очереди с конца массива, а во второй элемент “правого отца” этих вершин. “Правым отцом” называется Up(v)

3.2. Структуры данных

3.2.1. Класс WeakHeap.

Класс WeakHeap реализует структуру слабой кучи и набор стандартных методов для работы с ней: построение кучи, просеивание кучи, удаление минимума, а также сортировка и методы для пошаговой реализации построения и сортировки. Данные представлены в виде двух массивов — массива со значениями и массива битов. Массив битов нужен для того, чтобы можно было легко менять местами правые и левые поддеревья у выбранного узла.

Ниже приведены некоторые формулы расчета индексов элементов-родителей и элементов-детей:

V -> 2*V+bit[V] — левый ребенок

V -> 2*V+1-bit[V] — правый ребенок

V -> V/2 — родитель

3.2.2. Класс Controller.

Класс, реализующий графический интерфейс приложения.

3.2.3. Класс EditableButton extends Button

Класс, реализующий кнопку с текстовым полем и контекстным меню.

3.2.4. Класс MainWindow.

Класс содержащий окно и структуру данных слабой кучи.

3.2.5. Класс FileIO.

Класс, отвечающий за запись на диск и чтение с него.

3.2.6. Класс Main extends Application.

Класс создает экземпляр MainWindow, содержит точку входа, функцию main()

3.2.7. Класс stepState.

Класс используется как оболочка для сохранения состояния кучи.

3.2.8. Класс WeakHeapRenderer.

Класс, реализующий рисование дерева кучи.

3.2.9. Класс PrevStepController.

Класс, отвечающий за интерфейс окна, на котором выводится состояние предыдущего шага алгоритма.

3.3. Основные методы

Методы класса WeakHeap.

1. public boolean join(int v, int w) — проталкивает минимум из вершины w в v, если нужно, сохраняя свойство структуры слабой кучи.
2. public int up(int v) — возвращает первую вершину для которой v является правым потомком.
3. void siftUp(int v) — выполняет просеивание кучи сверху. Это нужно для добавления новых элементов в кучу.
4. int siftDown() — выполняет просеивание кучи снизу. Это нужно для удаление минимума из кучи.
5. public void buildStep(), public void build() — методы, реализующие построение кучи, то есть куча становится корректной слабой кучей.
6. public boolean heapsortStep(), public void heapsort() — методы, реализующие алгоритм сортировки.
7. public List<Integer> allChildrenOf(int id) — возвращает список всех потомков узла.

Методы класса Main.

1. public void start(Stage primaryStage) throws Exception - метод, в котором создаётся экземпляр MainWindow.
2. public static void main(String[] args) - главный метод программы, где находится точка входа, и запускается приложение.

Методы класса MainWindow.

1. public MainWindow(Stage stage) - конструктор класса, принимающий окно приложения.
2. public int[] sort(Integer[] data) - метод, создающий на основе массива целых чисел экземпляр структуры WeakHeap, и вызывающий методы её сортировки без пошагового режима. Возвращает отсортированный массив чисел.
3. public void startStepSort(Integer[] data, AnchorPane drawField) - метод создающий экземпляр структуры, которая предназначена для пошагового режима алгоритма. Также рисует её начальное дерево в переданную панель.
4. public stepState stepSort(AnchorPane drawField) - метод, вызывающий шаг алгоритма сортировки/ построения слабой кучи, обрабатывает текущие состояния кучи и возвращает данные для визуализации каждого шага.
5. public Stage getPrimaryStage() - метод возвращающий текущее окно приложения.
6. public String readData(File source) - метод создающий и инициализирующий объект класса FileIO, с последующим вызовом метода считывания из файла, возвращает считанные данные, если не поймано исключение.
7. public void writeData(File destination, int[] data) - метод создающий и инициализирующий объект класса FileIO, с последующим вызовом метода записи в файл массива чисел.

Методы класса Controller.

1. void initialize() - метод, в котором создаётся дополнительное окно, на котором отображается предыдущий шаг алгоритма. Также задаётся обработчик события изменения размера окна, который располагает кнопки так, чтобы они заполняли всю доступную длину.
2. private void button...Pressed() - множество методов, каждый из которых вызывается при нажатии соответствующей кнопки в окне приложения. И каждый из которых выполняет соответствующие инструкции. Ничего не возвращает.
3. private void addElemToBox(Button newElem) - метод, добавляющий кнопку в панель, визуализирующую массив элементов. С корректным перемещением кнопки “+”.
4. private boolean isStringCorrect(String str) - проверяет строку на корректное представление одного числа.
5. private int[] stringToIntArray(String str) throws NumberFormatException - превращает строку чисел в массив чисел, бросает исключение в случае некорректной строки.
6. private void stringToButtons(String str) throws NumberFormatException - превращает строку чисел в массив кнопок с соответствующими значениями, бросает исключение в случае некорректной строки.

Методы класса FileIO.

1. public FileIO(File resource) - конструктор класса.
2. public void writeToFile(int[] data) - метод, записывающий массив чисел в файл, который передан конструктору при создании объекта класса.
3. public String readFromFile() - метод, считывающий всё содержимое файла в строку и возвращающий её. Файл должен быть задан при создании объекта класса.

Методы класса WeakHeapRender.

1. public static void render(WeakHeap wh, AnchorPane drawField,
   List<Integer> actionNodes, Paint actionColor,
   List<Integer> childrenOfActionNodes, Paint actionColorForChildren)
   - отображает кучу wh на поле drawField, окрашивая узлы actionNodes, над которыми только что были произведены действия, в цвет actionColor, и узлы childrenOfActionNodes, (пра)родитель которых поменял бит, в другой цвет actionColorForChildren.

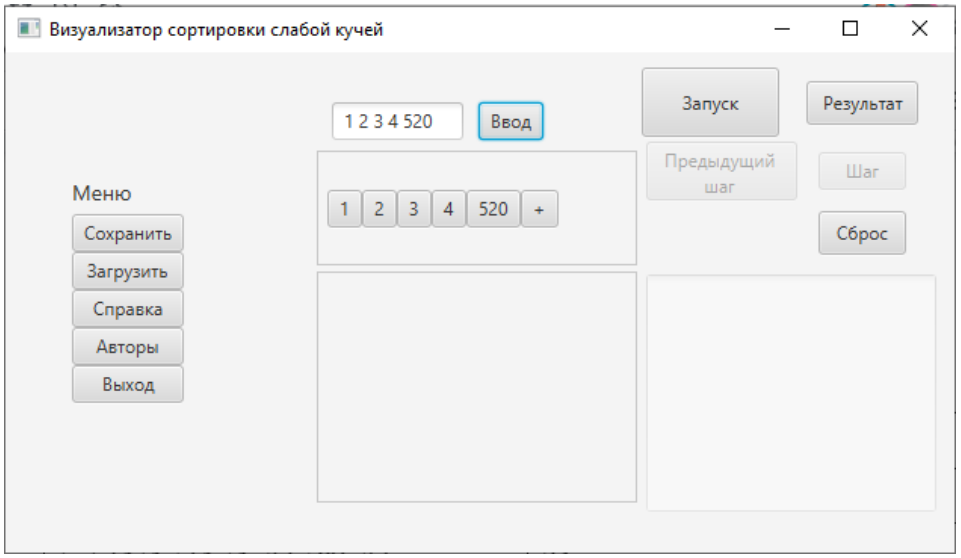
Методы класса PrevStepController.

1. public void setPrevStepState(ArrayList<Controller.EditableButton> massButtonElem, String textAreaMessage, ObservableList<Node> list)
2. void initialize() - метод, в котором задаётся обработчик события изменения размера окна, который располагает кнопки так, чтобы они заполняли всю доступную длину.

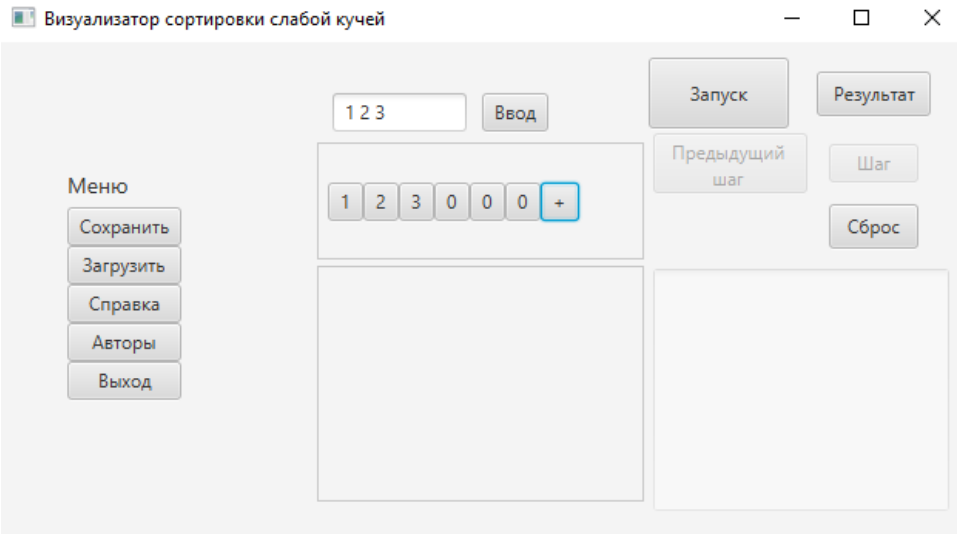
4. ТЕСТИРОВАНИЕ

4.1. Тестирование графического интерфейса.

