Object Oriented Programming (COMP6699001)

Final Project Documentation

Philipus Adriel Tandra: 2502031715

Data visualizer for sorting algorithms

Table of Contents

Program Description 3
Class Diagram 4
Lessons learned 5
Code explanation 5
Conclusion 13
Project Link: 13

Program Description

This data visualizer is built with the help of the Java Swing framework and helps visualize the following sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort. With the help of swing workers, we can gradually see how these sorting algorithms sort data.

As someone who has been learning data structures, the concept of sorting algorithms can be somewhat complex but with the help of this program, anyone is able to see these algorithms in action.

Modules:

- Java Swing
    - Used to create the frame and panel for visualizer as well as the long-running tasks in the background.
- Java AWT
    - Used for action listeners for the different buttons in the program
- Java Util (Random)
    - Used to randomize the array to start with it unsorted

Files:

- Visualizer.java
    - Inherits from JFrame and is the driver code
- Sorting.java
    - Holds the visualizer graphics, buttons and the sorting algorithm

The program starts off with randomized white lines with a black background. There are 4 buttons in the top of the program which showcase different algorithms. It includes Bubble sort, Insertion sort, Selection sort and Shuffle.

Whenever any of the sorting algorithms are pressed, the program runs and gradually ends with a staircase as the height of these lines represent the values of arrays which need to be sorted. The shuffle button on the other hand, randomises the array values and as such, the line lengths are also randomized

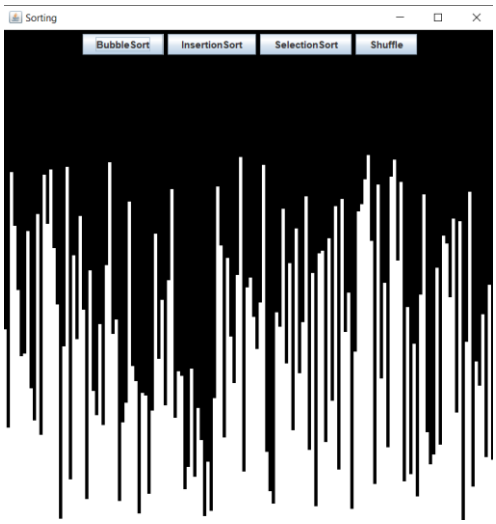
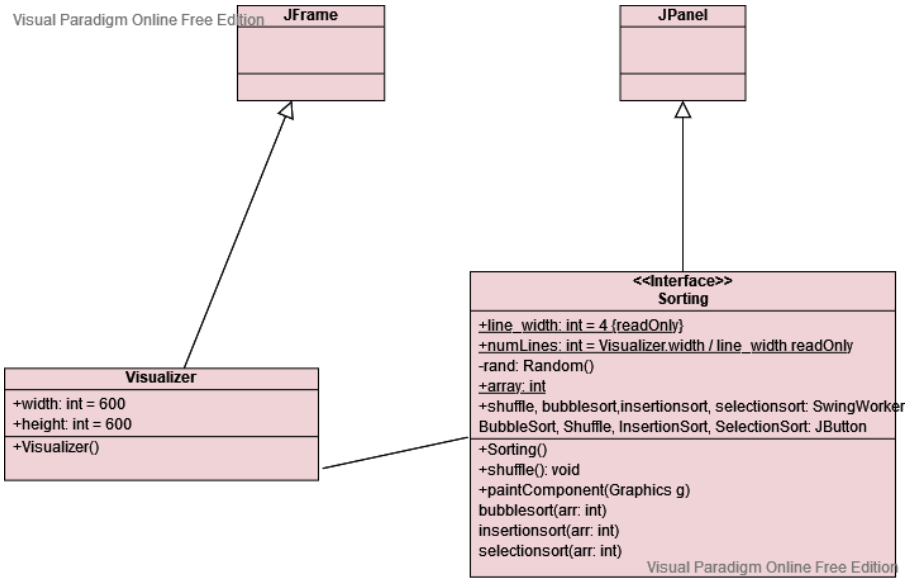


Figure I: Starting Screen

Class Diagram



Lessons learned

During the conception of this program, I learnt the Java Swing framework. The experience from first semester definitely helped with Java Swing having a few similarities with Pygame such as initializing the window, shapes and GUI work in general. I also learnt about action listeners and buttons and SwingWorker. However, they do not pale in comparison to the amount of Object-Oriented Programming learn during the development process. Java is an excellent programming language for helping developers learn about classes, methods and attributes at a reasonable pace. Sir Jude's lectures and exercises have also helped in practical problem-solving skills.

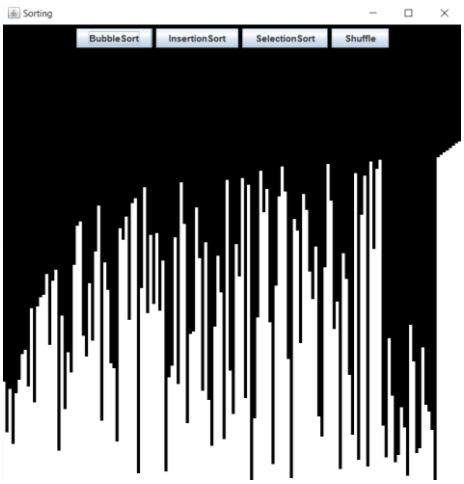
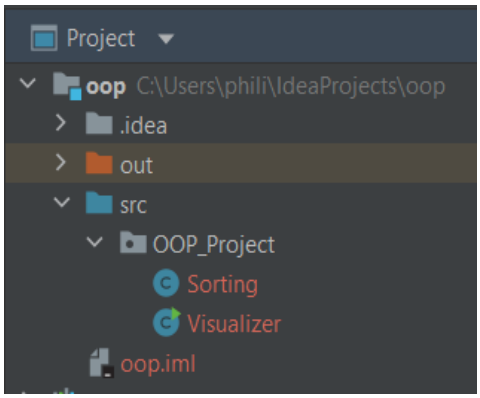


Figure II: Sorting

Code explanation

Project structure

The OOP_Project package holds the two classes, Sorting and Visualizer. This helps group related classes and helps me to initialize and use methods from different files and integrate them into others.



