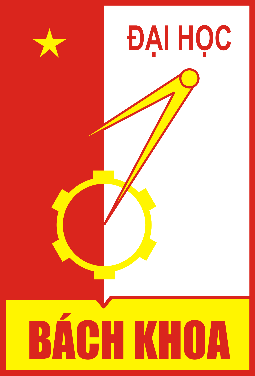
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY



OBJECT ORIENTED PROGRAMMING MINI PROJECT REPORT

# SORTING ALGORITHM VISUALIZATION

**Instructor: Nguyen Thi Thu Trang**

|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** | **Roles** | **Student ID** | **Tasks** |
| Nguyen Khanh Linh | MEMBER | 20194789 | GUI, Report, Help menu, Review |
| Hoang Dai Long | MEMBER | 20194792 | GUI, Shell sort, Slides, Review |
| Le Minh Hieu | MEMBER | 20184257 | GUI, Bubble sort, Heap sort, Menu, Diagram, Demo video, Review |

*Hanoi, 2022*

Contents

[SORTING ALGORITHM VISUALIZATION 1](#_Toc108271394)

[I. Introduction about application 3](#_Toc108271395)

[II. Use case Diagram 3](#_Toc108271396)

[III. Package Diagram and Class Diagram 4](#_Toc108271397)

[The package diagram: 4](#_Toc108271398)

[The class diagrams: 5](#_Toc108271399)

[Analysis of the classes: 8](#_Toc108271400)

[1. Package “application” 8](#_Toc108271401)

[2. Package “console” 8](#_Toc108271402)

[3. Package “controller” 8](#_Toc108271403)

[4. Package “exception” 9](#_Toc108271404)

[5. Package “model” 9](#_Toc108271405)

[6. Package “view” 10](#_Toc108271406)

[IV. Run the program 10](#_Toc108271407)

[How to run the executable file from command lines: 10](#_Toc108271408)

[V. Analysis on each sorting algorithm 20](#_Toc108271409)

[1. Bubble Sort 20](#_Toc108271410)

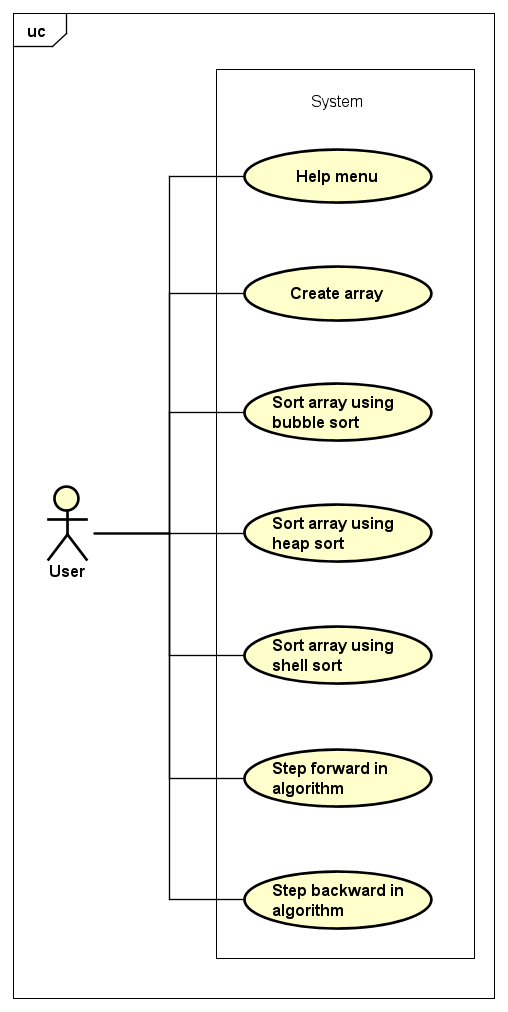
[2. Heap Sort 20](#_Toc108271411)

[3. Shell Sort 20](#_Toc108271412)

# Introduction about application

The application demonstrates about three basic sorting algorithms: Bubble sort, Heap sort, Shell sort. These algorithms solve the problems of reordering items (that can be compared, e.g, integers, floating point numbers, string, etc) of an array (or a list) in a certain order (increasing, non-increasing, decreasing, non-decreasing, lexicographical, etc). This application will focus on integer number of an array, therefore, user can understand deeply via demonstration step by step on how the sorting algorithm works.

# Use case Diagram



Users can make use of the sorting visualizations or see help menu. Moreover, they can create the array by entering the size of an array and the values of integer numbers. During the process of using application, they can have three choices to sort an array: sorting array using bubble sort, sorting array using heap sort, sorting array using shell sort. To observe the state of an array during sorting, users can do step forward in algorithm to see the previous state of an array and do step backward in algorithm to see the next state of an array.

# Package Diagram and Class Diagram

## The package diagram:

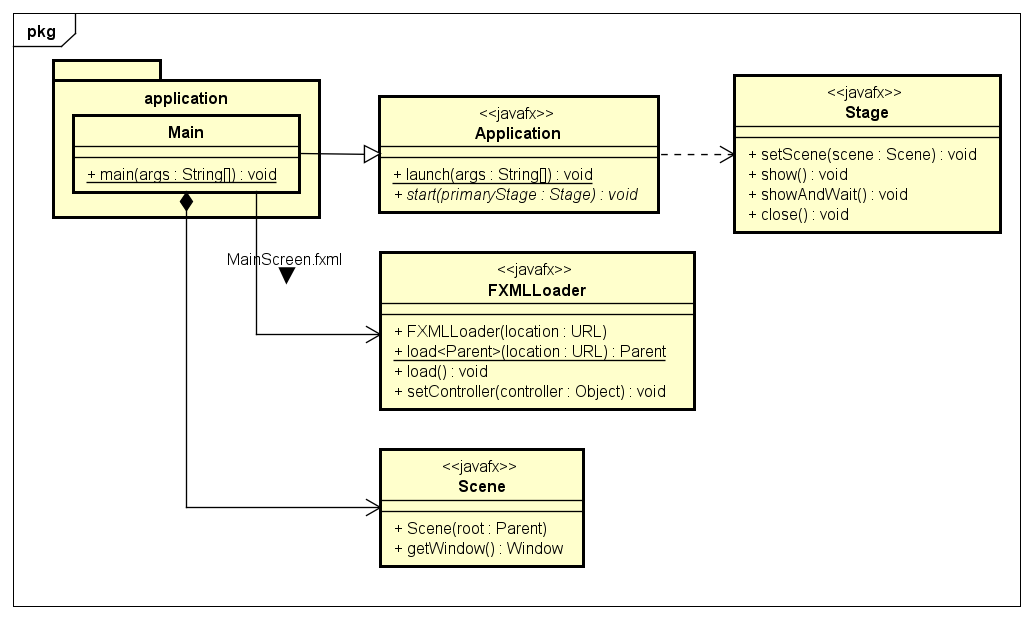
Diagram, schematic

Description automatically generated

The program attempts to follow the Model-View-Controller architecture, consisting of 6 **packages** (in sourcecode, 1 exists for console program which is not reflected in the diagrams):

* 1. The package “**application**”: contains **Main** class for GUI program
  2. The package “**console**”: contains **Main** class for console program
  3. The package “**controller**”: contains classes that directly interact with JavaFX GUI, providing functionality to the GUI’s components
  4. The package “**exception**”: contains two exceptions used in the program
  5. The package “**model**”: contains classes that represent data that can be manipulated, and sort algorithms used on the data
  6. The package “**view**”: contains classes that visualize each of the respective models by rendering them to the JavaFX GUI.