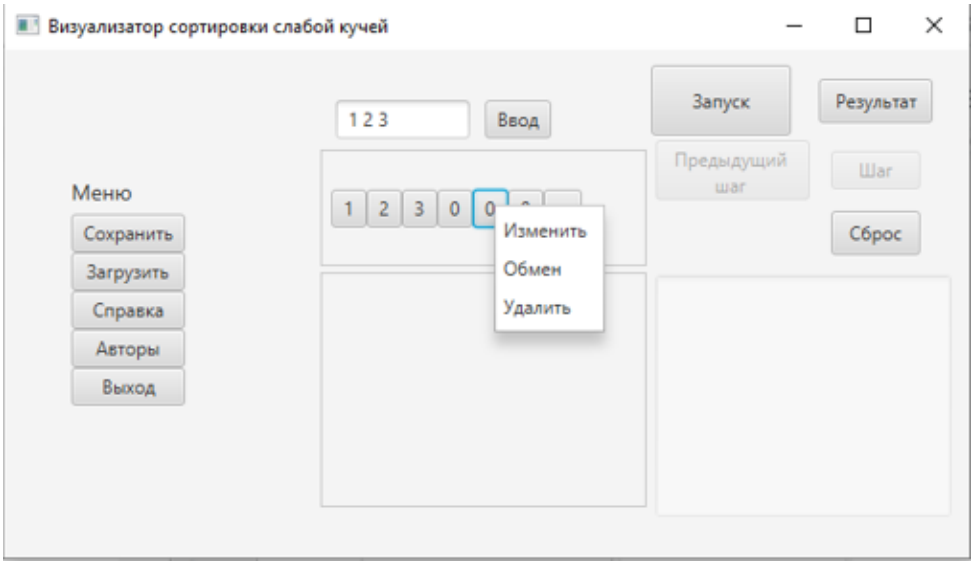
1. Ввод данных из текстового поля.



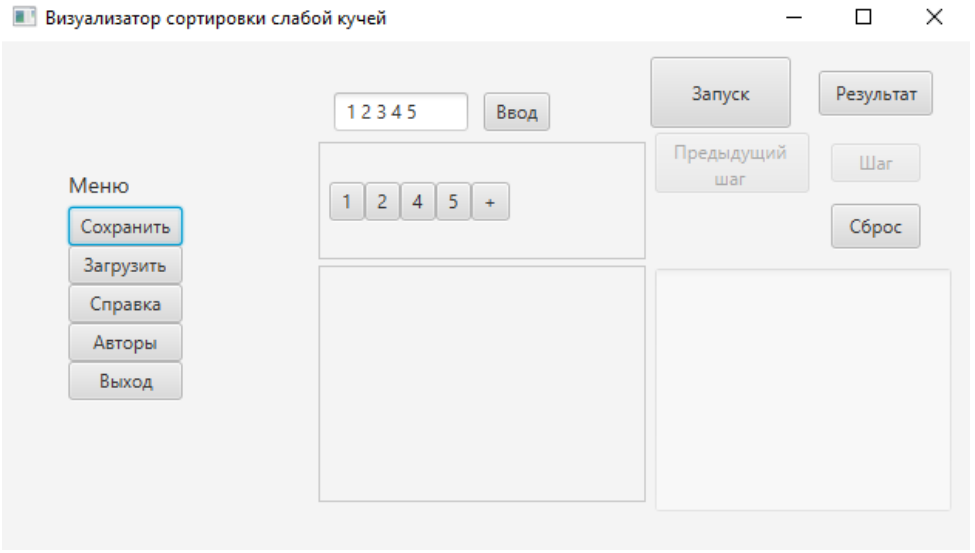
2. Добавление элементов через кнопку +.



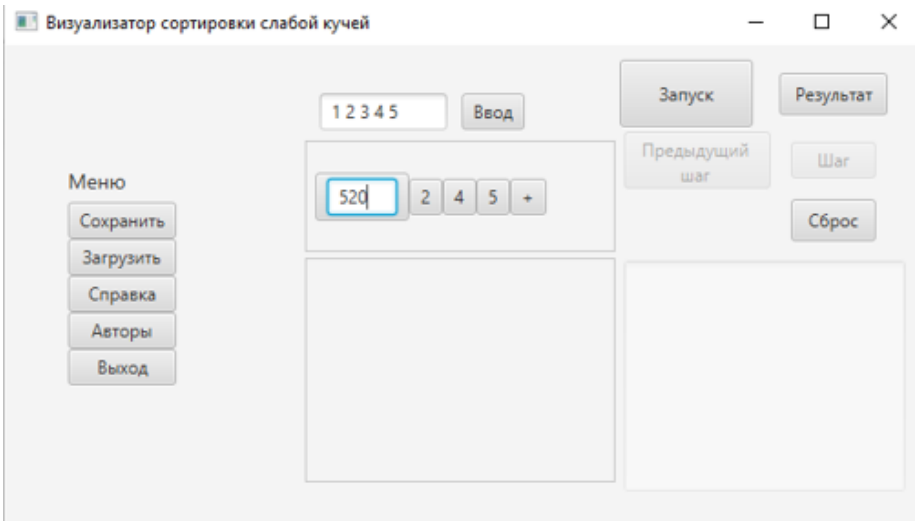
3. Вывод контекстного меню при нажатии.



4. Удаление элемента с помощью контекстного меню.

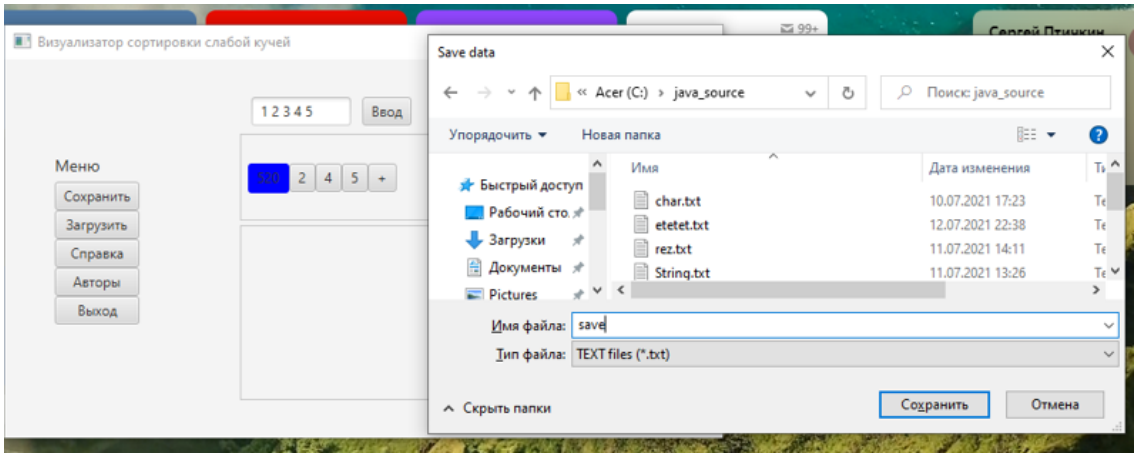


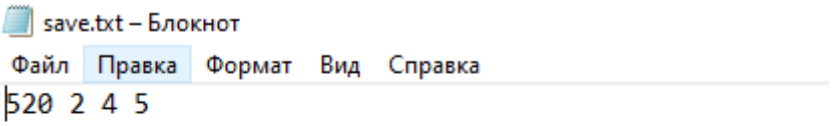
5. Изменение значения элемента с помощью контекстного меню.



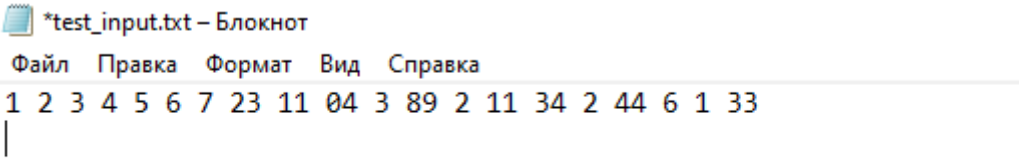
6. Обмен элементов с помощью контекстного меню.

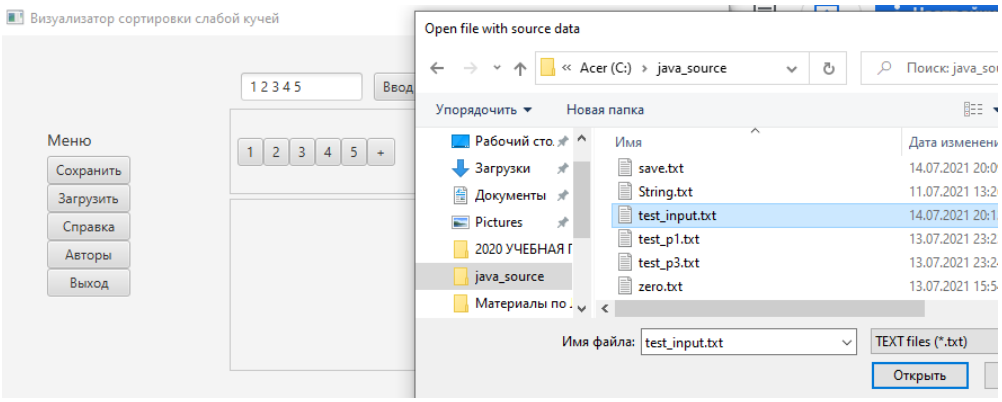
7. Сохранение текущего состояния массива.

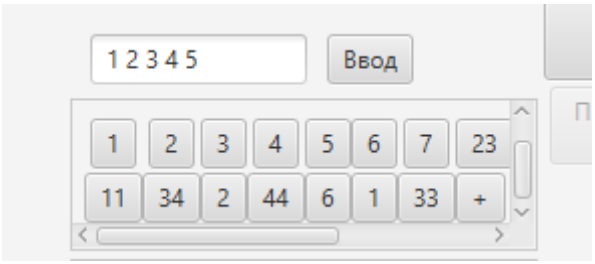




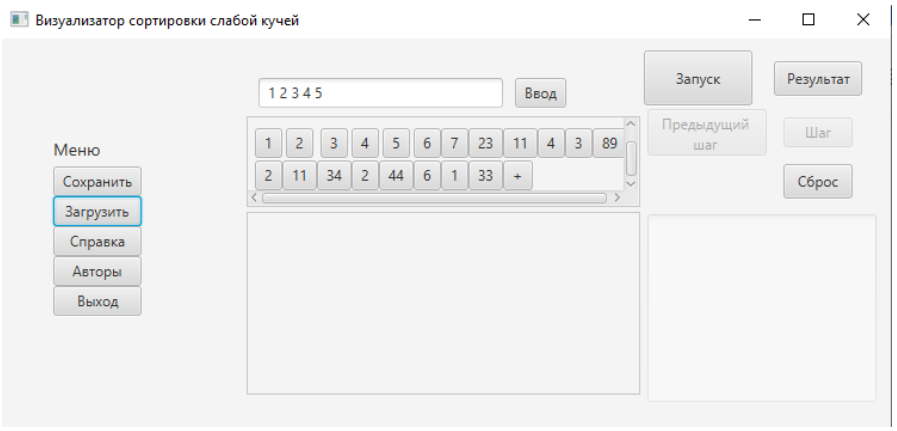
8. Загрузка данных из файла.



