A PROJECT REPORT

ON

A JAVA GUI VISUALIZATION OF BUBBLE SORT

ABSTRACT

This project titled "A JAVA GUI VISUALIZATION OF BUBBLE SORT" implements a GUI-based visualization of the bubble sort algorithm. The program utilizes Swing components for the graphical interface, including text fields, combo boxes, buttons, and labels. The main panel displays an array of elements visually represented as labeled boxes. These boxes change color during sorting to highlight swaps and comparisons. Users can input the array size, values, and sorting order (ascending or descending). Additionally, the program provides buttons to run the sorting algorithm, reset inputs, and navigate through the sorting steps. The sorting process is executed and visualized step by step. Each step shows the current array state and any performed swaps. Overall, this program provides a user-friendly interface for visualizing and understanding the bubble sort algorithm's behavior on an integer array.

TABLE OF CONTENT

TABLE OF FIGURES

CHAPTER I: Introduction

1.1 Introduction to Project

The project aims to develop a Java-based graphical user interface (GUI) application for visualizing the bubble sort algorithm. Bubble sort is a simple sorting algorithm that repeatedly steps through the list to be sorted, compares each pair of adjacent items, and swaps them if they are in the wrong order. The primary objective of this project is to create an interactive platform that enables users to observe the step-by-step execution of the bubble sort algorithm on a given array, enhancing their understanding of its functionality and performance.

1.2 Problem statement

Understanding sorting algorithms, such as bubble sort, can be challenging for individuals without a background in computer science or programming. Traditional methods of learning algorithms often involve textual descriptions or static visualizations, which may not effectively convey the algorithm's behavior and dynamics. Thus, there is a need for an intuitive and interactive tool that facilitates visual learning and comprehension of sorting algorithms, particularly bubble sort.

1.3 Objective

The main objective of this project is to develop a Java GUI application that provides a dynamic and interactive visualization of the bubble sort algorithm. Specific objectives include:

- Implementing a user-friendly interface for inputting array size, values, and sorting order.
- Designing visual elements to represent the array and its elements during sorting.
- Developing functionality to execute the bubble sort algorithm step by step and visualize each iteration.
- Enabling user control over the sorting process, allowing navigation through sorting steps and resetting inputs.
- Enhancing user understanding of the bubble sort algorithm through real-time visualization and interaction.

CHAPTER II: Implementation

The project was implemented using a structured methodology comprising the following steps:

2.1 Requirement Analysis

User's requirements for the bubble sort visualization application were analyzed. Consideration was given to the input methods for array size, values, and sorting order, as well as the desired level of interactivity during the sorting process.

2.2 Data and Information Gathering

Relevant data and information about the working of bubble sort algorithm and how it could be visualized through a GUI were collected. This involved studying some existing bubble sort visualization programs on internet, exploring GUI development techniques in Java, and understanding user preferences for interactive learning tools.

2.3 Tools and Techniques

Programming Language: Java was chosen as the programming language due to its versatility, platform independence, and strong support for object-oriented programming.

Integrated Development Environment (IDE): Visual Studio Code (VS Code) was utilized as the integrated development environment for its lightweight nature and extensive support for Java development.

Swing Library of Java Programming was selected as the primary tool for implementing the GUI application.

Event-driven programming technique was employed to handle user interactions with GUI components. Additionally, strategies were made to dynamically update visual elements to reflect the sorting process to user.

2.4 Procedure

The GUI application was designed and implemented using a modular approach. Features for inputting array size, values, and sorting order were incorporated using text fields and combo boxes. Visualization of the sorting process was achieved by dynamically updating the visual representation of the array elements using JLabels within a JPanel. User interaction was facilitated through action listeners attached to buttons for running the sorting algorithm, resetting inputs, and navigating through sorting steps.

2.5 Testing and Iteration

In-depth testing of the application was conducted to ensure its functionality and usability. Testing scenarios included input validation, correct visualization of sorting steps, and responsiveness of

user interface components. Feedback from testing was used to identify and address any bugs or issues, leading to iterative refinements in the code.

2.6 Feedback and Refinement:

Feedbacks were taken from the potential users about the application's effectiveness in facilitating understanding of the bubble sort algorithm. Based on feedback received, refinements were made to improve user experience, enhance visualization clarity, and optimize application performance.