## The class diagrams:



**Application.png**: An overview of a typical JavaFX application.

Diagram

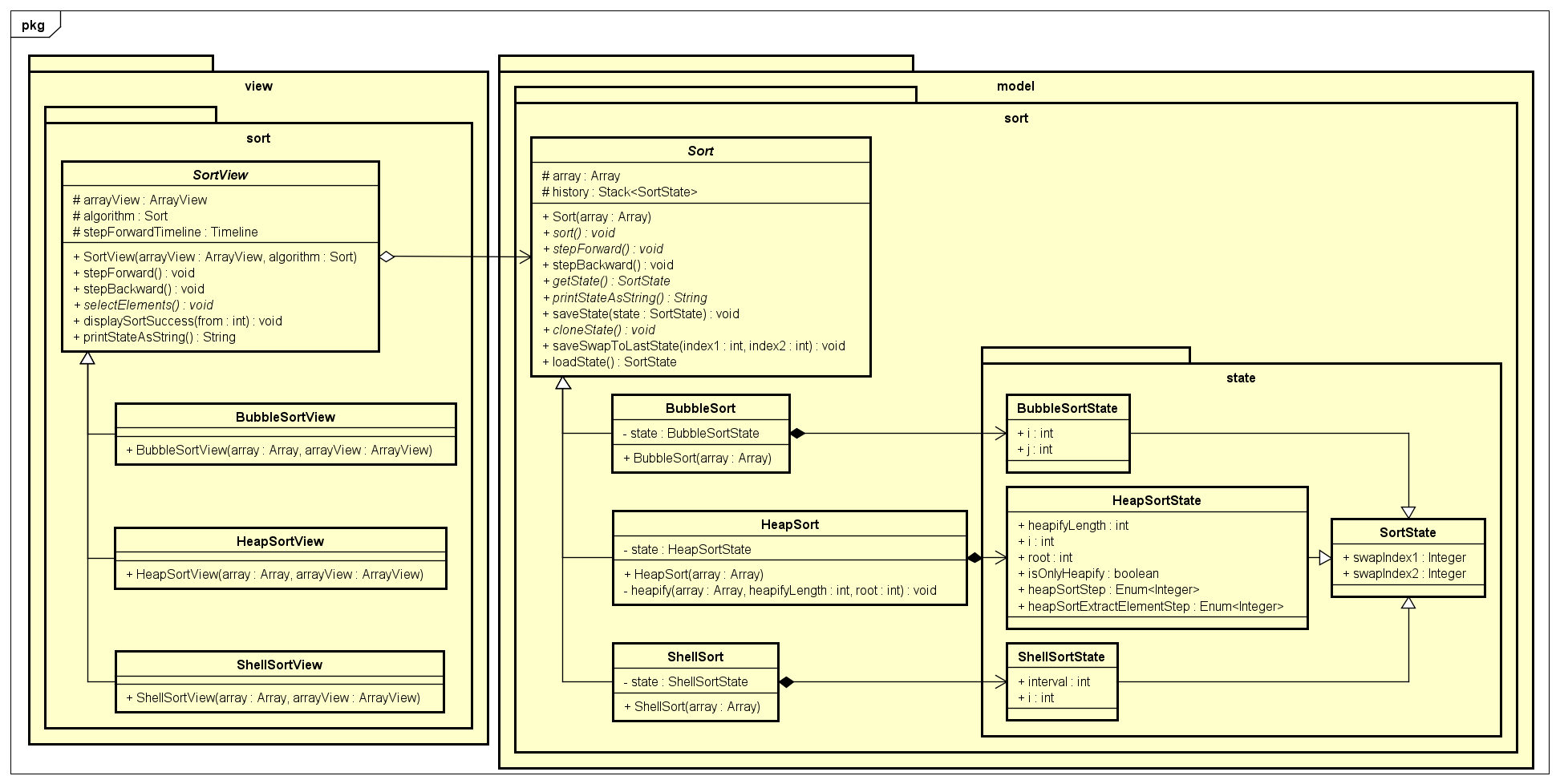
Description automatically generated

**GUI.png:** Shows how most of the visualization classes are related to each other. Sort visualizations are represented more clearly in a later diagram.

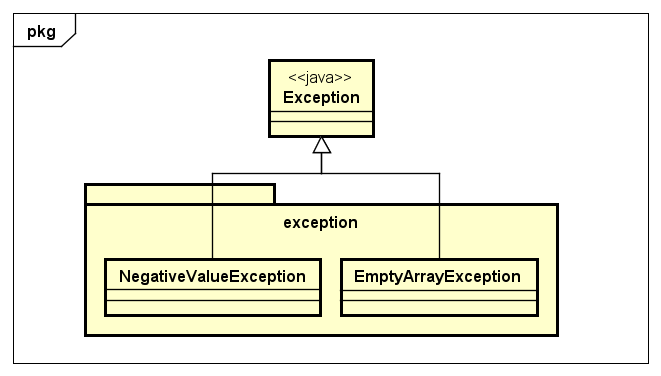
Diagram

Description automatically generated

**Model.png:** Shows the models of the program: An Array of Elements that can be manipulated by Sort. Sort has internal states for saving a sequence of sorting actions.



**Sort.png:** How each of the three sorting algorithms and their visualizers are designed. Each view will use the corresponding sorting algorithm, and each of the algorithms have their own corresponding states for going through its steps.



**Exceptions.png:** Basic representation of our self-defined exceptions.

## Analysis of the classes:

|  |  |
| --- | --- |
| Package “application” | |
| Class | Role |
| Main | Run program with GUI |

|  |  |
| --- | --- |
| Package “console” | |
| Class | Role |
| Main | Run program with console |

|  |  |
| --- | --- |
| Package “controller” | |
| Sub-package “controller.dialog”(Contains the blueprints and controllers for dialogs) | |
| Class/File | Role |
| CreateArrayDialogController | Defines an interface for SortScreenController to make dialogs. The anonymous classes implementing this interface should create a new Array, then display it with ArrayView. |
| CreateInputArrayDialog.fxml | Blueprint for a dialog to create an array with user input. |
| CreateRandomArrayDialog.fxml | Blueprint for a dialog to create an array with random values (and user-inputted length). |
| HelpMenu.fxml | Blueprint for a pop-up window that explains the basics of the program. |
| Sub-package “controller.scene”(Contains the blueprints and controllers for screens) | |
| Class/File | Role |
| MainScreen.fxml | Blueprint for main menu |
| MainScreenController | Control main menu: Start sorting, Quit |
| SortScreen.fxml | Blueprint for sort screen |
| SortScreenController | Control sort screen: Show dialogs for create array, buttons to choose sorting algorithm, buttons to run/step forward/step backward in sorting algorithm, show help menu |

|  |  |
| --- | --- |
| Package “exception” | |
| Class | Role |
| NegativeValueException | Thrown if the value entered is negative |
| EmptyArrayException | Thrown if the Array has no Elements |

|  |  |
| --- | --- |
| Package “model” | |
| **Sub-package “model.array”** | |
| Class | Role |
| Array | An Array of Elements. Other classes can only access its length, get/set the value of an Element, swap two Elements, or check if the array is sorted |
| Sub-package “model.element” | |
| Class | Role |
| Element | A wrapper of an integer, and its index (the Element’s position in an array) |
| Sub-package “model.sort” | |
| Class | Role |
| Sort | A base class for sorting algorithms, keeping a stack of past states, methods to step forward/backward in an algorithm as well as manipulate state. |
| BubbleSort | A subclass of Sort using the bubble sort algorithm. Keeps a reference to its current state. |
| HeapSort | A subclass of Sort using the heap sort algorithm. Keeps a reference to its current state. |
| ShellSort | A subclass of Sort using the shell sort algorithm. Keeps a reference to its current state. |
| Sub-package “model.sort.state” | |
| Class | Role |
| SortState | A base class representing a sorting algorithm’s state, specifically the indices of two elements that have been swapped. |
| BubbleSortState | A SortState with the necessary attributes for bubble sort. |
| HeapSortState | A SortState with the necessary attributes for heap sort. |
| ShellSortState | A SortState with the necessary attributes for shell sort. |

|  |  |
| --- | --- |
| Package “view” | |
| Sub-package “view.array” | |
| Class | Role |
| ArrayView | A visualizer of an Array using a JavaFX HBox. Contains methods to create/show array, handling user button presses, and show alerts. |
| Sub-package “view.element” | |
| Class | Role |
| ElementView | A visualizer of an Element using an HBox. Contains a method to change its background color. |
| Sub-package “view.sort” | |
| Class | Role |
| SortView | An intermediate visualizer class to show what the algorithm will do before a step forward. |
| BubbleSortView | A SortView that highlights the two elements that are being compared. |
| HeapSortView | A SortView that highlights the two elements that will be swapped,or three elements (a node and its two children). |
| ShellSortView | A SortView that highlights the elements that are being compared. |

# Run the program

## How to run the executable file from command lines:

**Step 1**: Make sure you already have the **javafx library** installed in your system. The program can not run without it.