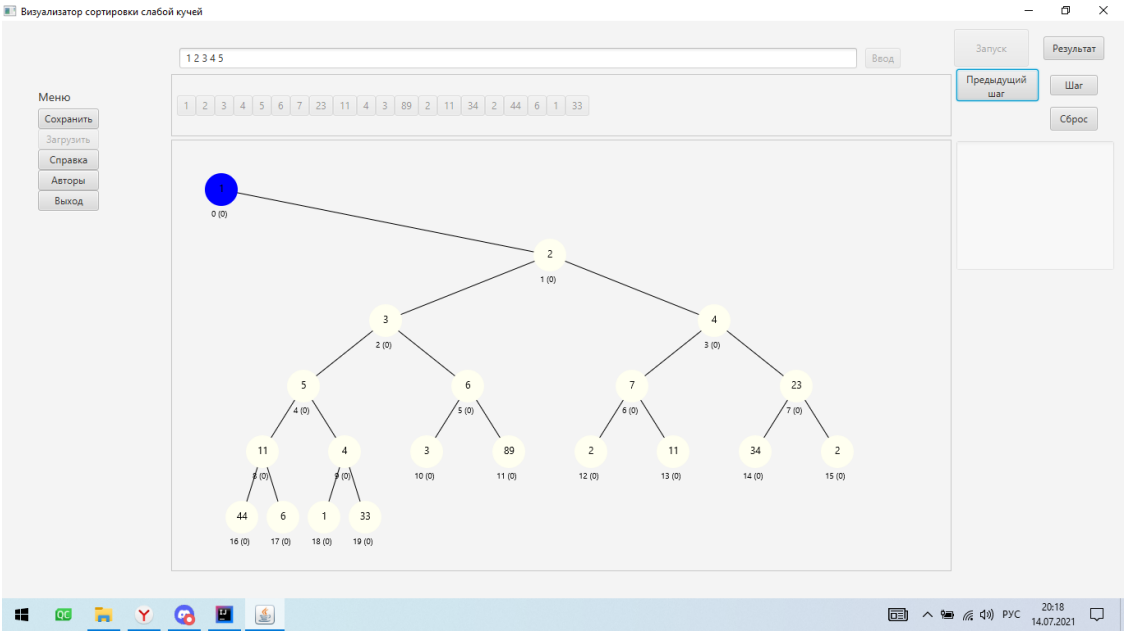




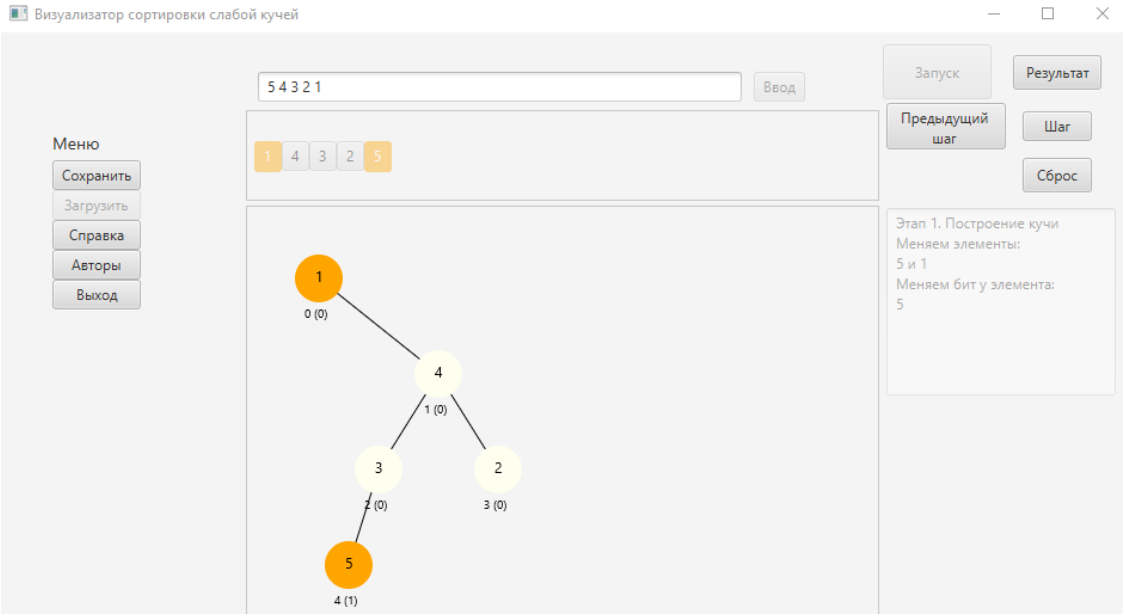
9. Перестройка элементов при изменении ширины окна.

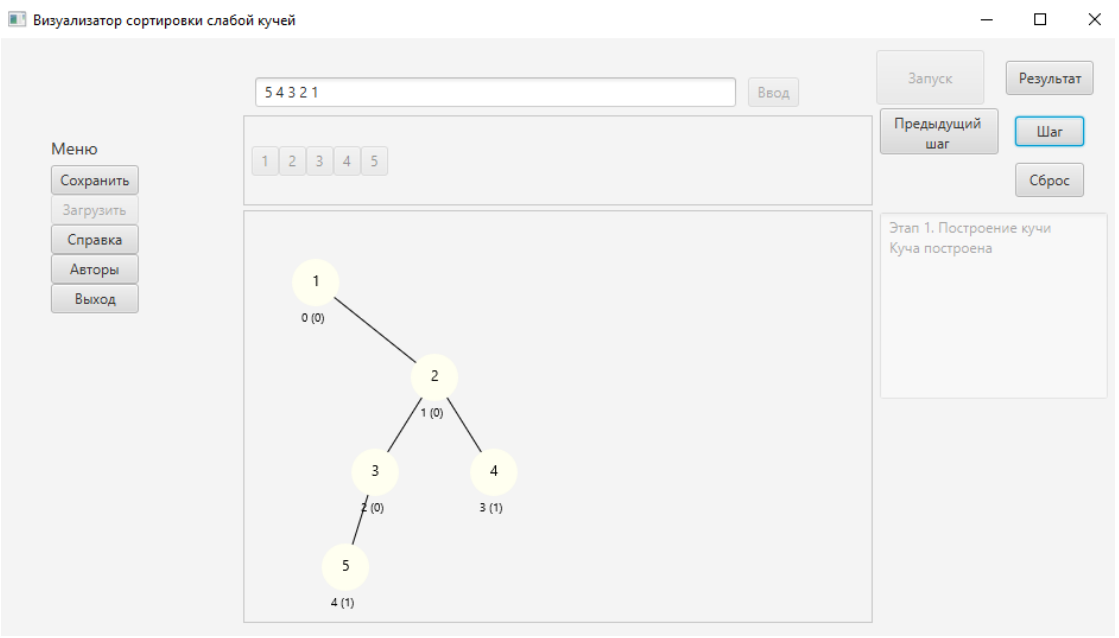


10. Запуск алгоритма и отрисовка дерева кучи.

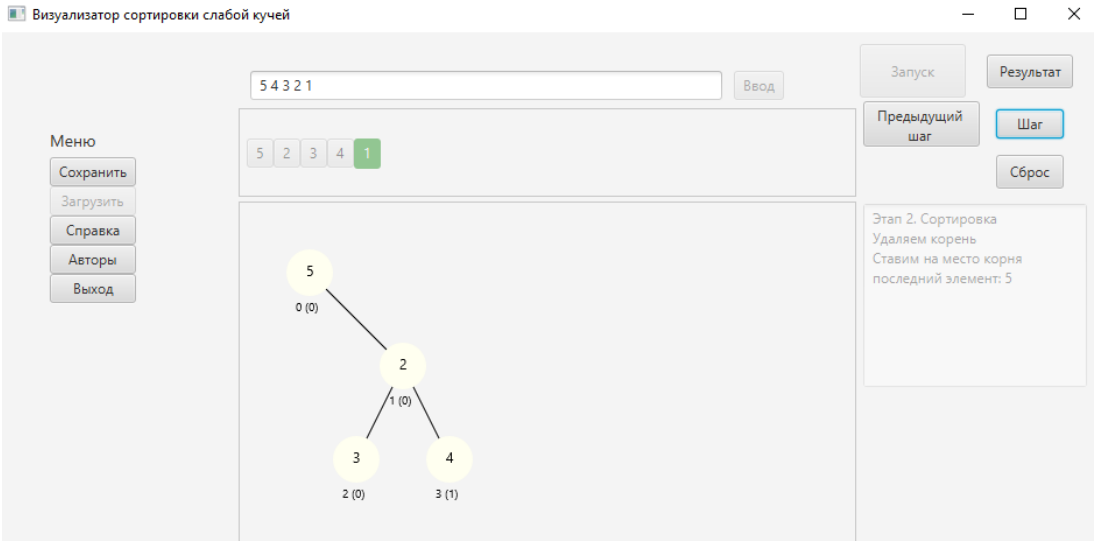


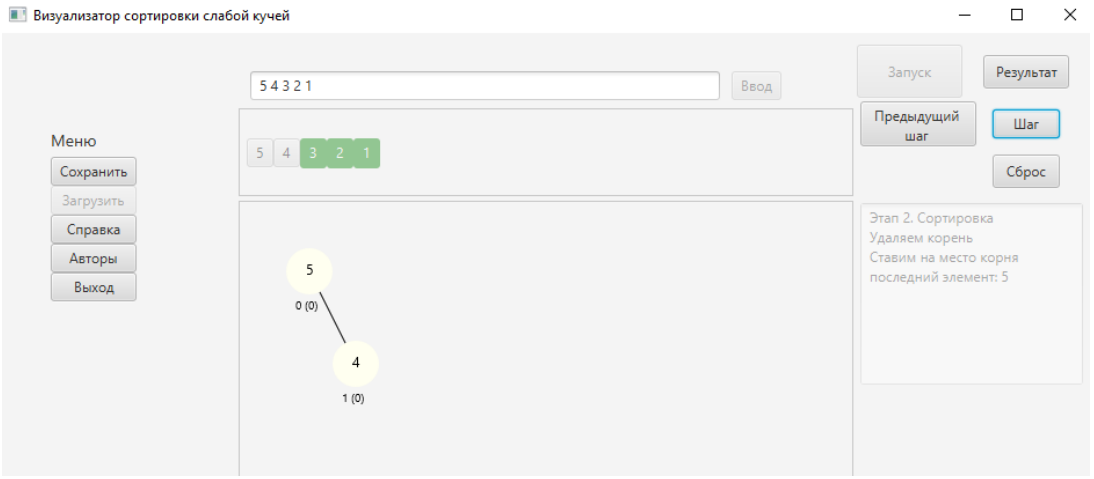
11. Пошаговое построение кучи.