2.7 User defined methods and classes

BubbleSortVisualization Class: This class serves as the main entry point of the application. It extends the JFrame class to create the main window of the application. Within this class, the GUI components such as text fields, buttons, and panels are initialized. Action listeners are attached to the buttons to handle user interactions.

runSorting() Method: This method is responsible for executing the bubble sort algorithm based on the user input. It parses the input array values, initializes the intermediateArrays list, and calls the bubbleSort() method to perform the sorting. After sorting, it updates the currentIndex and calls the updateVisualization() method to display the sorting steps.

bubbleSort() Method: This method implements the bubble sort algorithm. It iterates through the array elements and compares adjacent elements to determine if they need to be swapped based on the selected sorting order (ascending or descending). During each iteration, the intermediateArrays list is updated to store the intermediate states of the array.

updateVisualization() Method: This method updates the visualization of the sorting process. It retrieves the current state of the array from the intermediateArrays list based on the currentIndex and updates the JLabels representing the array elements in the arrayPanel. It also updates the stepLabel and swapCountLabel to provide feedback on the sorting progress.

reset() Method: This method resets the application to its initial state. It clears the input fields, resets the currentIndex, swapCount, and sortingComplete flags, and removes all JLabels from the arrayPanel.

ActionListener Interface Implementation: The actionPerformed(ActionEvent e) method implements the ActionListener interface to handle user actions such as clicking buttons. It determines the action performed based on the event source and invokes the corresponding method (e.g., runSorting(), reset()).

GUI Components Initialization: Various GUI components such as text fields, combo boxes, buttons, and labels are initialized within the constructor of the BubbleSortVisualization class. Layout managers such as GridLayout and BorderLayout are used to organize the components within the inputPanel and visualizationPanel.

CHAPTER III: Source Code

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.Vector;

public class BubbleSortVisualization extends JFrame implements ActionListener {
    private JTextField sizeField, arrayField;
    private JComboBox<String> orderComboBox;
    private JButton runButton, resetButton, leftButton, rightButton;
    private JLabel stepLabel, swapCountLabel;
    private JPanel inputPanel, visualizationPanel, arrayPanel;

    private List<int[]> intermediateArrays;
    private List<Integer> compare;
    private int currentIndex = 0;
    private int swapCount = 0;
    private boolean sortingComplete = false;
    private boolean back=false;

    public BubbleSortVisualization() {
        setTitle("Bubble Sort Visualization");
        setSize(800, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        inputPanel = new JPanel(new GridLayout(5, 2));
        visualizationPanel = new JPanel(new BorderLayout());
```

```
arrayPanel = new JPanel(new GridLayout(1, 10, 5, 5));

sizeField = new JTextField(10);
arrayField = new JTextField(20);
orderComboBox = new JComboBox<>(new String[]{"Ascending", "Descending"});
runButton = new JButton("Run");
resetButton = new JButton("Reset");
leftButton = new JButton("◀");
rightButton = new JButton("▶");
stepLabel = new JLabel();
swapCountLabel = new JLabel();

runButton.addActionListener(this);
resetButton.addActionListener(this);
leftButton.addActionListener(this);
rightButton.addActionListener(this);

inputPanel.add(new JLabel("Array Size:"));
inputPanel.add(sizeField);
inputPanel.add(new JLabel("Array Values (comma-separated):"));
inputPanel.add(arrayField);
inputPanel.add(new JLabel("Order of Sorting:"));
inputPanel.add(orderComboBox);
inputPanel.add(new JLabel());
inputPanel.add(runButton);
inputPanel.add(new JLabel());
inputPanel.add(resetButton);

visualizationPanel.add(leftButton, BorderLayout.WEST);
visualizationPanel.add(arrayPanel, BorderLayout.CENTER);
visualizationPanel.add(rightButton, BorderLayout.EAST);
```

```
    visualizationPanel.add(stepLabel, BorderLayout.SOUTH);
    visualizationPanel.add(swapCountLabel, BorderLayout.NORTH);

    add(inputPanel, BorderLayout.NORTH);
    add(visualizationPanel, BorderLayout.CENTER);
    setVisible(true);
}

@Override
public void actionPerformed(ActionEvent e) {
    if (e.getSource() == runButton) {
        runSorting();
    } else if (e.getSource() == resetButton) {
        reset();
    } else if (e.getSource() == leftButton) {
        if (currentIndex > 0) {
            back=true;
            currentIndex--;
            updateVisualization();
        }
    } else if (e.getSource() == rightButton) {
        if (currentIndex < intermediateArrays.size() - 1) {
            back=false;
            currentIndex++;
            updateVisualization();
        }
    }
}

private void runSorting() {
    String[] values = arrayField.getText().split(",");
```