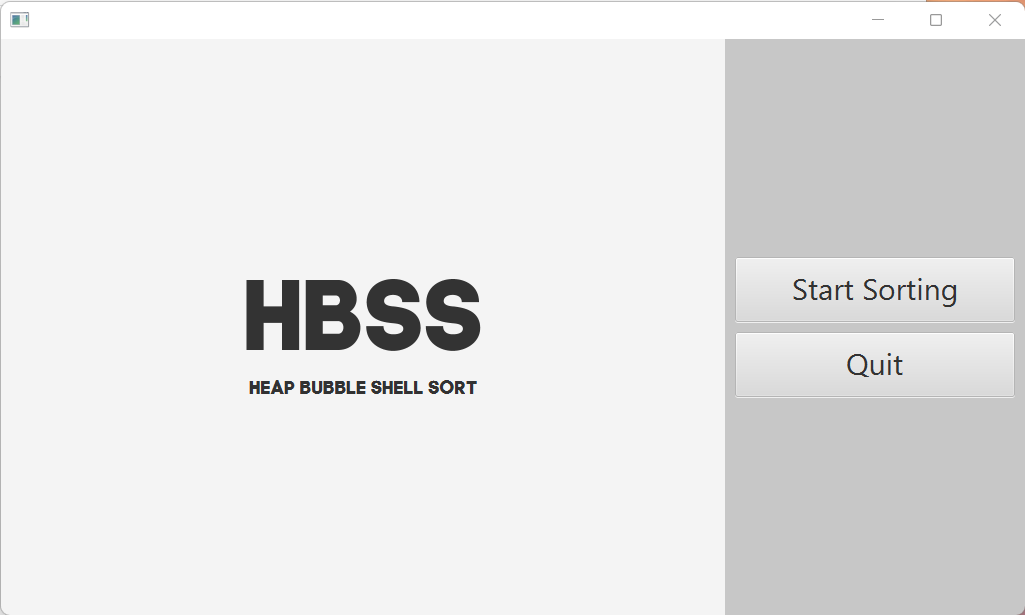
**Step 2**: After extracting the zip file, access the .jar file in its folder, The relative path from the archive to the **.jar** file is “OOLT.ICT.20212.4”

Right-click and choose the option “**Open with cmd**”. **Step 3**: The command format is as follow:

**java *[VM Arguments]* -jar OOLT.ICT.20212.4.jar [VM Arguments] format**:

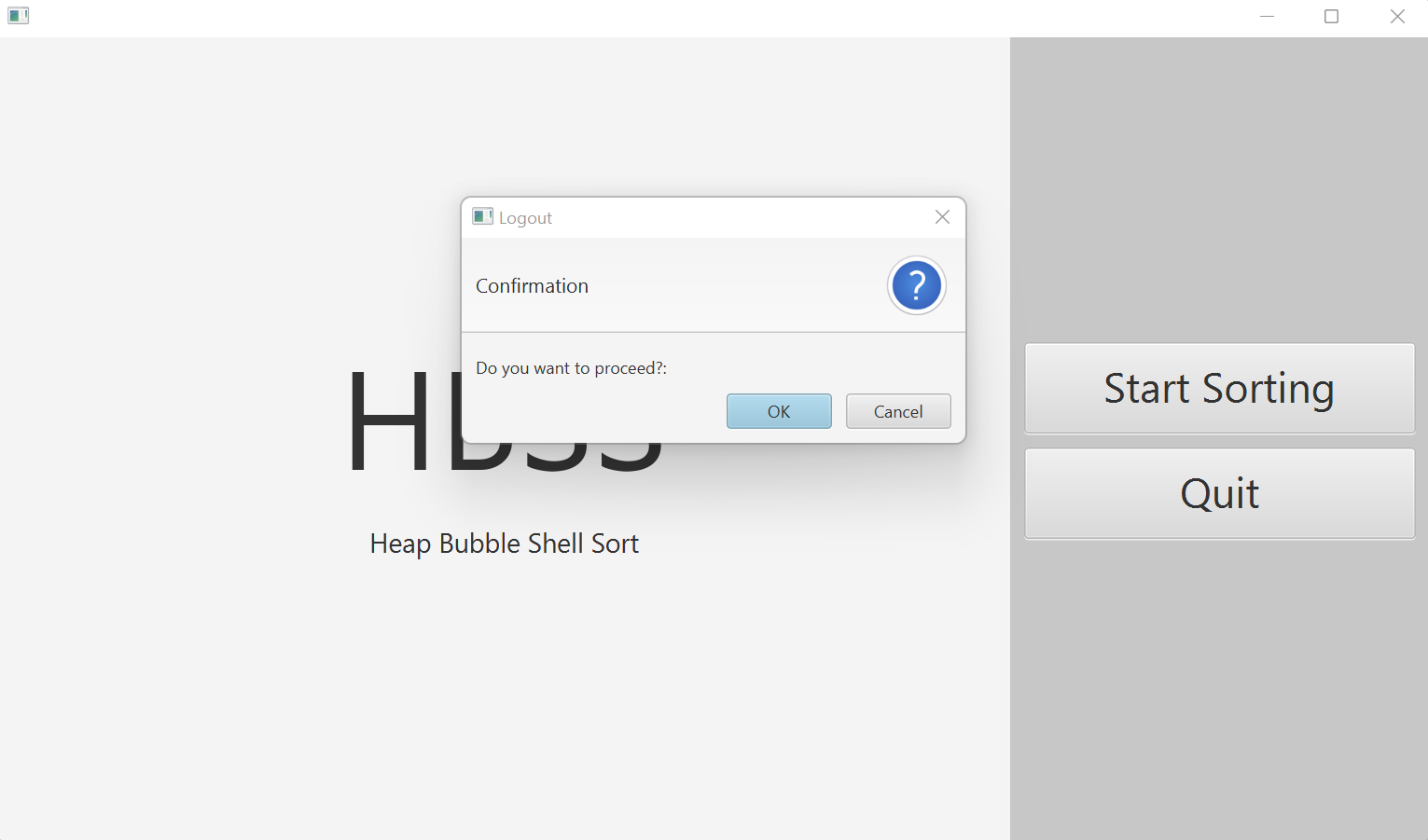
--module-path "*javafx lib file path*" --add- modules=javafx.controls,javafx.fxml