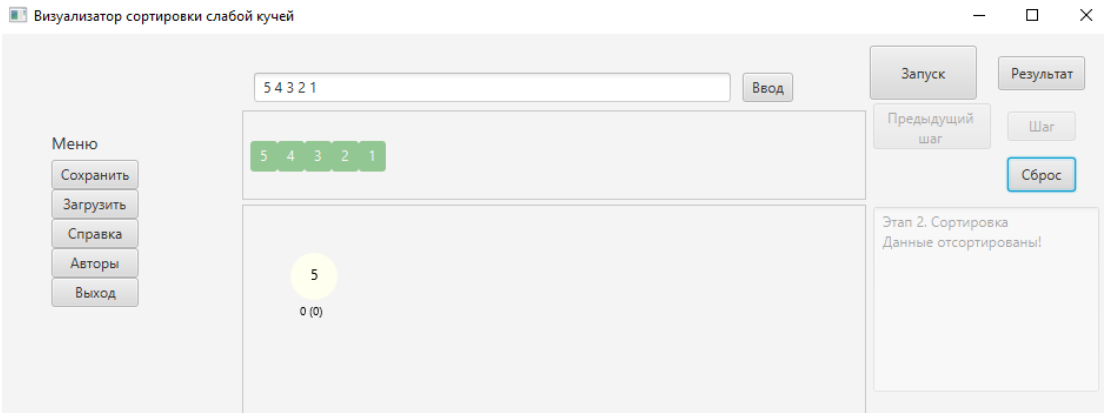




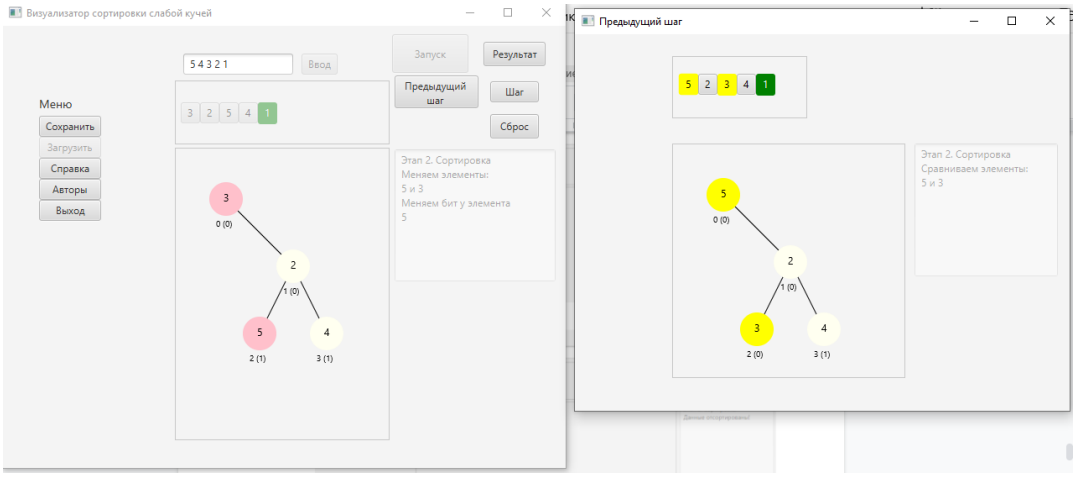
12. Визуализация пошаговой сортировки.



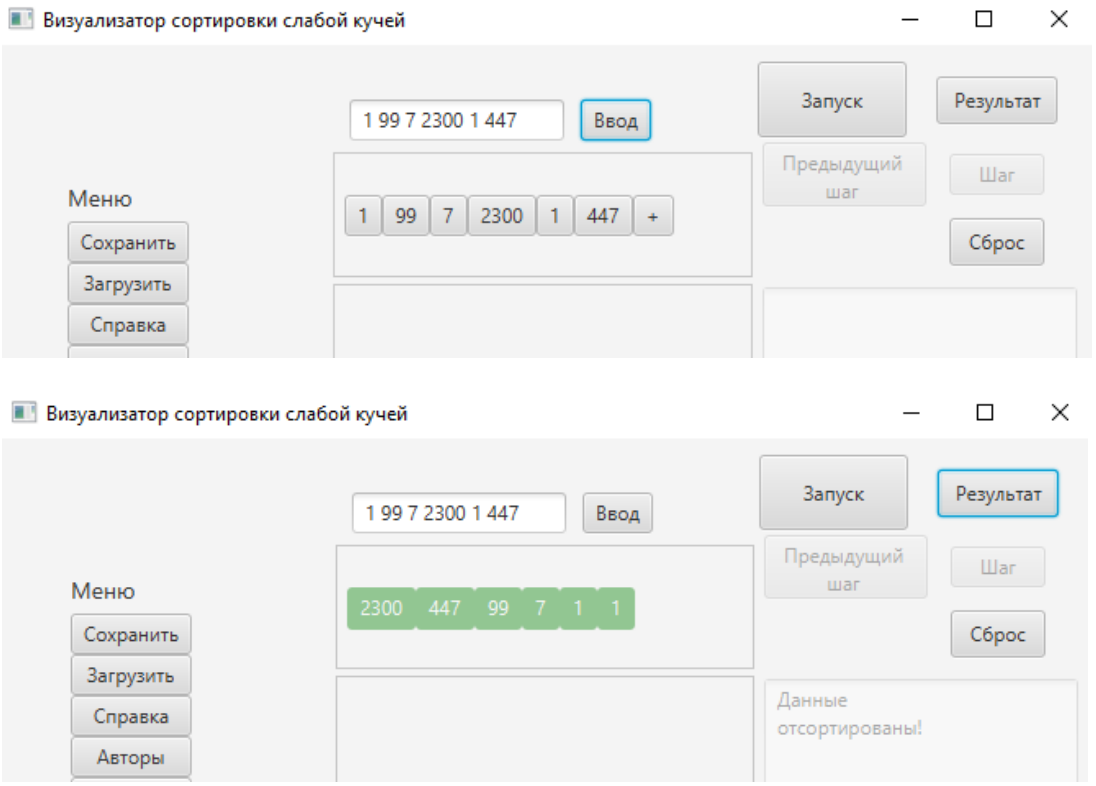




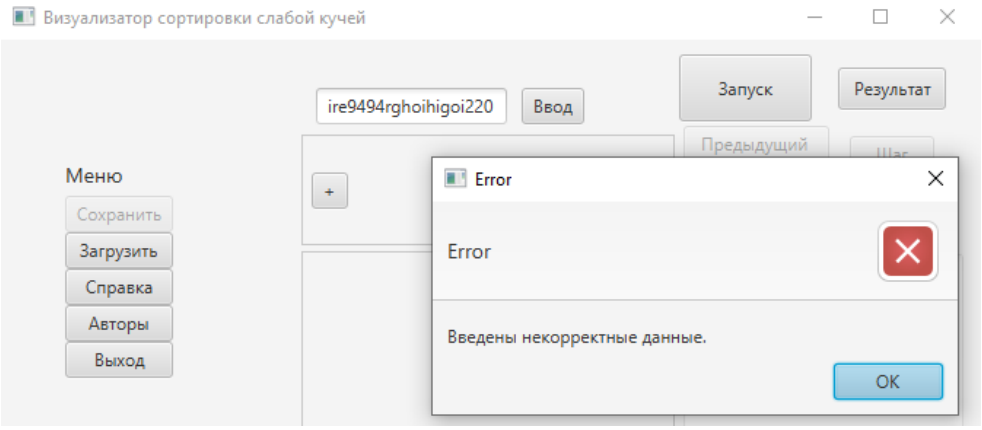
13. Отображение предыдущего шага в отдельном окне.



14. Получение результата без пошагового режима.



15. Попытка ввести некорректные данные.



4.2. Тестирование корректности введённых данных.

Входные данные	Результат
1 2 3 4	Корректная строка
2 -7 1828 +18 28 -45 +90 -45	Корректная строка
3 14 15 92 6	Корректная строка
1 2f 3 4	Некорректная строка
2 - 7 15	Некорректная строка
1 2 3 4	Корректная строка
20 -30 -40 50	Корректная строка

4.3. Тестирование построения слабой кучи.

Входные данные	Результат
8 7 6 5 4 3 2 1	значения: 1 2 3 4 5 6 7 8 биты: 0 0 0 0 0 0 0 0
1 2 3 4 5 6	значения: 1 2 3 4 5 6 биты: 0 0 0 0 0 0
30 8 1 7 60 2	значения: 1 2 8 7 60 30 биты: 0 0 0 0 0 0
1 5 12 3 3 3 8 1 9 5	значения: 1 1 3 5 3 12 8 3 9 5 биты: 0 0 0 1 0 0 1 0 0 0
1 2 3 4 8 5 7 4 10 9	значения: 1 2 3 4 8 5 7 4 10 9 биты: 0 0 0 0 0 0 0 0 0 0
1	значения: 1 биты: 0

4.4. Тестирование сортировки слабой кучей.

Входные данные	Результат
1 2 3 4 5 6 7 8	8 7 6 5 4 3 2 1
6 5 4 3 2 1	6 5 4 3 2 1

1 5 12 3 3 3 8 1 12 5	12 12 8 5 5 3 3 3 1 1
1 2 3 4 3 5 5 4 10 9	10 9 5 5 4 4 3 3 2 1
5 5 5 5 5 5	5 5 5 5 5 5
2 3	3 2
1	1
55 62 65 54 56 54 59 59 55 59 42 47 59 59 50 66 56 46 45 69 50 66 56 51 53 46 65 69 42 53 44 69 67 65 65 51 51 49 69 68 63 58 68 68 48 58 48 44 45 53 48 44 61 44 58 43 56 63 66 55 46 69 54 43 64 50 42 53 45 58 53 42	69 69 69 69 69 68 68 68 67 66 66 66 65 65 65 65 64 63 63 62 61 59 59 59 59 59 58 58 58 58 56 56 56 56 55 55 55 54 54 54 53 53 53 53 53 51 51 51 50 50 50 49 48 48 48 47 46 46 46 45 45 45 44 44 44 44 43 43 42 42 42 42

ЗАКЛЮЧЕНИЕ

Была реализована программа-визуализатор сортировки целых чисел слабой кучей с моментальным и пошаговым режимами работы и текстовыми пояснениями к действиям алгоритма. Программой предусмотрены загрузка данных из текстового поля, набора кнопок и файла, сохранение текущего состояния (в т.ч. и результата) в файл, наглядное отображение кучи в виде дерева с покраской узлов при пошаговом режиме работы.