```
    int[] initialArray = new int[values.length];
    for (int i = 0; i < values.length; i++) {
        initialArray[i] = Integer.parseInt(values[i].trim());
    }

    intermediateArrays = new ArrayList<>();
    compare=new Vector<>();
    intermediateArrays.add(initialArray.clone());

    swapCount = 0;
    bubbleSort(initialArray);
    currentIndex = 0;
    updateVisualization();
}

private void bubbleSort(int[] arr) {
    int n = arr.length;
    boolean swapped;
    for (int j = 0; j < n-1; j++) {
        swapped = false;
        for (int i = 0; i < n-j-1; i++) {
            if (orderComboBox.getSelectedIndex() == 0) {
                if (arr[i] > arr[i + 1]) {
                    int temp = arr[i];
                    arr[i] = arr[i + 1];
                    arr[i + 1] = temp;
                    swapped = true;
                }
            } else {
                if (arr[i] < arr[i + 1]) {
                    int temp = arr[i];
```

```
                arr[i] = arr[i + 1];
                arr[i + 1] = temp;
                swapped = true;
            }
        }
        compare.add(i);
        intermediateArrays.add(arr.clone());
    }
    if(!swapped)
        break;
  }
}

private void updateVisualization() {
   if (currentIndex == intermediateArrays.size() - 1) {
      stepLabel.setText("Sorting is complete");
      swapCountLabel.setText("Total swaps: " + swapCount);
      sortingComplete=true;
   } else {
      sortingComplete=false;
      int[] currentArray = intermediateArrays.get(currentIndex);
      int[] nextArray = intermediateArrays.get(currentIndex + 1);
      boolean noSwappingNeeded = true;
      StringBuilder swapValues = new StringBuilder();

      for (int i = 0; i < currentArray.length; i++) {
         if (currentArray[i] != nextArray[i]) {
            if (!noSwappingNeeded) {
               swapValues.append(", ");
            }
            swapValues.append(currentArray[i]).append(" <-> ").append(nextArray[i]);
```

```
            noSwappingNeeded = false;
            break;
        }
    }

    if (noSwappingNeeded) {
        stepLabel.setText("Step "+ (currentIndex+1) +": (No swapping needed)");
        swapCountLabel.setText("Swap count: " + swapCount);
    } else if(back) {
        stepLabel.setText("Step "+ (currentIndex+1) +": Swap values: " + swapValues);
        swapCountLabel.setText("Swap count: " + swapCount);
        if(currentIndex!=0)
            swapCount--;
        back=false;
    } else {
        stepLabel.setText("Step "+ (currentIndex+1) +": Swap values: " + swapValues);
        swapCountLabel.setText("Swap count: " + ++swapCount);
    }
}

int[] currentArray = intermediateArrays.get(currentIndex);
int currentCmp=-1;
int[] nextArray=null;
if(!sortingComplete){
    currentCmp=compare.get(currentIndex);
    nextArray = intermediateArrays.get(currentIndex + 1);
}

arrayPanel.removeAll();
for (int i = 0; i < currentArray.length; i++) {
    JLabel box = new JLabel(Integer.toString(currentArray[i]));
```

```
            box.setBorder(BorderFactory.createLineBorder(Color.BLACK));
            box.setHorizontalAlignment(JLabel.CENTER);

            if(!sortingComplete && (i==currentCmp || i==currentCmp+1)){
                if(currentArray[i] != nextArray[i]){
                    box.setOpaque(true);
                    box.setBackground(Color.RED);
                }
                else{
                    box.setOpaque(true);
                    box.setBackground(Color.ORANGE);
                }
            }
            else{
                box.setOpaque(true);
                box.setBackground(Color.WHITE);
            }
            arrayPanel.add(box);
        }
        arrayPanel.revalidate();
        arrayPanel.repaint();
    }

    private void reset() {
        sizeField.setText("");
        arrayField.setText("");
        orderComboBox.setSelectedIndex(0);
        intermediateArrays = null;
        currentIndex = 0;
        swapCount = 0;
        sortingComplete = false;
```

```
        stepLabel.setText("");
        swapCountLabel.setText("");
        arrayPanel.removeAll();
        arrayPanel.revalidate();
        arrayPanel.repaint();
    }