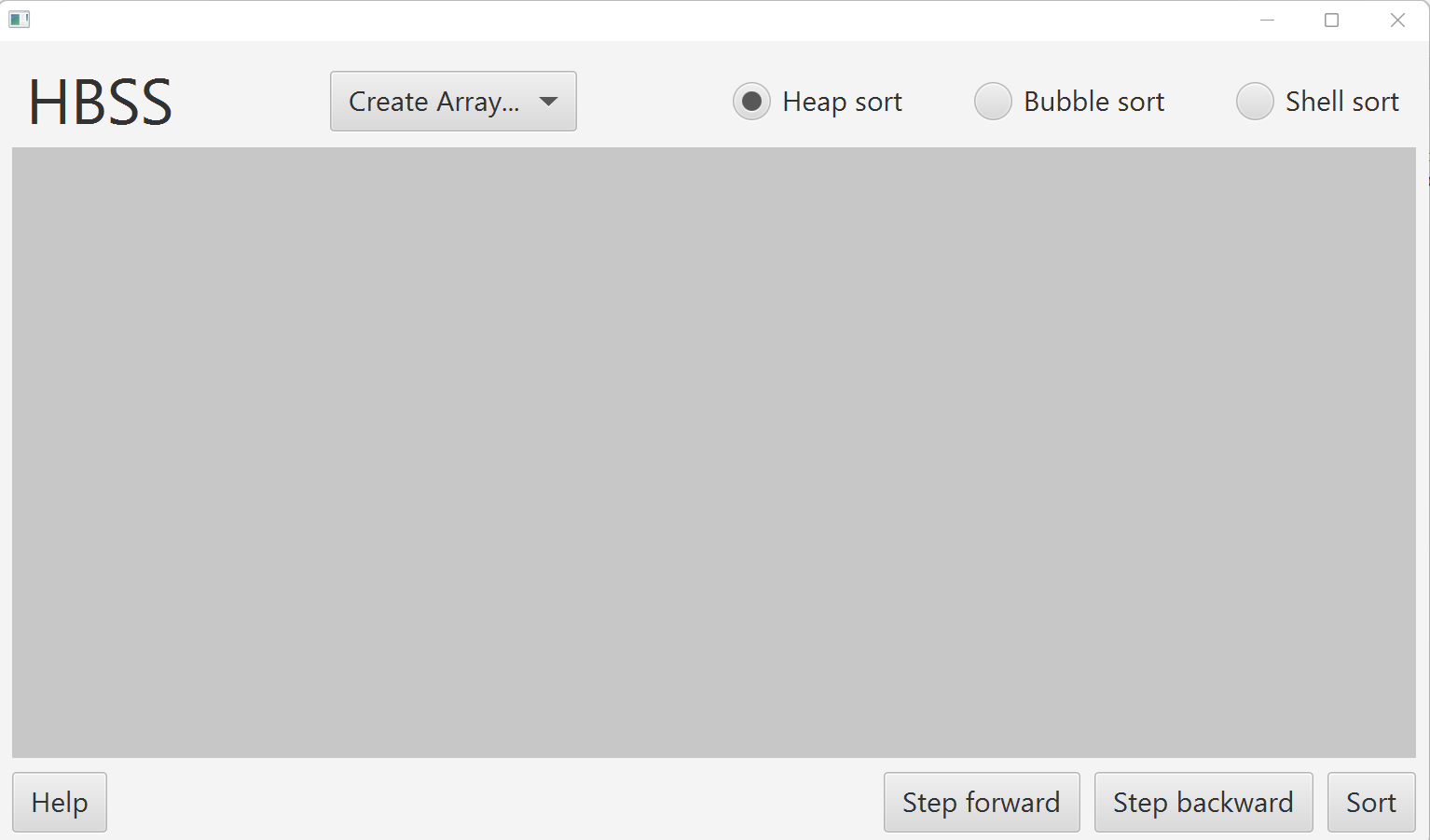
When the program starts running, a main **menu window** will pop up



The user can choose **Start Sorting** to come to Sorting algorithms or **Quit** to exit the program .

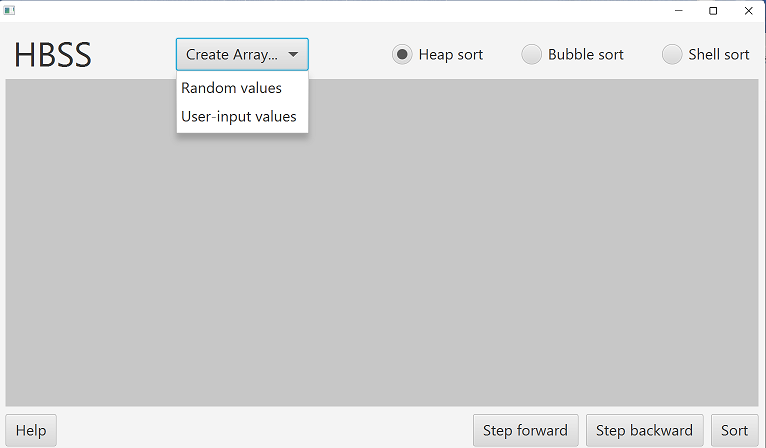
If user chooses **Quit** , the system will pop out the dialog to ensure that user does want to exit.



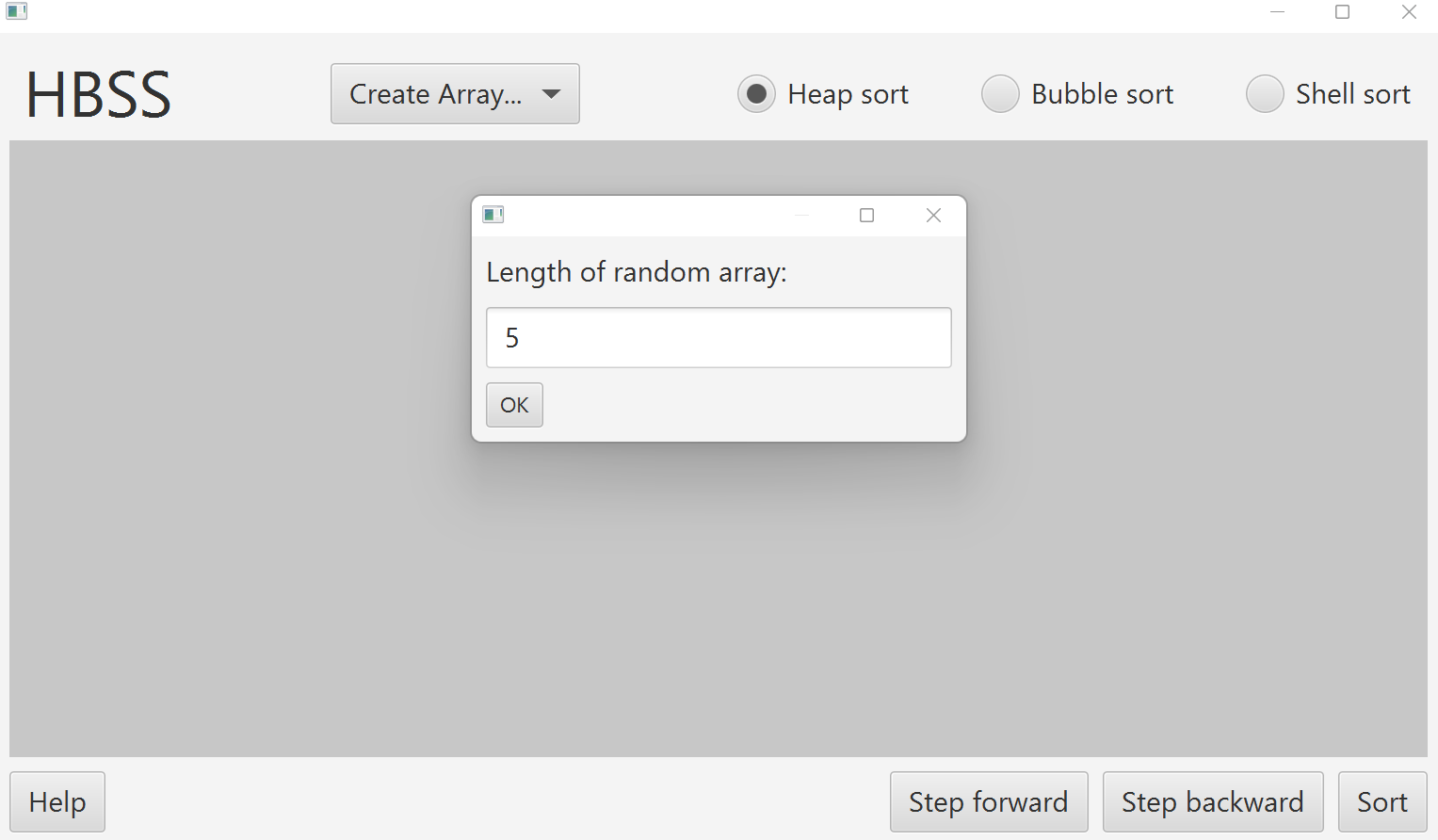
If user choose Start Sorting, the system will switch to a new scene