В ходе работы были получены знания о разработке на языке Java, работе с графическим интерфейсом, а также навыки командной разработки.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Хорстманн Кей С. Java. Библиотека профессионала. Том 1. Основы. М.: Вильямс, 2018. 864 с.
2. Эккель Брюс. Философия Java. П.: Питер, 2019. 1168 с.
3. Сортировка слабой кучей // Хабр. URL: https://habr.com/ru/company/edison/blog/499786/
4. Документация Java // Java Documentation. URL: https://docs.oracle.com/en/java/
5. Java. Базовый курс // Stepik. URL: https://stepik.org/course/187/syllabus

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРИЛОЖЕНИЯ

Файл: Main.java

```
package sample;

import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {
      private MainWindow mainwindow;
      @Override
      public void start(Stage primaryStage) throws Exception{
      mainwindow = new MainWindow(primaryStage);
      }

      public static void main(String[] args) {
      launch(args);
      }
}
```

Файл: WeakHeap.java

```
package sample;

import java.util.ArrayList;
import java.util.List;

public class WeakHeap {
      // Data structure fields
      int length;
      int[] values;
      int[] bits;
      /* to add element -> add to the bottom and siftUp
      * to remove element -> swap with last and siftDown
```

```
* */

// Algorithm fields
public enum State {
initial,
preBuilding,
building,
built,
delMin,
preSiftDown,
siftDown,
done
}

public State state = State.preBuilding;
public int joinId1 = 0;
public int joinId2 = 0;
int initialLength;
int buildIterator;

WeakHeap(Integer[] data) {
this.length = data.length;
this.values = new int[length];
this.bits = new int[length];
for (int i = 0; i < length; i++) {
      this.values[i] = data[i];
      this.bits[i] = 0;
}
initialLength = length;
buildIterator = length-1;
}

public boolean join(int v, int w) {
if (values[v] < values[w]) {
      // swap values
      values[v] ^= values[w];
      values[w] ^= values[v];
      values[v] ^= values[w];
```

```
            // change bit
            bits[v] = 1 - bits[v];
            return true;
        }
        return false;
        } // проталкивает минимум из v наверх в w

        public int up(int v) {
        if (((v % 2) ^ (bits[v / 2])) == 1)
            return v / 2;
        return up(v / 2);
        } // returns Right parent

        void siftUp(int v) {
        int w;
        while (join(v, w = up(v)))
            v = w;
        } // sifts WeakHeap

        int siftDown() {
        if (length < 1) {
            return values[0];
        }
        int min = values[0];                    // save min
        length--;                               // delete min
        values[0] ^= values[length];            // swap values
        values[length] ^= values[0];            // .
        values[0] ^= values[length];            // .
        if (length < 2)                         // 1-node case (only root)
            return min;                         // .
        int v = 1;                              // greedy go down to the
left
        while (2 * v + bits[v] < length)        // .
            v = 2 * v + bits[v];                // .
        while (v > 0) {                         // siftDown
            join(v, 0);                 // .
            v /= 2;                             // .
        }
```

```
return min;
} // Deletes minimum

public void build() {
while (!state.equals(State.built))
     buildStep();
} // makes WeakHeap a correct one

public void heapsort() {
while (!state.equals(State.done))
     heapsortStep();
length = initialLength;
}

public boolean heapsortStep() {
if (length < 2) {
     state = State.done;
}
boolean swapped = false;
switch (state) {
     case initial, preBuilding, building -> {
     }
     case built -> {
          state = State.delMin;
     }
     case delMin -> {
          if (length < 2) {   // length == 0 or length == 1
               state = State.done;
               break;
          }
          length--;
          values[0] ^= values[length];
          values[length] ^= values[0];
          values[0] ^= values[length];
          state = State.preSiftDown;
          joinId1 = 1;
          joinId2 = 0;
          while (2 * joinId1 + bits[joinId1] < length)
```

```
                    joinId1 = 2 * joinId1 + bits[joinId1];     //
left Child
                //joinId1*=2;
                if (length == 1) {
                    state = State.done;
                }
            }
            case preSiftDown -> {
                //joinId1 /= 2;
                state = State.siftDown;
            }
            case siftDown -> {
                swapped = join(joinId1, joinId2);
                joinId1 /= 2;
                if (joinId1 > 0)
                    state = State.preSiftDown;
                else
                    state = State.delMin;
            }
            case done -> length = initialLength;
        }
        return swapped;
        }

        public List<Integer> allChildrenOf(int id) {
        ArrayList<Integer> lst = new ArrayList<Integer>(List.of());
        if (id*2 < length && id*2 > 0) {
            lst.add(id*2);
            lst.addAll(allChildrenOf(id*2));
        }
        if (id*2+1 < length) {
            lst.add(id*2+1);
            lst.addAll(allChildrenOf(id*2+1));
        }
        return lst;
        }

        public boolean buildStep() {
```

```
        boolean swapped = false;
        if (length < 2) {
                state = State.built;
        }
        switch (state) {
                case initial -> {
                        state = State.preBuilding;
                }
                case preBuilding -> {
                        if (buildIterator < 1)
                                state = State.built;
                        else {
                                joinId1 = buildIterator;
                                joinId2 = up(buildIterator);
                                state = State.building;
                        }
                }
                case building -> {
                        swapped = join(joinId1, joinId2);
                        buildIterator--;
                        if (buildIterator < 1)
                                state = State.built;
                        else
                                state = State.preBuilding;
                }
                case delMin, siftDown, built -> {}
        }
        return swapped;
        }
}
```

Файл: MainWindow.java

```
package sample;

import javafx.fxml.FXMLLoader;
import javafx.scene.Scene;
```

```
import javafx.scene.layout.AnchorPane;
import javafx.scene.paint.Paint;
import javafx.stage.Stage;

import java.io.File;
import java.net.URL;
import java.util.List;

public final class MainWindow {
      private Stage primaryStage;
      private AnchorPane root;
      private WeakHeap stepHeap;
      private WeakHeap.State prevState;
      private stepState currState;


      public MainWindow(Stage stage) {
      this.primaryStage = stage;
      try{
            FXMLLoader loader = new FXMLLoader();
            URL xmlUrl = Main.class.getResource("/sample.fxml");
            loader.setLocation(xmlUrl);
            root = loader.load();
            Controller appController = loader.getController();
            appController.setMainWindow(this);
      } catch (Exception e) {
            System.exit(1);
      }
      primaryStage.setTitle("Визуализатор сортировки слабой кучей");
      primaryStage.setScene(new Scene(root, 700, 400));
      primaryStage.show();
      }
      public int[] sort(Integer[] data){
      stepHeap = new WeakHeap(data);
      stepHeap.build();
      stepHeap.heapsort();
      return stepHeap.values;
      }
```

```
      public void startStepSort(Integer[] data, AnchorPane
drawField){
      stepHeap = null;
      stepHeap = new WeakHeap(data);
      currState = new stepState();
      WeakHeapRenderer.render(stepHeap, drawField);
      }

      public stepState stepSort(AnchorPane drawField){
      currState.state = stepHeap.state;
      switch (currState.state) {
            case preBuilding, building -> {
                  currState.first = stepHeap.joinId1;
                  currState.second = stepHeap.joinId2;
                  WeakHeapRenderer.render(stepHeap, drawField,
List.of(currState.first, currState.second),
                        currState.isChanged ? Paint.valueOf("ORANGE") :
Paint.valueOf("BLUE"),
                        currState.isChanged ?
stepHeap.allChildrenOf(currState.first) : null,
                        Paint.valueOf("LIGHTBLUE"));
                  currState.isChanged = stepHeap.buildStep(); // made a
step
                  currState.first = stepHeap.joinId1;
                  currState.second = stepHeap.joinId2;
                  prevState = currState.state;
            }
            case built -> {
                  currState.first = stepHeap.joinId1;
                  currState.second = stepHeap.joinId2;
                  WeakHeapRenderer.render(stepHeap, drawField,
                        List.of(currState.first, currState.second),
                        currState.isChanged ? Paint.valueOf("ORANGE") :
Paint.valueOf("BLUE"),
                        currState.isChanged ?
stepHeap.allChildrenOf(currState.first) : null,
                        Paint.valueOf("LIGHTBLUE"));
```

```
                currState.isChanged = stepHeap.heapsortStep(); //
made a step
                prevState = currState.state;
           }
           case delMin -> {
                WeakHeapRenderer.render(stepHeap, drawField,
                     List.of(currState.first, currState.second),
                     currState.isChanged ? Paint.valueOf("PINK") :
null,
                     currState.isChanged ?
stepHeap.allChildrenOf(currState.first) : null,
                     Paint.valueOf("AQUA"));
                currState.first = stepHeap.length - 1;
                currState.second = stepHeap.joinId2;
                currState.isChanged = stepHeap.heapsortStep(); //
made a step
                prevState = currState.state;
           }
           case preSiftDown -> {
                if (prevState == WeakHeap.State.delMin) {
                     WeakHeapRenderer.render(stepHeap, drawField);
                     currState.first = stepHeap.joinId1;
                     currState.second = stepHeap.joinId2;
                } else {
                     //currState.first = stepHeap.joinId1;
                     if (currState.first != currState.first / 2 * 2
+ stepHeap.bits[currState.first / 2]) {
                     currState.first = currState.first / 2 * 2 +
stepHeap.bits[currState.first / 2];
                     }
                     //currState.second = stepHeap.joinId2;
                     WeakHeapRenderer.render(stepHeap, drawField,
                          List.of(currState.first,
currState.second),
                          currState.isChanged ?
Paint.valueOf("PINK") : Paint.valueOf("YELLOW"),
                          currState.isChanged ?
stepHeap.allChildrenOf(currState.first) : null,
```

```
                        Paint.valueOf("AQUA"));
                }
                prevState = currState.state;
                currState.isChanged = stepHeap.heapsortStep();
                currState.first = stepHeap.joinId1;
                currState.second = stepHeap.joinId2;
            }
            case siftDown -> {
                if (prevState == WeakHeap.State.delMin) {
                    WeakHeapRenderer.render(stepHeap, drawField);
                    currState.first = stepHeap.joinId1;
                    currState.second = stepHeap.joinId2;
                } else {
                    currState.first = stepHeap.joinId1;
                    currState.second = stepHeap.joinId2;
                    WeakHeapRenderer.render(stepHeap, drawField,
                        List.of(currState.first,
currState.second),
                        currState.isChanged ?
Paint.valueOf("PINK") : Paint.valueOf("YELLOW"),
                        currState.isChanged ?
stepHeap.allChildrenOf(currState.first) : null,
                        Paint.valueOf("AQUA"));
                    prevState = currState.state;
                    currState.isChanged = stepHeap.heapsortStep();
// made a step
                }
            }
            case done -> {
                WeakHeapRenderer.render(stepHeap, drawField);
            }
        }
        return currState;
        }
        public Stage getPrimaryStage() {
        return primaryStage;
        }
        public String readData(File source){
```

```
        FileIO loadFile = new FileIO(source);
        return loadFile.readFromFile();
        }
        public void writeData(File destination, int[] data){
        FileIO saveFile = new FileIO(destination);
        saveFile.writeToFile(data);
        }
}
```