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

CHAPTER IV: Output

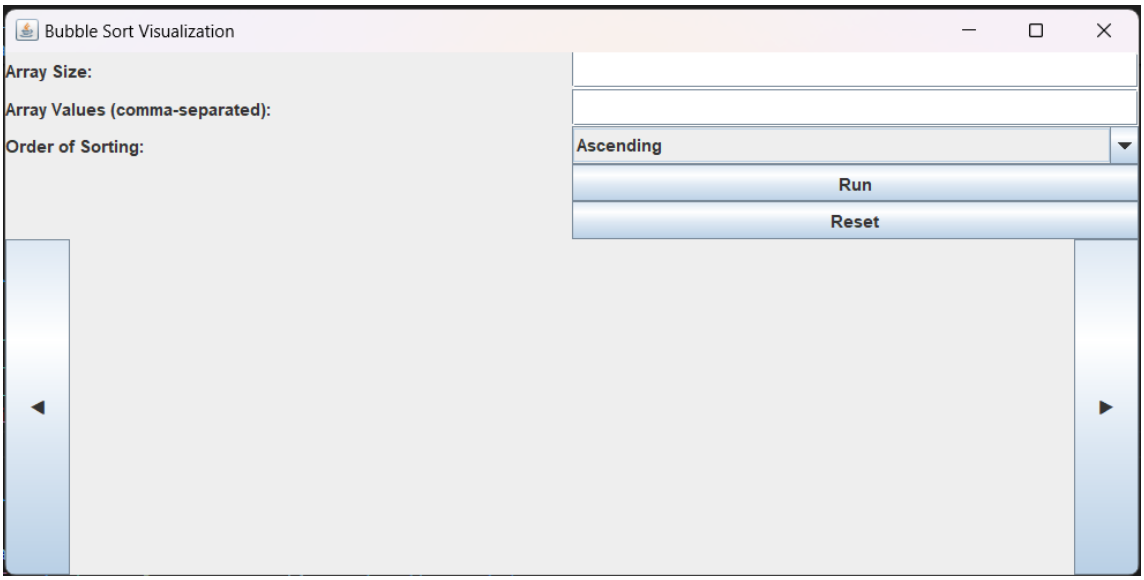

Figure 1 : Primary Interface

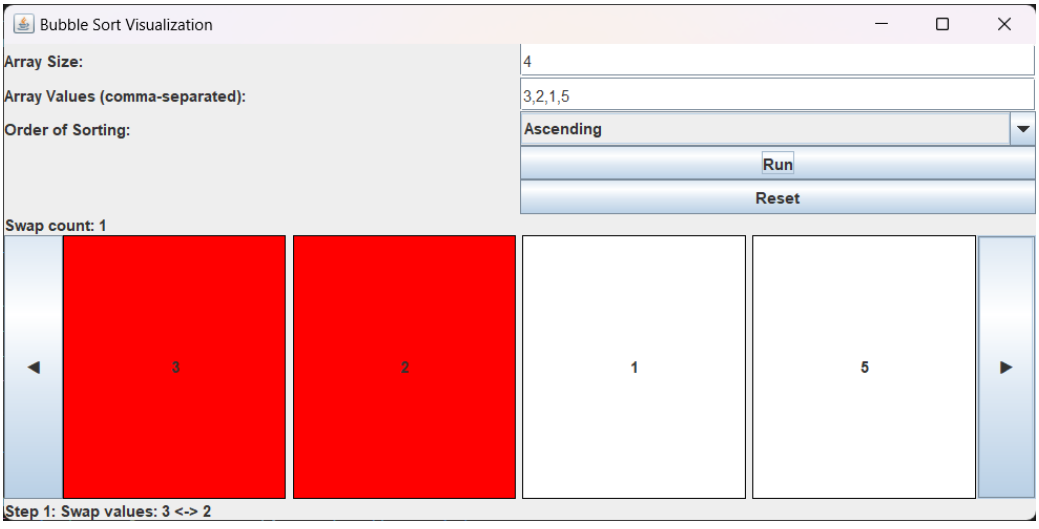

Figure 2: Ascending Sorting (Step 1)

Figure 3: Ascending Sorting (Step 2)