In the north of scene, there are menubar **Create Array**, Checkboxes: **Heap sort**, **Bubble sort**, **Shell sort**. In the south of scene, there are buttons **Help**, **Step forward**, **Step backward**, **Sort.**

When user click “**Create Array**”, there are two choices for users : **Random values** and **User-input Values**

****

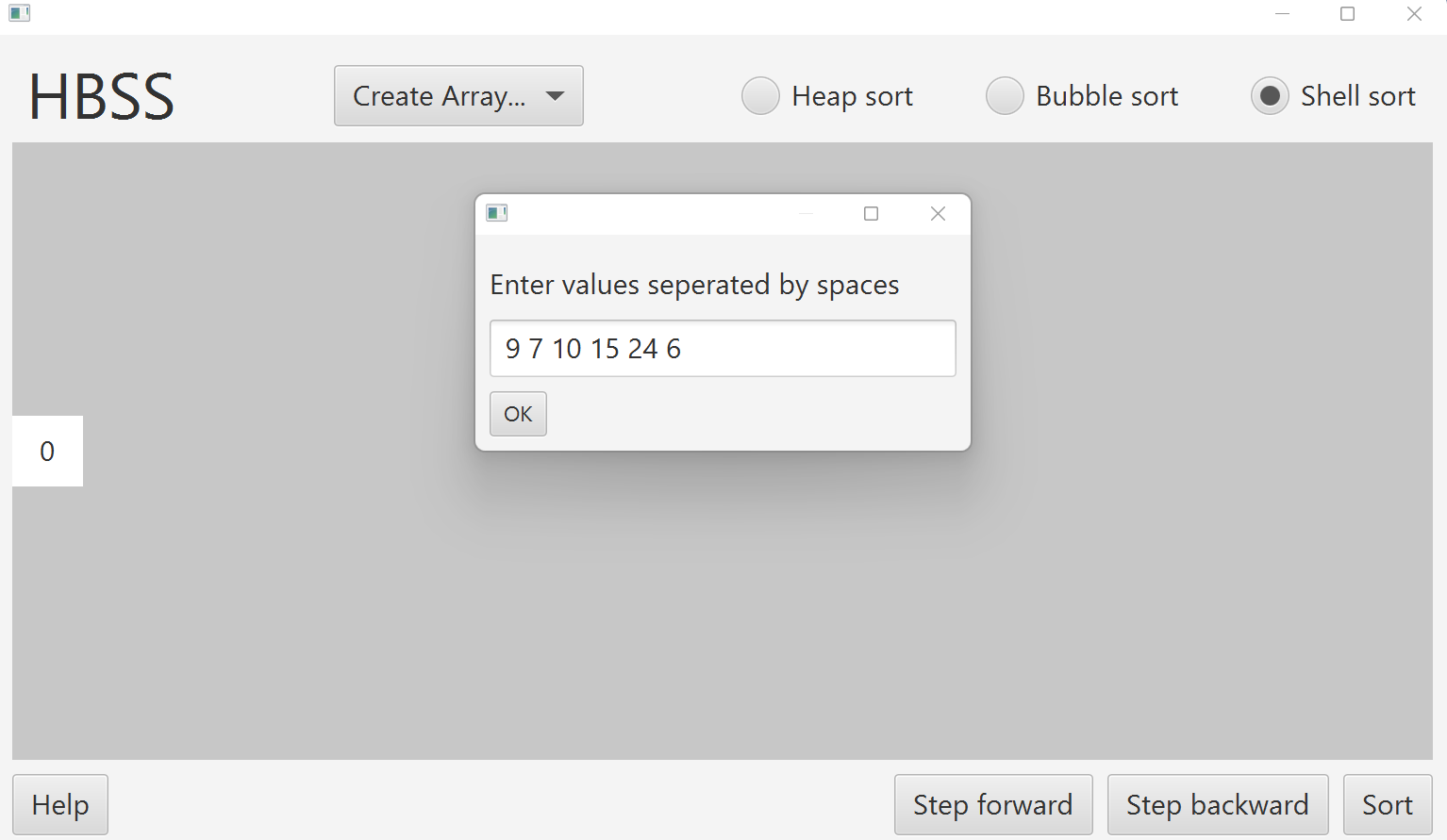
When user chooses **“Random values”** , the screen will appear a dialog for user to enter the length of array



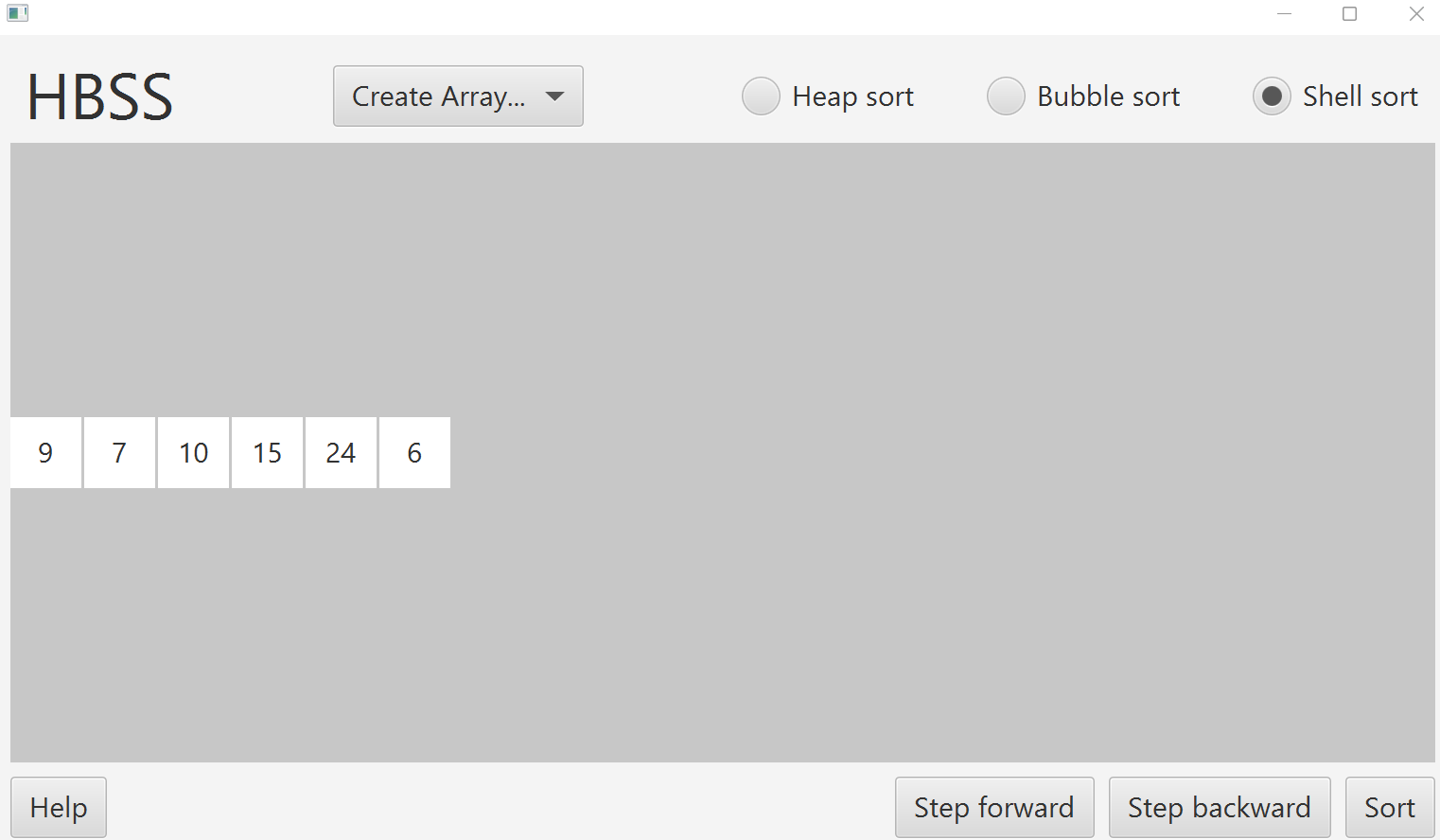
After input the length of array, there are n random unordered elements appear in the center of scene