Файл: Controller.java

```
package sample;

import java.io.File;
import java.net.URL;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.ResourceBundle;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.geometry.HPos;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.input.KeyEvent;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.GridPane;
import javafx.scene.control.Alert.AlertType;
import javafx.scene.layout.Region;
import javafx.scene.paint.Paint;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

public class Controller {
        private MainWindow mainwindow;
        private ArrayList<EditableButton> massButtonElem = null;
        private int countElem = 0;
```

```
private static int MAX_ELEM_IN_ROW = 35;
private int swapState = 0;
private EditableButton first, second;
stepState currState = null;
private Stage prevStepStage;
private PrevStepController prevStepController;


@FXML
private ResourceBundle resources;
@FXML
private URL location;
@FXML
private Label menuLabel;
@FXML
private Button helpButton;
@FXML
private Button saveButton;
@FXML
private Button loadButton;
@FXML
private Button authorsButton;
@FXML
private Button exitButton;
@FXML
private TextField insertDataLine;
@FXML
private Button insertButton;
@FXML
private Button rezultButton;
@FXML
private Button stepButton;
@FXML
private Button addElemButton;
@FXML
private GridPane elemBox;
@FXML
private TextArea informationArea;
```

```
	@FXML
	private AnchorPane drawField;
	@FXML
	private Button runButton;
	@FXML
	private Button prevStepButton;
	@FXML
	private ScrollPane scrPane;
	@FXML
	void initialize() {
	//создание дополнительного окна
	prevStepStage = new Stage();
	AnchorPane root = null;
	try{
		FXMLLoader loader = new FXMLLoader();
		URL xmlUrl =
Main.class.getResource("/prevStepWindow.fxml");
		loader.setLocation(xmlUrl);
		root = loader.load();
		prevStepController = loader.getController();
	} catch (Exception e) {
		System.exit(2);
	}
	prevStepStage.setTitle("Предыдущий шаг");
	prevStepStage.setScene(new Scene(root, 700, 400));

	scrPane.widthProperty().addListener((obs, oldVal, newVal) -> {
		if (massButtonElem != null) {
			if (scrPane.getWidth() > 300) {
				int sum = 0;
				for (EditableButton b : massButtonElem) {
				sum += b.getWidth();
				}
				int ar = sum / massButtonElem.size();
				MAX_ELEM_IN_ROW = (int) (scrPane.getWidth() /
(ar * 1.05));
				elemBox.getChildren().clear();
				if (currState == null) {
```

```
                    elemBox.getChildren().add(addElemButton);
                    }
                    for (int i = 0; i < massButtonElem.size(); i++)
{
                    GridPane.setHalignment(massButtonElem.get(i),
HPos.CENTER);
                    elemBox.getChildren().remove(addElemButton);
                    elemBox.add(massButtonElem.get(i), i %
MAX_ELEM_IN_ROW,i / MAX_ELEM_IN_ROW);
                    if (currState == null) {
                         elemBox.add(addElemButton, (i + 1) %
MAX_ELEM_IN_ROW, (i + 1) / MAX_ELEM_IN_ROW);
                    }
                    }
               }
          }
     });
     }

     @FXML
     private void buttonRunPressed(){
     if(countElem==0){
          new Alert(AlertType.INFORMATION, "Данные не
введены!").showAndWait();
          return;
     }
     Integer[] data = new Integer[massButtonElem.size()];
     for(Button a : massButtonElem ){
          data[massButtonElem.indexOf(a)] =
Integer.valueOf(a.getText());
     }
     elemBox.getChildren().removeAll(addElemButton);
     for (int i = 0; i < countElem; i++) {
          elemBox.getChildren().get(i).setDisable(true);
     }
     mainwindow.startStepSort(data, drawField);
     stepButton.setDisable(false);
     prevStepButton.setDisable(false);
```

```
        loadButton.setDisable(true);
        insertButton.setDisable(true);
        runButton.setDisable(true);
        currState = null;
        buttonStepPressed();
        }
        @FXML
        private void buttonStepPressed() {
        if(currState==null){
              currState = new stepState();
              currState.state = WeakHeap.State.initial;
              currState.first = 0;currState.second = 0;
              currState.length = 0;
        }
        prevStepController.setPrevStepState(massButtonElem,
informationArea.getText(), drawField.getChildren());
        clearStyleButtons();
        first = massButtonElem.get(currState.first);
        second = massButtonElem.get(currState.second);
        switch (currState.state) {
              case preBuilding -> {
                    first.setTextFill(Paint.valueOf("WHITE"));
                    second.setTextFill(Paint.valueOf("WHITE"));
                    first.setStyle("-fx-background-color: blue");
                    second.setStyle("-fx-background-color: blue");
                    informationArea.setText("Этап 1. Построение
кучи\nСравниваем элементы:\n"+ second.getText()+" и " +
first.getText());
              }
              case building-> {
                    if(currState.isChanged){
                          swapEditableButtons(first,
second,"-fx-background-color: orange");
                          first.setTextFill(Paint.valueOf("WHITE"));
                          second.setTextFill(Paint.valueOf("WHITE"));
                          informationArea.setText("Этап 1. Построение
кучи\nМеняем элементы:\n"+ second.getText()+" и " + first.getText()
```

```
                        + "\nМеняем бит у элемента:\n" +
second.getText());
                }
                else{
                      first.setStyle("-fx-background-color: blue");
                      second.setStyle("-fx-background-color: blue");
                      first.setTextFill(Paint.valueOf("WHITE"));
                      second.setTextFill(Paint.valueOf("WHITE"));
                      informationArea.setText("Этап 1. Построение
кучи\nНе меняем элементы:\n"+ second.getText()+" и "+
first.getText());
                }
           }
           case built -> {
                informationArea.setText("Этап 1. Построение
кучи\nКуча построена");
           }
           case preSiftDown -> {
                first.setStyle("-fx-background-color: yellow");
                second.setStyle("-fx-background-color: yellow");
                informationArea.setText("Этап 2.
Сортировка\nСравниваем элементы:\n"+ second.getText()+" и "+
first.getText());
           }
           case siftDown -> {
                if(currState.isChanged) {
                      swapEditableButtons(first, second, "");
                      informationArea.setText("Этап 2.
Сортировка\nМеняем элементы:\n"+ second.getText()+" и " +
first.getText()
                            + "\nМеняем бит у элемента\n" +
second.getText());
                }
                else {
                      informationArea.setText("Этап 2. Сортировка\nНе
меняем элементы:\n"+ second.getText()+" и "+ first.getText());
                }
           }
```

```
            case delMin -> {
                  swapEditableButtons(first, second,
"-fx-background-color: green");
                  informationArea.setText("Этап 2. Сортировка\nУдаляем
корень\nСтавим на место корня\nпоследний элемент: " +
first.getText());
            }
            case done -> {
                  informationArea.setText("Этап 2. Сортировка\nДанные
отсортированы!");
                  setAllButtonsStyle("-fx-background-color: green");
                  saveButton.setDisable(false);
                  insertButton.setDisable(false);
                  loadButton.setDisable(false);
                  stepButton.setDisable(true);
                  prevStepButton.setDisable(true);
                  runButton.setDisable(false);
                  currState = null;
                  countElem = 0;
                  massButtonElem = null;
            }
            case initial -> {

            }
      }
      currState = mainwindow.stepSort(drawField);
      }
      @FXML
      void prevStepButtonPressed() {
      prevStepStage.show();
      }
      @FXML
      private void buttonRezultPressed() {
      if(countElem==0){
            new Alert(AlertType.INFORMATION, "Данные не
введены!").showAndWait();
            return;
      }
```

```
      Integer[] data = new Integer[massButtonElem.size()];
      for(Button a : massButtonElem ){
            data[massButtonElem.indexOf(a)] =
Integer.valueOf(a.getText());
      }
      int[] sortData = mainwindow.sort(data);
      elemBox.getChildren().clear();
      countElem = sortData.length;
      massButtonElem = new ArrayList<EditableButton>();
      for (int i = 0; i < sortData.length; i++) {
            massButtonElem.add(new
EditableButton(Integer.toString(sortData[i])));
            elemBox.add(massButtonElem.get(i), i % MAX_ELEM_IN_ROW, i
/ MAX_ELEM_IN_ROW);
            GridPane.setHalignment(massButtonElem.get(i),
HPos.CENTER);
            massButtonElem.get(i).setDisable(true);
      }
      countElem = 0;
      informationArea.setText("Данные\nотсортированы!");
      setAllButtonsStyle("-fx-background-color: green");
      saveButton.setDisable(false);
      insertButton.setDisable(false);
      loadButton.setDisable(false);
      stepButton.setDisable(true);
      prevStepButton.setDisable(true);
      runButton.setDisable(false);
      drawField.getChildren().clear();
      }
      @FXML
      private void buttonInsertPressed() {
      try {
            stringToButtons(insertDataLine.getText());
      } catch (NumberFormatException nfe) {
            new Alert(AlertType.ERROR, "Введены некорректные
данные.").showAndWait();
      }
      rezultButton.setDisable(false);
```

```
      saveButton.setDisable(false);
      }
      @FXML
      private void buttonSavePressed() {
      if(massButtonElem == null)
            return;
      FileChooser fileChooser = new FileChooser();
      fileChooser.setInitialDirectory(new
File(System.getProperty("user.dir")));
      fileChooser.setTitle("Save data");
      FileChooser.ExtensionFilter extFilter =
                  new FileChooser.ExtensionFilter("TEXT files (*.txt)",
"*.txt");
      fileChooser.getExtensionFilters().add(extFilter);
      File file =
fileChooser.showSaveDialog(mainwindow.getPrimaryStage());
      int[] data = new int[massButtonElem.size()];
      for(Button a : massButtonElem ){
            data[massButtonElem.indexOf(a)] =
Integer.valueOf(a.getText());
      }
      if (file != null) {
            mainwindow.writeData(file, data);
      }
      }
      @FXML
      private void buttonLoadPressed() {
      FileChooser fileChooser = new FileChooser();
      fileChooser.setInitialDirectory(new
File(System.getProperty("user.dir")));
      fileChooser.setTitle("Open file with source data");
      FileChooser.ExtensionFilter extFilter = new
FileChooser.ExtensionFilter("TEXT files (*.txt)", "*.txt");
      fileChooser.getExtensionFilters().add(extFilter);
      File file =
fileChooser.showOpenDialog(mainwindow.getPrimaryStage());
      String data = null;
      if (file != null) {
```

```
            data = mainwindow.readData(file);
      }
      if(data != null){
            try {
                  stringToButtons(data);
            } catch (NumberFormatException nfe) {
                  new Alert(AlertType.ERROR, "Введены некорректные
данные.").showAndWait();
            }
      }
      }
      @FXML
      private void buttonAuthorsPressed() {
      Alert alert = new Alert(AlertType.INFORMATION);
      alert.setTitle("Авторы");
      alert.setHeaderText(null);
      alert.setContentText("""
                  Данное приложение написали студенты ФКТИ ЛЭТИ:
                  Птичкин Сергей: GUI, файловый ввод/вывод
                  Прибылов Никита: визуализация алгоритма,
тестирование, сборка
                  Ноздрин Василий: алгоритм сортировки

                  Преподаватель: Фирсов Михаил Александрович""");
      alert.getDialogPane().setMinWidth(550);
      alert.setResizable(true);
      alert.showAndWait();
      }
      @FXML
      private void buttonClearPressed(){
      massButtonElem = null;
      countElem = 0;
      elemBox.getChildren().clear();
      elemBox.add(addElemButton, 0, 0);
      rezultButton.setDisable(false);
      informationArea.clear();
      insertDataLine.clear();
      saveButton.setDisable(true);
```

```
        loadButton.setDisable(false);
        insertButton.setDisable(false);
        stepButton.setDisable(true);
        prevStepButton.setDisable(true);
        runButton.setDisable(false);
        drawField.getChildren().clear();
        currState = null;
        }

        @FXML
        private void buttonExitPressed() {
        System.exit(0);
        }

        private void addElemToBox(Button newElem){
        GridPane.setHalignment(newElem, HPos.CENTER);
        elemBox.getChildren().remove(addElemButton);
        elemBox.add(newElem, countElem % MAX_ELEM_IN_ROW,countElem /
MAX_ELEM_IN_ROW);
        elemBox.add(addElemButton, (countElem+1) % MAX_ELEM_IN_ROW,
(countElem+1) / MAX_ELEM_IN_ROW);
        }
        @FXML
        private void buttonAddElemPressed() {
        if(massButtonElem == null){
              massButtonElem = new ArrayList<EditableButton>();
        }
        massButtonElem.add(new EditableButton("0"));
        addElemToBox(massButtonElem.get(countElem));
        countElem++;
        saveButton.setDisable(false);
        }
        @FXML
        private void buttonHelpPressed() {
        Alert alert = new Alert(AlertType.INFORMATION);
        alert.setTitle("Справка");
        alert.setHeaderText(null);
        alert.setContentText("""
```