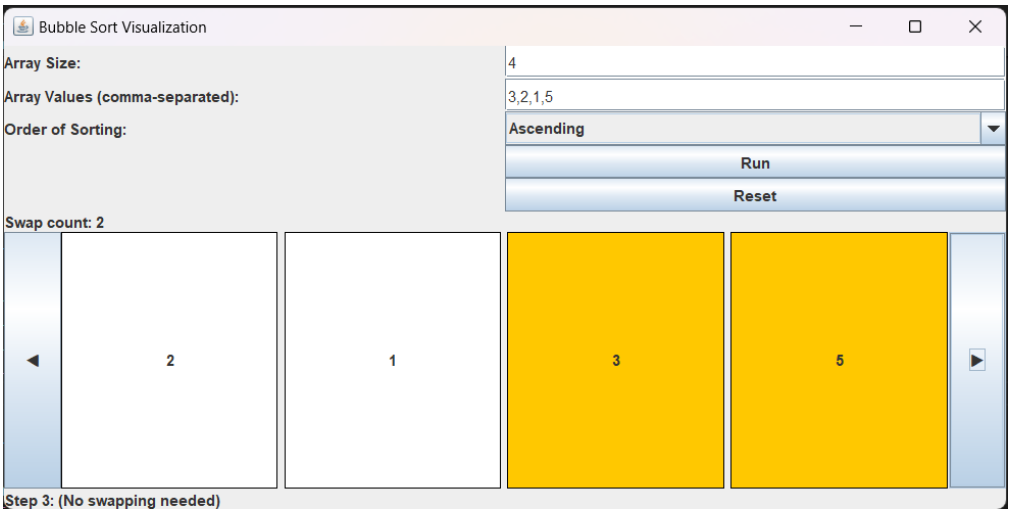

Figure 4: Ascending Sorting (Step 3)

Figure 5: Ascending Sorting (Step 4)

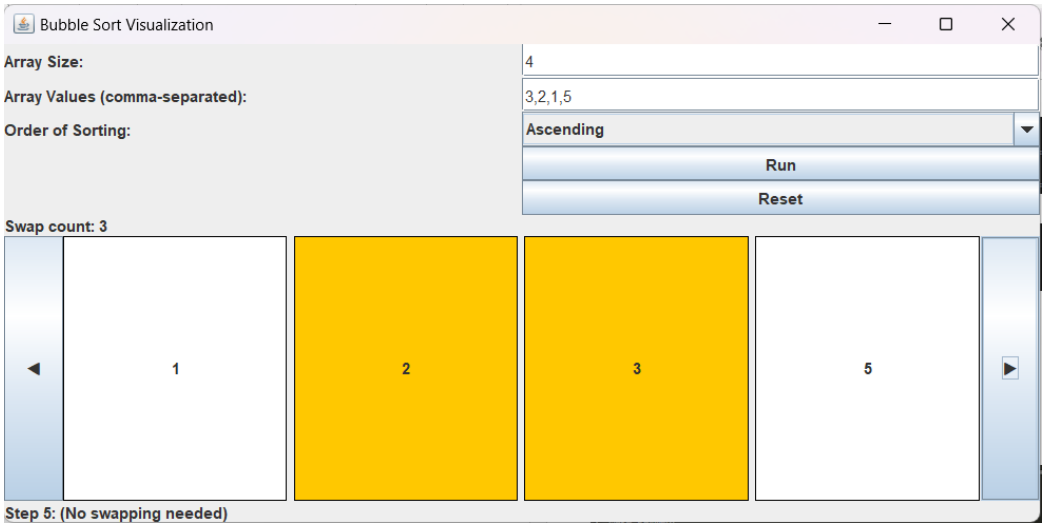

Figure 7: Ascending Sorting (Step 5)

Figure 6: Ascending Sorting (Step 6)