When user chooses User-input values, he/she must enters sequentially the values of element in array



These values appear in the screen



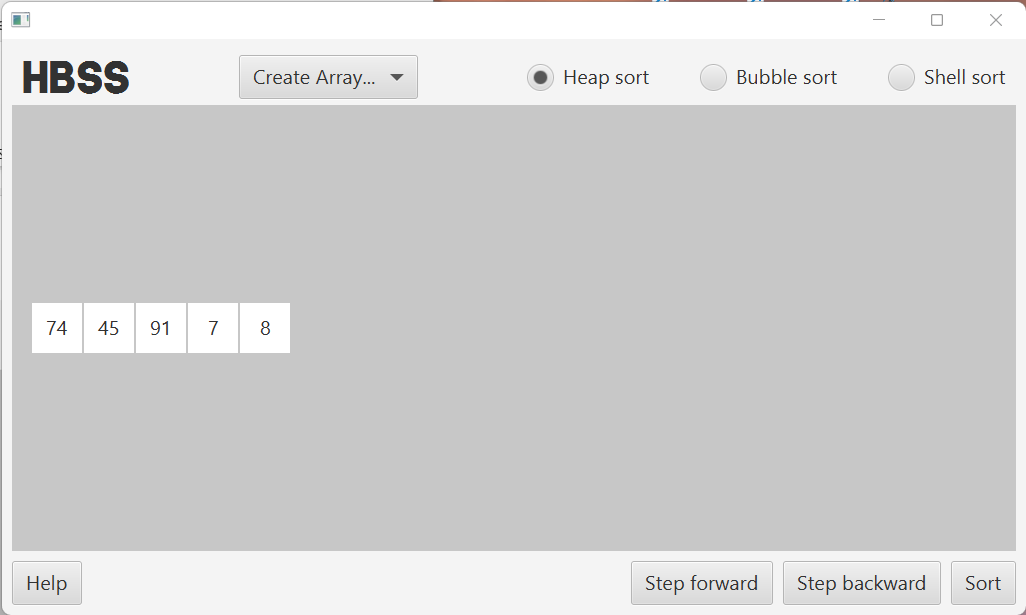
User can choose one of three sorting algorithms and click sort to see the result:

+ Choose Heap sort

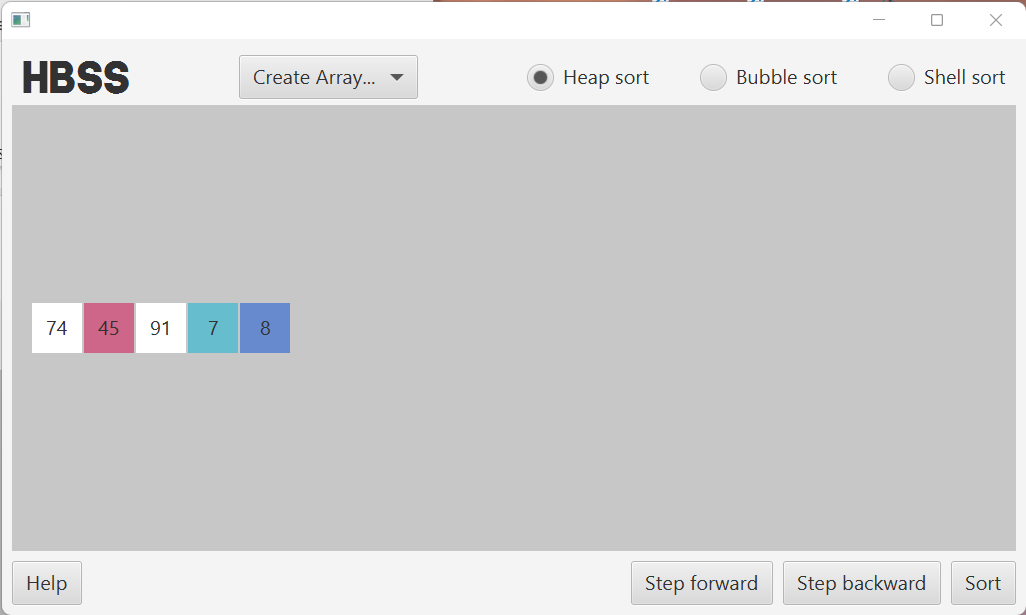
Graphical user interface, text, application

Description automatically generated

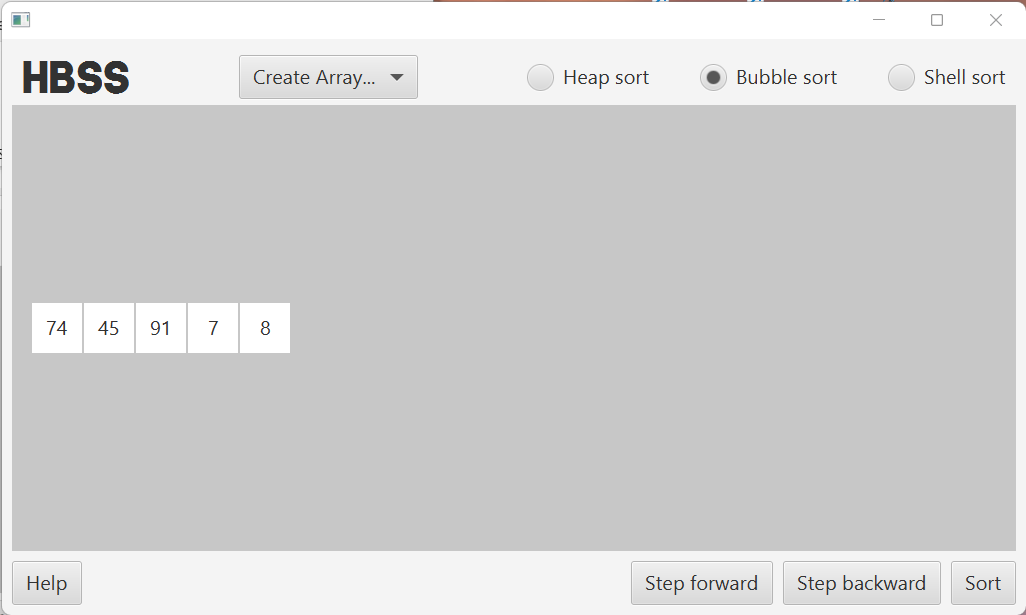
Clicking Step backward to see the previous state of array