Данное приложение визуализирует алгоритм сортировки слабой кучей.

Ввод данных:

Можно ввести данные через пробел в текстовую линию, затем нажать клавишу "ввод".

Можно добавлять элементы непосредственно в массив нажимая клавишу "+" и меняя значение элемента через контекстное меню кнопки.

Контекстное меню открывается нажатием ПКМ на кнопку элемента. По умолчанию значение элемента - 0.

Также можно удалять элементы и менять местами при нажатии соответствующей опции в меню.

Свап элементов происходит при выборе опции у двух элементов. Первый подсвечивается синим цветом.

Алгоритм:

Запуск алгоритма происходит при нажатии кнопки "Запуск".

Можно получить мгновенный результат, нажав кнопку "Результаты". Результат отобразится в массиве кнопок.

Для пошагового выполнения алгоритма можно использовать кнопку "Шаг".\s

Также есть возможность посмотреть на состояние выполнения алгоритма на предыдущем шаге, нажав кнопку "Предыдущий шаг".

На каждом шаге в текстовом поле справа описывается, что происходит на текущем шаге.

Визуализация алгоритма:

Текущее состояние массива данных отражается в панели с кнопками, на которых отражается значение каждого элемента.

При выполнении алгоритма кнопки могут поменять свой порядок расположения, что отражает изменения в массиве данных.

В нижней части окна рисуется текущее состояние бинарного дерева кучи.\s

Под вершинами также указывается индекс элемента и значение бита.

При сравнении, перемещении и удалении элементов они подсвечиваются специальным цветом.

Отсортированная часть массива принимает зелёный цвет, из кучи же они удаляются.\s

Сохранение:

При нажатии кнопки "Сохранить" будет выведено окно предлагающее сохранить данные в файл.

Данные, которые записываются в файл - текущее состояние массива данных.

Загрузка:

При нажатии кнопки "Загрузка" будет выведено окно, предлагающее считать данные из файла с расширением ".txt".

В файле должны быть данные, представленные символами чисел, разделённых исключительно пробелами.

Если будет введён отличный от пробела и цифр символ, данные не будут считаны.""");

```
alert.setResizable(true);
alert.getDialogPane().setMinWidth(1000);
alert.showAndWait();
}

public void setMainWindow(MainWindow mw){
mainwindow = mw;
}

class EditableButton extends Button {
private TextField tf;
final ContextMenu contextMenu = new ContextMenu();

public EditableButton(String text) {
      setText(text);
      tf = new TextField();
      MenuItem change = new MenuItem("Изменить");
      MenuItem swap = new MenuItem("Обмен");
      MenuItem delete = new MenuItem("Удалить");
      contextMenu.getItems().addAll(change, swap, delete);
```

```
            this.setContextMenu(contextMenu);

            // настройка контекстного меню
            change.setOnAction(e -> {
                  tf.setText(getText());
                  setGraphic(tf);
                  setText("");
                  tf.requestFocus();
                  tf.selectAll();
            });

            swap.setOnAction(e -> {
                  if (swapState == 0) {
                        first = this;
                        first.setStyle("-fx-background-color: blue");
                  } else {
                        second = this;
                        if (first == second) {
                        swapState = 0;
                        } else {
                        swapEditableButtons(first, second, "");
                        }
                  }
                  swapState = ++swapState % 2;
            });

            delete.setOnAction(e -> {
                  massButtonElem.remove(this);
                  elemBox.getChildren().remove(0,
elemBox.getChildren().size());
                  for (int i = 0; i < massButtonElem.size(); i++) {
                        elemBox.add(massButtonElem.get(i), i %
MAX_ELEM_IN_ROW, i / MAX_ELEM_IN_ROW);
                        GridPane.setHalignment(massButtonElem.get(i),
HPos.CENTER);
                  }
                  elemBox.add(addElemButton, countElem %
MAX_ELEM_IN_ROW, countElem / MAX_ELEM_IN_ROW);
```

```
                countElem--;
            });

            // настройка текстового поля
            tf.textProperty().addListener((obs) ->
                        tf.setPrefWidth(0.5 * tf.getFont().getSize() *
tf.getText().length() + 30));

            tf.addEventFilter(KeyEvent.KEY_TYPED, keyEvent -> {
                if (!"-0123456789".contains(keyEvent.getCharacter()))
{
                    keyEvent.consume();
                }
            });

            tf.focusedProperty().addListener((obs, oldVal, newVal) ->
{
                if (newVal) return;
                if (isStringCorrect(tf.getText())) {
                    setText(tf.getText());
                    setGraphic(null);
                } else {
                    new Alert(AlertType.ERROR, "Вы ввели
некорректные данные.").showAndWait();
                    tf.requestFocus();
                    tf.selectAll();
                }
            });
        }
        }

        private boolean isStringCorrect(String str) {
        boolean result = true;
        try {
            int test =
Integer.parseInt(str.trim().replaceAll("(\\s)+", " "));
        } catch (NumberFormatException nfe) {
            result = false;
```

```
    }
    return result;
    }

    private int[] stringToIntArray(String str) throws
NumberFormatException {
    int[] array = Arrays.stream(str.trim().replaceAll("(\\s)+", "
").split(" ")).mapToInt(Integer::parseInt).toArray();
    return array;
    }

    private void stringToButtons(String str) throws
NumberFormatException {
    int[] numbers = stringToIntArray(str);
    countElem = numbers.length;
    massButtonElem = new ArrayList<EditableButton>();
    elemBox.getChildren().remove(0, elemBox.getChildren().size());
    for (int i = 0; i < numbers.length; i++) {
        massButtonElem.add(new
EditableButton(Integer.toString(numbers[i])));
        elemBox.add(massButtonElem.get(i), i % MAX_ELEM_IN_ROW, i
/ MAX_ELEM_IN_ROW);
        GridPane.setHalignment(massButtonElem.get(i),
HPos.CENTER);
    }
    elemBox.add(addElemButton, countElem % MAX_ELEM_IN_ROW,
countElem / MAX_ELEM_IN_ROW);
    }

    private void swapEditableButtons(EditableButton first,
EditableButton second, String color) {
    int firstRow = GridPane.getRowIndex(first);
    int firstCol = GridPane.getColumnIndex(first);
    int secondRow = GridPane.getRowIndex(second);
    int secondCol = GridPane.getColumnIndex(second);
    Collections.swap(massButtonElem, massButtonElem.indexOf(first),
massButtonElem.indexOf(second));
    elemBox.getChildren().removeAll(first, second);
```

```
        elemBox.add(first, secondCol, secondRow);
        elemBox.add(second, firstCol, firstRow);
        if(color.equals("-fx-background-color: green")) {
              second.setStyle(color);
              second.setTextFill(Paint.valueOf("WHITE"));
        }
        else {
              first.setStyle(color);
              second.setStyle(color);
        }
        }

        private void clearStyleButtons(){
        for(EditableButton a: massButtonElem){
              if(!a.getStyle().equals("-fx-background-color: green")) {
                    a.setStyle("");
                    a.setTextFill(Paint.valueOf("BLACK"));
              }
        }
        }

        private void setAllButtonsStyle(String style){
        for(EditableButton a: massButtonElem){
              a.setStyle(style);
              a.setTextFill(Paint.valueOf("WHITE"));
        }
        }

}
```