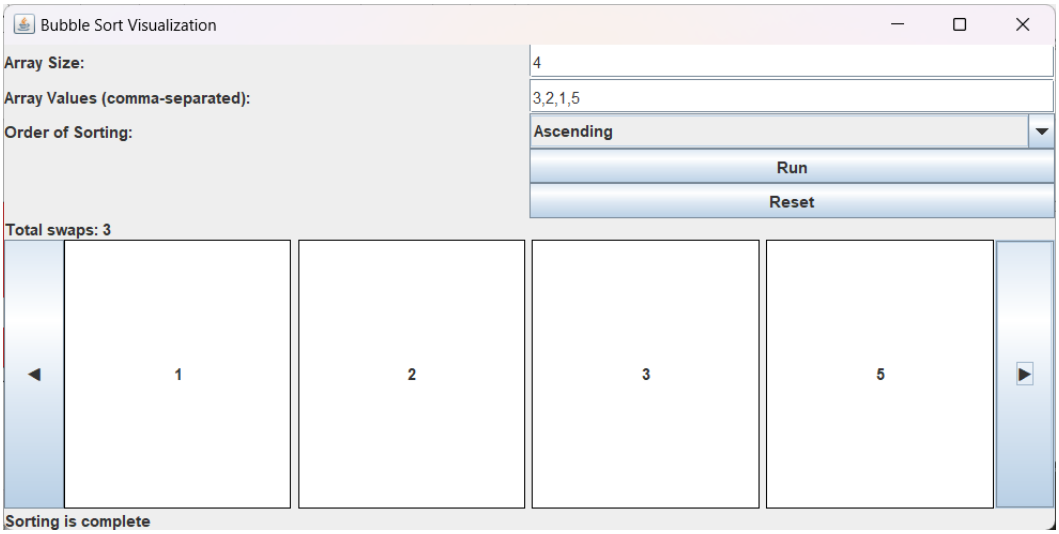

Figure 8: Ascending Sorting Completed

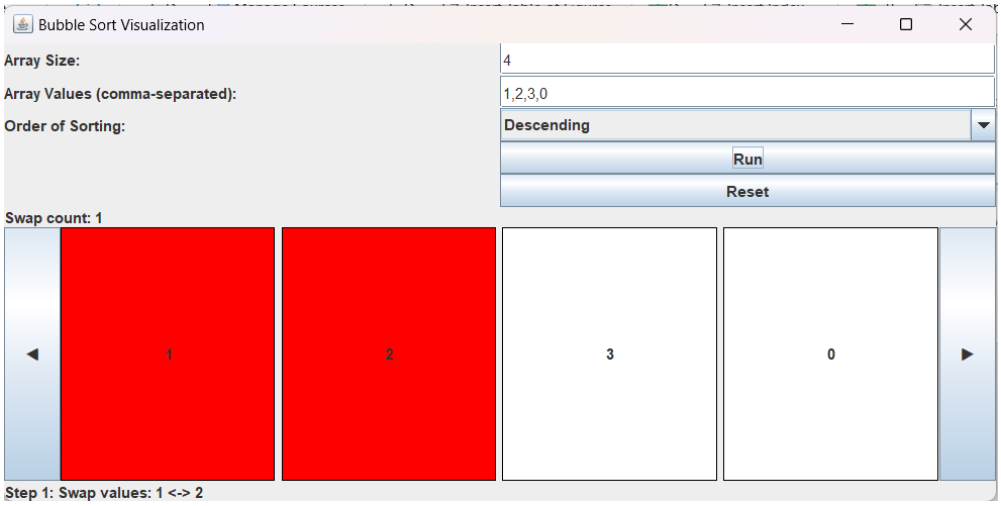

Figure 9: Descending Sorting (Step 1)

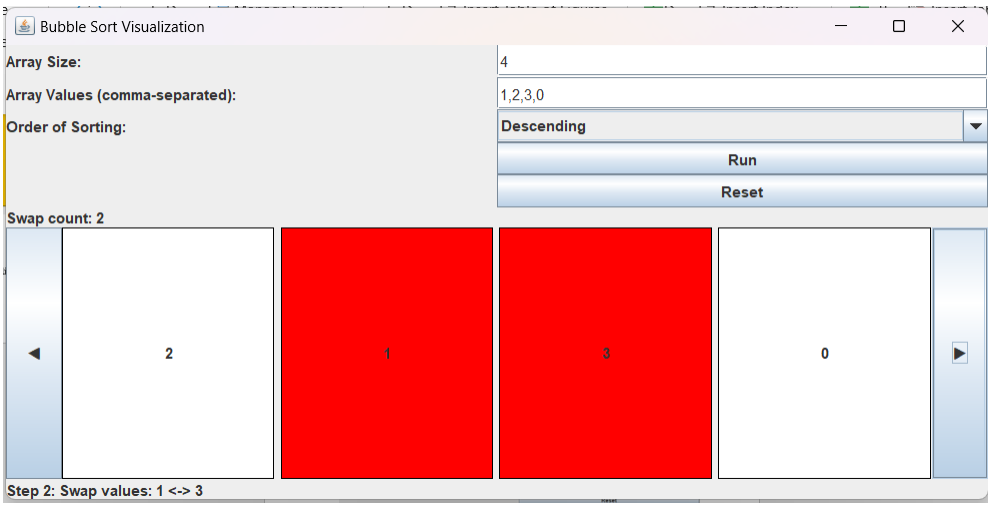

Figure 10: Descending Sorting (Step 2)