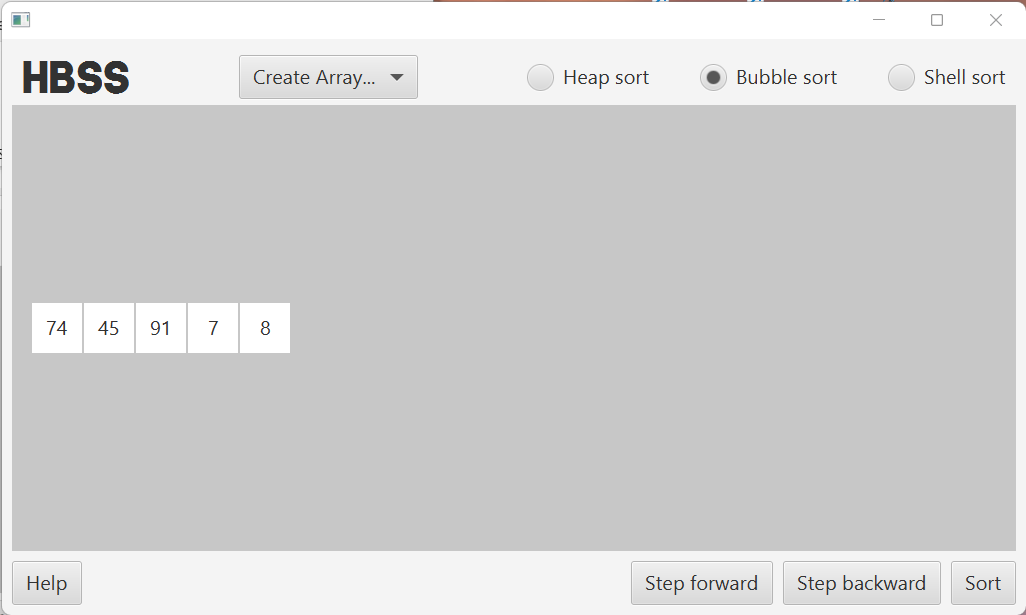
Clicking Step forward to see the next state of array



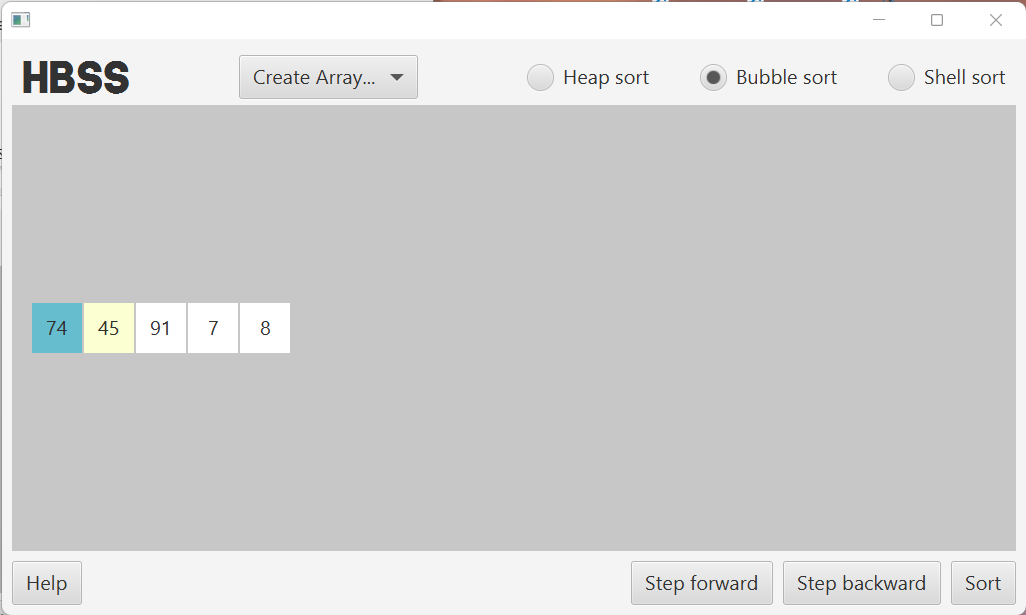
+Choose Bubble sort



Clicking Step backward to see the previous state of array



Clicking Step forward to see the next state of array

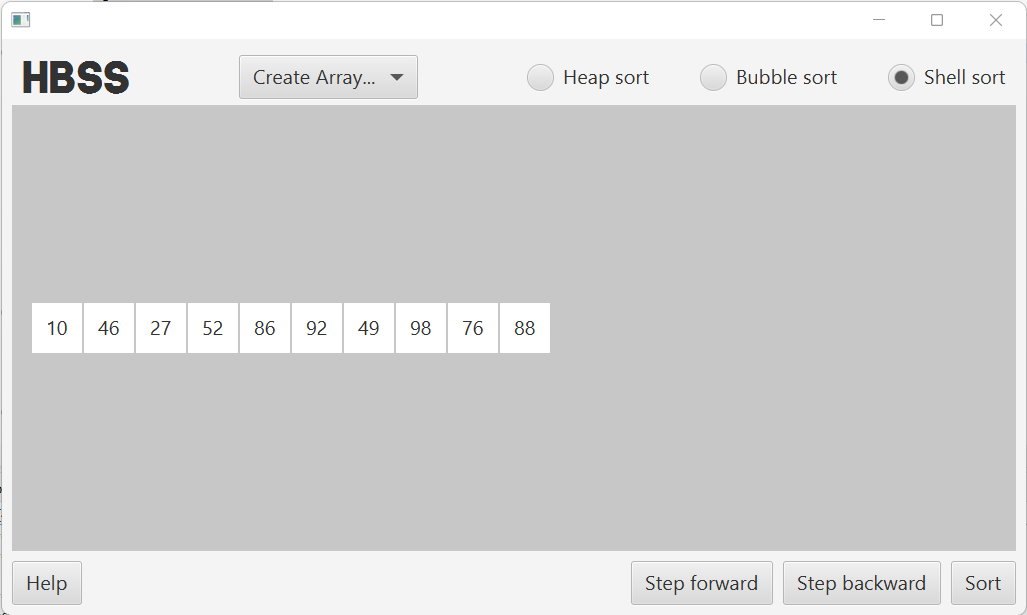


+Choose Shell sort

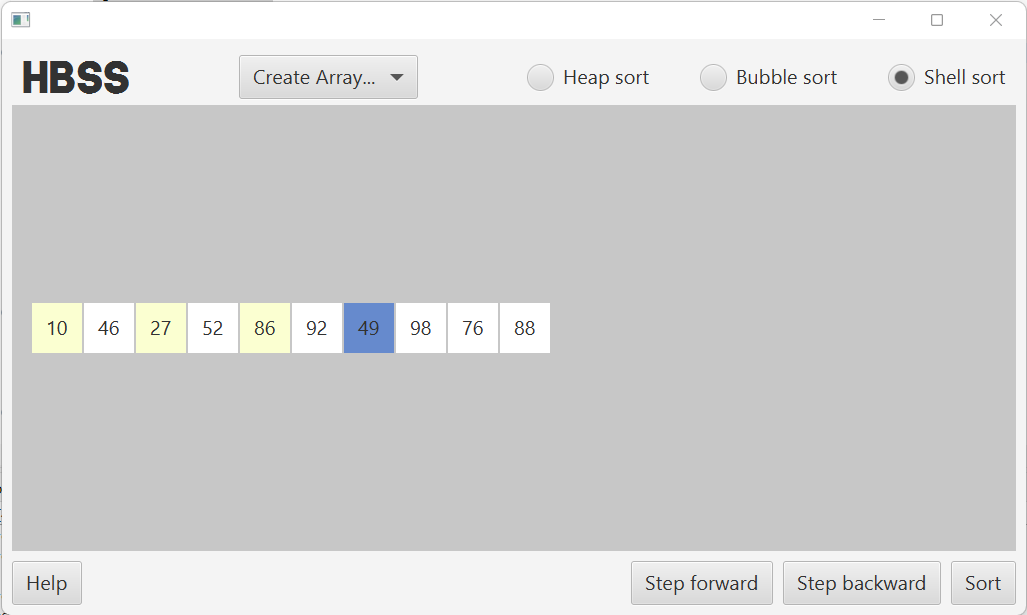
Graphical user interface, text, application