Файл: FileIO.java

```
package sample;

import javafx.scene.control.Alert;

import java.io.*;
```

```
import java.nio.file.Files;
import java.nio.file.Paths;

public class FileIO {
	private File resource;
	public FileIO(File resource){
	this.resource = resource;
	}
	public void writeToFile(int[] data){
	try (FileWriter dataWriter = new FileWriter(resource, true);){
		for(int i = 0; i< data.length; i++){
			dataWriter.write(String.valueOf(data[i]+" "));
		}
		dataWriter.write("\n");
	}
	catch (IOException e){
		new Alert(Alert.AlertType.ERROR, "Ошибка при записи в
файл!").showAndWait();
	}
	}
	public String readFromFile(){
	String data = null;
	try {
		data = new
String(Files.readAllBytes(Paths.get(resource.getPath())));
	}
	catch (IOException e){
		new Alert(Alert.AlertType.ERROR, "Ошибка при считывании из
файла!").showAndWait();
	}
	return data;
	}
}
```

Файл: WeakHeapRenderer.java

```
package sample;

import javafx.scene.layout.AnchorPane;
import javafx.scene.paint.Paint;
import javafx.scene.shape.Circle;
import javafx.scene.shape.Line;
import javafx.scene.text.Font;
import javafx.scene.text.Text;

import java.util.*;

public class WeakHeapRenderer {
     private static final double radius = 20;
     private static final double topLeftX = 3*radius, topLeftY =
3*radius;
     private static final Paint defaultColor =
Paint.valueOf("IVORY");

     public static void render(WeakHeap wh, AnchorPane drawField,
                               List<Integer> actionNodes, Paint
actionColor,
                               List<Integer> childrenOfActionNodes, Paint
actionColorForChildren) {
     if (drawField == null) return;
     if (wh.length < 1) {
          drawField.getChildren().clear();
          return;
     }
     if (actionNodes == null) actionNodes = List.of();
     if (actionColor == null) actionColor = defaultColor;
     if (childrenOfActionNodes == null) childrenOfActionNodes =
List.of();
     if (actionColorForChildren == null) actionColorForChildren =
defaultColor;

     drawField.getChildren().clear();
     Line line;
     Text text;
```

```
      Circle circle;

      // draw root
      circle = new Circle(topLeftX, topLeftY, radius,
actionNodes.contains(0) ? actionColor : defaultColor);
      drawField.getChildren().add(circle);

      text = new Text(topLeftX, topLeftY+3,
Integer.toString(wh.values[0]));
      text.setX(text.getX() - text.getLayoutBounds().getWidth()/2);
      drawField.getChildren().add(text);

      text = new Text(topLeftX, topLeftY+3 + radius*1.5, 0 + " (" +
wh.bits[0] + ")");
      text.setX(text.getX() - text.getLayoutBounds().getWidth()/2);
      text.setFont(Font.font(10));
      drawField.getChildren().add(text);

      // prep for drawing the rest
      LinkedList<Integer> row = new LinkedList<>();
      LinkedList<Integer> new_row = new LinkedList<>();
      int length = wh.length;
      int height = 1 + (int) (Math.log(length) / Math.log(2));
      double curX;
      double curY = topLeftY;
      double cellWidth = 2*radius * length;
      double cellHeight = 4*radius;

      row.addLast(1);
      for(int level = 1; level <= height; level++) { // add each
layer
            curY += cellHeight;
            curX = topLeftX-cellWidth/2;
            while (!(row.isEmpty())) {
                  Integer node = row.pollFirst();
                  curX += cellWidth;
                  if (node == null || !(node < length)) {
                        new_row.addLast(null);
```

```
                    new_row.addLast(null);
                } else {
                    // draw line from child to parent
                    line = new Line(curX, curY, (wh.bits[node/2] +
node%2) % 2 == 0 ? curX+cellWidth/2 : curX-cellWidth/2,
curY-cellHeight);
                    drawField.getChildren().add(line);
                    line.toBack();

                    // draw child
                    drawField.getChildren().add(new Circle(curX,
curY, radius,
                            actionNodes.contains(node) ? actionColor :

childrenOfActionNodes.contains(node) ? actionColorForChildren :
defaultColor));

                    // draw value
                    text = new Text(curX,curY+3,
Integer.toString(wh.values[node]));
                    text.setX(text.getX() -
text.getLayoutBounds().getWidth()/2);
                    drawField.getChildren().add(text);

                    // draw index + bit
                    text = new Text(curX, curY+3 + radius*1.5, node
+ " (" + wh.bits[node] + ")");
                    text.setX(text.getX() -
text.getLayoutBounds().getWidth()/2);
                    text.setFont(Font.font(10));
                    drawField.getChildren().add(text);

                    // prep next children
                    new_row.addLast(2 * node + wh.bits[node]); //
add Left child
                    new_row.addLast(2 * node + 1 - wh.bits[node]);
// add right child
                }
```

```
            }
            row = new_row;
            new_row = new LinkedList<>();
            cellWidth /= 2;
      }

      }

      public static void render(WeakHeap wh, AnchorPane drawField,
List<Integer> actionNodes, Paint actionColor) {
      render(wh, drawField, actionNodes, actionColor, List.of(),
defaultColor);
      }

      public static void render(WeakHeap wh, AnchorPane drawField) {
      render(wh, drawField, List.of(), defaultColor, List.of(),
defaultColor);
      }

}



package sample;

import javafx.scene.control.Alert;

import java.io.*;
import java.nio.file.Files;
import java.nio.file.Paths;

public class FileIO {
      private File resource;
      public FileIO(File resource){
      this.resource = resource;
      }
      public void writeToFile(int[] data){
      try (FileWriter dataWriter = new FileWriter(resource, true);){
            for(int i = 0; i< data.length; i++){
```

```
                dataWriter.write(String.valueOf(data[i]+" "));
            }
            dataWriter.write("\n");
        }
        catch (IOException e){
            new Alert(Alert.AlertType.ERROR, "Ошибка при записи в
файл!").showAndWait();
        }
        }
        public String readFromFile(){
        String data = null;
        try {
            data = new
String(Files.readAllBytes(Paths.get(resource.getPath())));
        }
        catch (IOException e){
            new Alert(Alert.AlertType.ERROR, "Ошибка при считывании из
файла!").showAndWait();
        }
        return data;
        }
}
```

Файл: stepState.java

```
package sample;

public class stepState {
        public WeakHeap.State state;
        public Integer first;
        public Integer second;
        public boolean isChanged;
        public int length;
}
```

Файл: PrevStateController.java

```
package sample;

import java.net.URL;
import java.util.ArrayList;
import java.util.ResourceBundle;

import javafx.collections.ObservableList;
import javafx.fxml.FXML;
import javafx.geometry.HPos;
import javafx.scene.Node;
import javafx.scene.control.Button;
import javafx.scene.control.ScrollPane;
import javafx.scene.control.TextArea;
import javafx.scene.layout.AnchorPane;
import javafx.scene.layout.GridPane;

public class PrevStepController {
      private ArrayList<Button> prevMassButtonElem = null;
      private static int MAX_ELEM_IN_ROW = 35;

      @FXML
      private ResourceBundle resources;

      @FXML
      private URL location;

      @FXML
      private AnchorPane prevDrawField;

      @FXML
      private TextArea prevInformationArea;

      @FXML
      private ScrollPane prevScrollPane;

      @FXML
```

```
    private GridPane prevElemBox;

    @FXML
    void initialize() {
    prevScrollPane.widthProperty().addListener((obs, oldVal,
newVal) -> {
          if (prevMassButtonElem != null) {
               if (prevScrollPane.getWidth() > 300) {
                    int sum = 0;
                    for (Button b : prevMassButtonElem) {
                    sum += b.getWidth();
                    }
                    int ar = sum / prevMassButtonElem.size();
                    MAX_ELEM_IN_ROW = (int)
(prevScrollPane.getWidth() / (ar * 1.05));
                    prevElemBox.getChildren().clear();
                    for (int i = 0; i < prevMassButtonElem.size();
i++) {

GridPane.setHalignment(prevMassButtonElem.get(i), HPos.CENTER);
                    prevElemBox.add(prevMassButtonElem.get(i), i %
MAX_ELEM_IN_ROW,i / MAX_ELEM_IN_ROW);
                    }
               }
          }
    });
    }

    public void
setPrevStepState(ArrayList<Controller.EditableButton> massButtonElem,
String textAreaMessage, ObservableList<Node> list){
    prevInformationArea.setText(textAreaMessage);
    prevMassButtonElem = new ArrayList<>();
    prevElemBox.getChildren().clear();
    Button nextButton;
    if(massButtonElem!= null) {
          for (int i = 0; i < massButtonElem.size(); i++) {
               nextButton = new Button();
```

```
                nextButton.setText(massButtonElem.get(i).getText());

nextButton.setStyle(massButtonElem.get(i).getStyle());

nextButton.setTextFill(massButtonElem.get(i).getTextFill());
                prevMassButtonElem.add(nextButton);
                prevElemBox.add(nextButton, i % MAX_ELEM_IN_ROW, i /
MAX_ELEM_IN_ROW);
                GridPane.setHalignment(nextButton, HPos.CENTER);
            }
        }
        prevDrawField.getChildren().clear();
        prevDrawField.getChildren().addAll(list);
        }
}
```