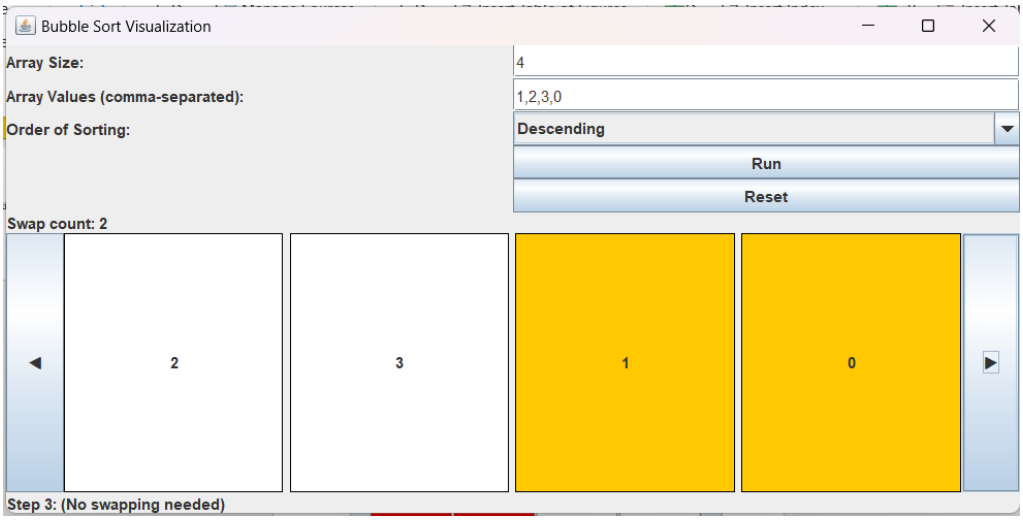

Figure 11: Descending Sorting (Step 3)

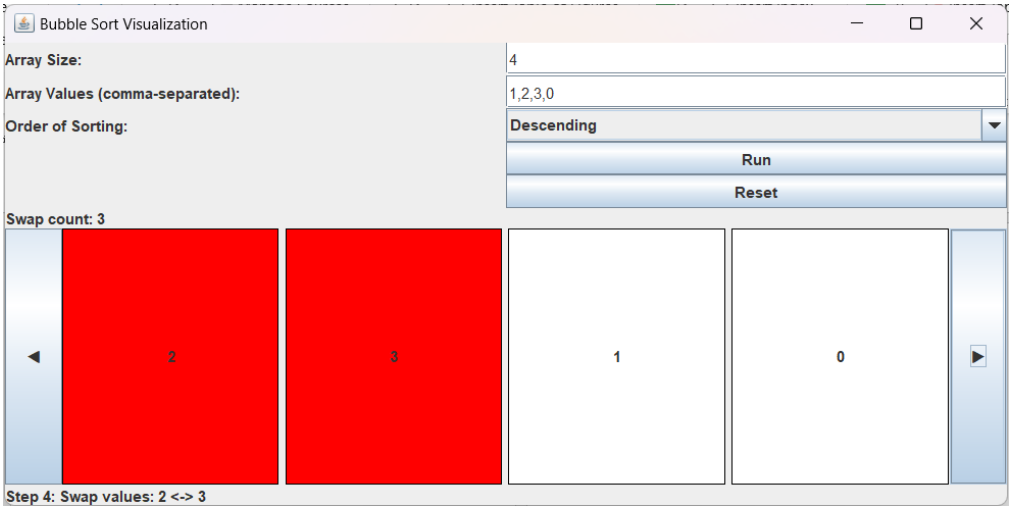

Figure 12: Descending Sorting (Step 4)