Description automatically generated

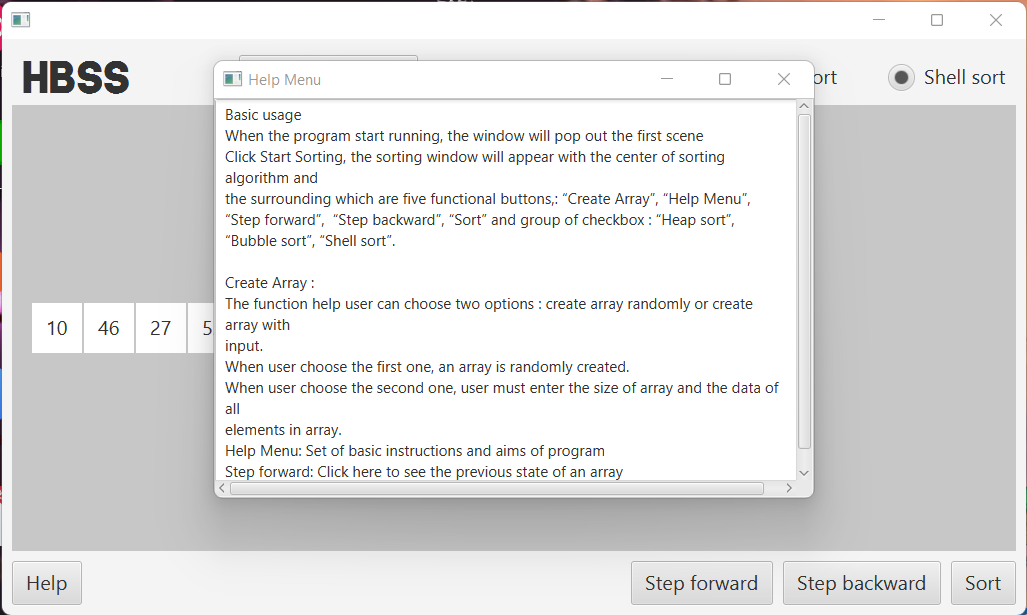
Clicking Step backward to see the previous state of array



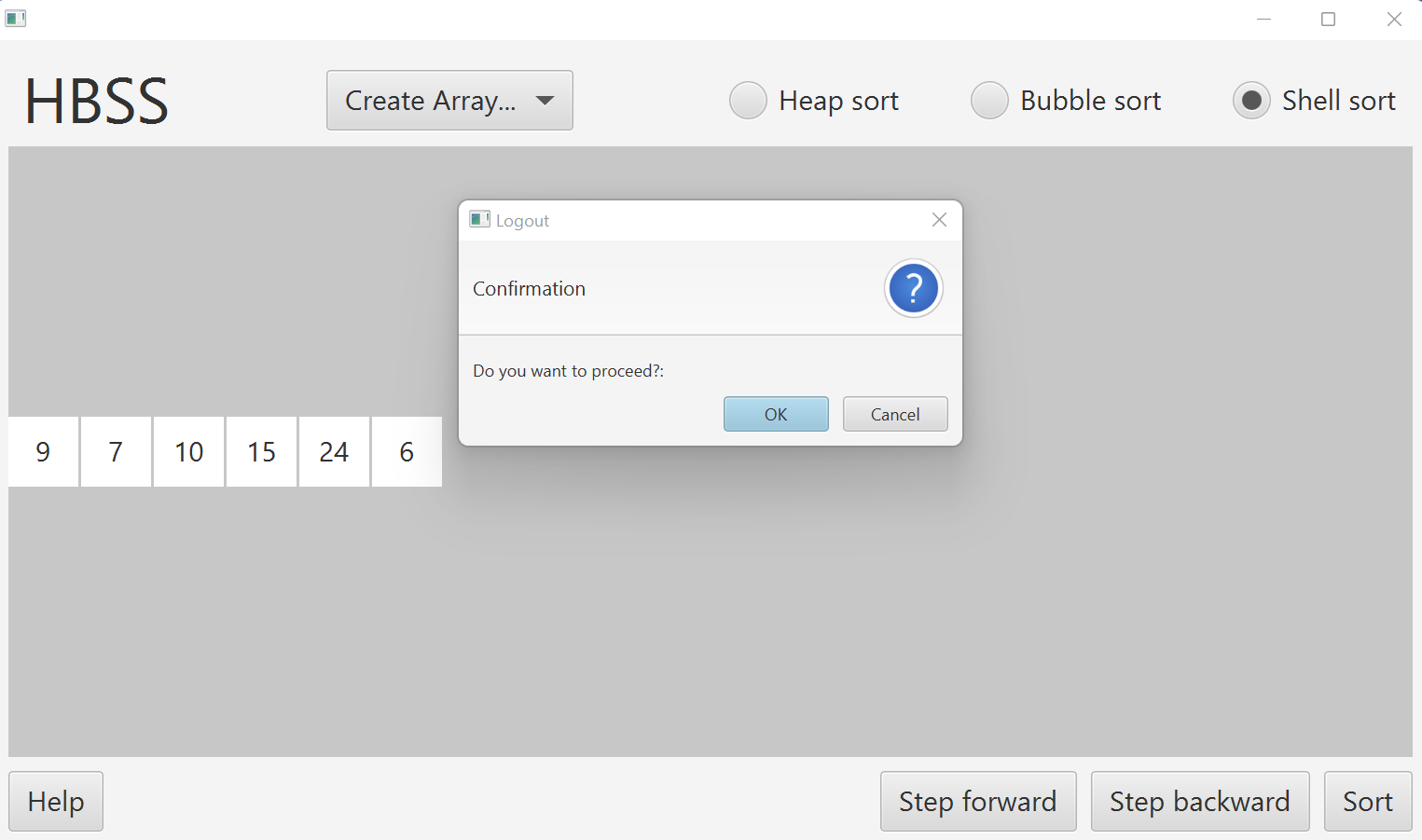
Clicking Step forward to see the next state of array (below we have stepped forward a few iterations)



Clicking **Help** to see basic usage and aim of application



When user want to exit, the dialog will appear to ensure that user does want to quit.



# Analysis on each sorting algorithm

## Bubble Sort

This is one of the most simple sorting algorithms out there. All we need to do is track two **adjacent** values, or in this case to see if the first value is smaller than the one behind it. If that is true, then move on to the next pair, if not, then those two elements **swap** positions.

After the first loop, the highest value of element will be moved to the bottom of the array, after the second loop, the second value of element will be moved to the second position from the bottom, and so on,... At the end, the sort algorithm is completed and the array has increasing-order elements.

## Heap Sort

First, the array of random numbers will be formatted into a **heap** - a tree-based data structure in which the tree is a complete **binary** tree. The HeapSort will start from the **last** parent node (downward and rightmost from the top of the tree. If that parent node happens to be smaller than the bigger child node out of the two child nodes, we will **swap** the parent node with that child node.

After each loop the goal of the program is to build a **max heap**, which is a binary tree in which the parent is always larger than the child nodes. One loop ends when the positions of the root node and the last node are swapped. The BuildMaxHeap continues in the following loops, until the whole array is sorted.

## Shell Sort

The **shell sort**, sometimes called the “diminishing increment sort,” improves on the insertion sort by breaking the original list into a number of smaller sublists, each of which is sorted using an insertion sort. The unique way that these sublists are chosen is the key to the shell sort. Instead of breaking the list into sublists of contiguous items, the shell sort uses an increment i, sometimes called the **gap**, to create a sublist by choosing all items that are i items apart.

**END OF REPORT**