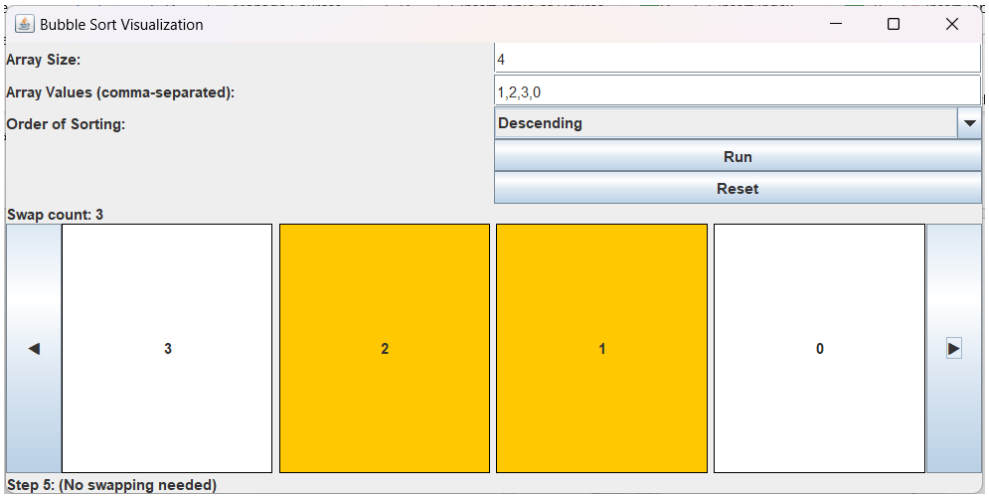

Figure 13: Descending Sorting (Step 5)

Figure 14: Descending Sorting (Step 6)

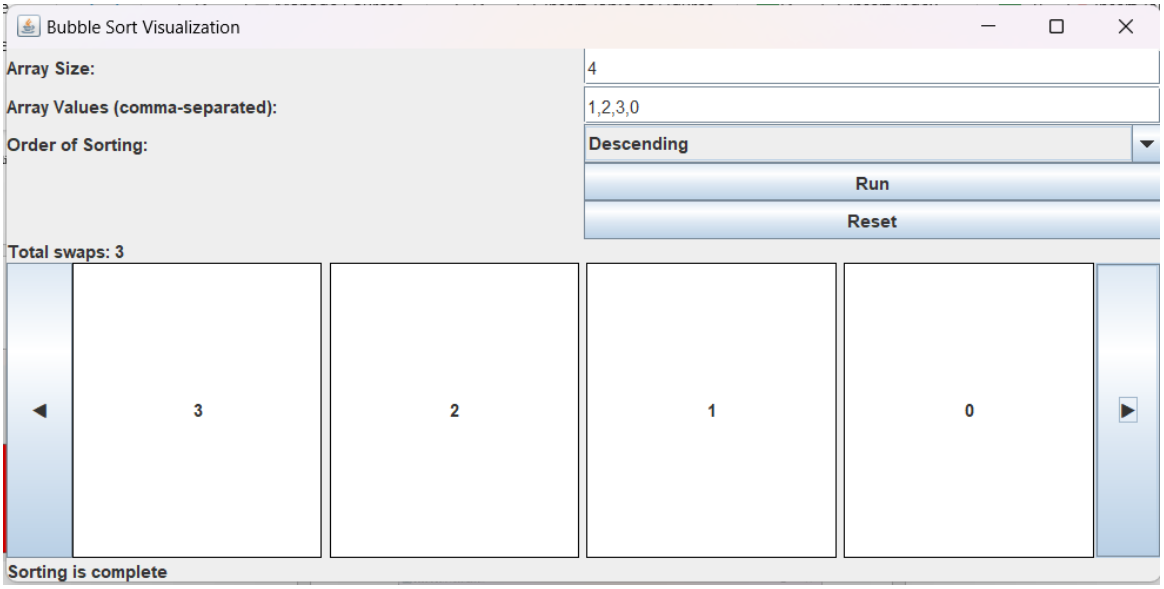

Figure 15: Descending Sorting Completed

CHAPTER V: Conclusion

In conclusion, a user-friendly and interactive tool was successfully developed for enhancing understanding of bubble sort algorithms. Through the utilization of Java programming with Swing GUI components, the project successfully created a dynamic platform where users can input arrays, observe the step-by-step execution of the bubble sort algorithm, and visualize sorting progress in real-time. By incorporating modular design principles, iterative refinement cycles, and user feedback, the project achieved its objectives of providing an intuitive interface, facilitating visual learning, and improving comprehension of sorting algorithms. Despite encountering limitations such as constraints of the Swing library, performance issues with large datasets and performance issues with floating point numbers, this project lays a foundation for further exploration and refinement in the field of algorithm visualization and educational software development. Overall, the bubble sort visualization project contributes to the advancement of educational tools and techniques, empowering users to explore and understand fundamental concepts in computer science and programming with greater clarity and engagement.