The Visualizer holds the JFrame and as such initializes the window frame size and the JPanel which will be found in the Sorting class.

Frame code

```java
package OOP_Project;
import javax.swing.*;
import java.awt.*;

//Inherit JFrame class and as a result taking in JFrame's methods
public class Visualizer extends JFrame {
    //Initialize window size
    public static final int width = 600;
    public static final int height = 600;
    //Initialize JPanel
    private Sorting panel = new Sorting();
    //Standard window code for title and showing JPanel contents
    public Visualizer() {
        this.setTitle("Sorting");
        this.getContentPane().setPreferredSize(new Dimension( width: 603,  height: 603));
        //Implement JPanel
        this.getContentPane().add(panel);
        this.setVisible(true);
        //Makes sure that the window size is at or above its preferred size in line 15
        this.pack();
        this.repaint();
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

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

Visualizer.java inherits from JFrame and as such is able to use JFrame's methods. The attributes are the width and height of the frame, however for the actually initialization of the frame, I decided to make it slightly bigger as the graphics that I will go over later on have pixel width to them and as such would require more frame size to show the full graphics. We also initialize the Sorting's main method with the getContentPane().add(panel) method. The methods after are standard for JFrame initialization. The main function also runs the main Visualizer method which holds Sorting's main method.

Sorting.java attributes

```java
//Inherit JPanel Class
public class Sorting extends JPanel {
    //Rectangle width
    public static final int line_width = 4;
    //Number of lines in the JPanel
    public static final int numLines = Visualizer.width / line_width;
    private Random rand = new Random();
    public final int[] array;
    //Swingworker for timers
    public SwingWorker<Void, Void> shuffle, bubblesort, insertionsort, selectionsort;
    //JButtons for the following functions
    JButton bubble = new JButton( text: "BubbleSort");
    JButton reset = new JButton( text: "Shuffle");
    JButton insert = new JButton( text: "InsertionSort");
    JButton select = new JButton( text: "SelectionSort");
```

As Sorting inherits from JPanel, it also inherits JPanel's attributes. Here we have a few mathematical notations that I will go over later but for now, we have initialized the width of the graphics of the visualizer to 4 pixels. We also implement Java.util.random for the randomizing of the graphics and array which is also initialized. SwingWorker is used for the delay for the following functions. There is also the initialization of JButton with their respective sorting algorithms.

Shuffle method

```java
//Shuffle Method
public void shuffle() {
    shuffle = new SwingWorker<>() {
        protected Void doInBackground() throws InterruptedException {
            //Randomises the indexes
            for (int i = 0; i < numLines; i++) {
                int swapindex = rand.nextInt(numLines);
                int temp = array[i];
                array[i] = array[swapindex];
                array[swapindex] = temp;
                Thread.sleep( millis: 1);
                repaint();
            }
        return null;
    }
};shuffle.execute();
```

The shuffle method shuffles the values of each index in the array with rand.nextInt(). We use a for loop as we want to randomize and go through all the indexes in the array. It also has a SwingWorker, this means that the process runs in the background with a 1 millisecond delay as seen in the Thread.sleep. Repaint() is also used for update of graphics and the shuffle.execute runs the SwingWorker.

paintComponent method

```
//paintComponent for JPanel and graphics for the rectangles
public void paintComponent(Graphics g) {
    Graphics2D graphics = (Graphics2D) g;
    super.paintComponent(g);
    graphics.setColor(Color.white);
    //Loops through to create a staircase of rectangles
    for (int x = 0; x < numLines; x++) {
        //Initializing locations of the rectangles
        int height = array[x] * 3;
        int x_coordinate = x + (line_width - 1) * x;
        int y_coordinate = Visualizer.height - height;
        graphics.fillRect(x_coordinate, y_coordinate, line_width, height);
    }
}
```

This is standard paintComponent method for a JPanel with Graphics being the helped class and g being the object. We set the graphics color to white which means the graphics in the JPanel will be colored accordingly. A for loop is used to loop through all number of lines and draw that many lines which is in the form of rectangles. The height is simply a number that is dependent on the values of the index, in this case it would simply result in the graphics being a staircase of rectangles due to the rectangles being dependent on the values of the indexes in the array which is just ascending numbers to 150. The x coordinate is where the vertical rectangles would be horizontally in the JPanel. We want it to perfectly fit the dimensions in the Visualizer.java file so we do a few mathematical notations to keep it all fit together as we need to somehow fit 150 vertical rectangles in a window. We also have the y coordinate which would be where the rectangles are vertically. The other two parameters for

graphics.fillRect will be the width and height of the rectangle which is already the aforementioned variables. With this we now have graphics that change height based on the value of the indexes.

Bubble Sort method

```
//Bubble Sort initialization, taken from GeeksForGeeks.org
void bubbleSort(int arr[]) {
    bubblesort = new SwingWorker<>() {
        @Override
        protected Void doInBackground() throws InterruptedException {
            int n = arr.length;
            for (int i = 0; i < n - 1; i++)
                for (int j = 0; j < n - i - 1; j++)
                    if (arr[j] > arr[j + 1]) {
                        // swap arr[j+1] and arr[j]
                        int temp = arr[j];
                        arr[j] = arr[j + 1];
                        arr[j + 1] = temp;
                        Thread.sleep( millis: 1);
                        repaint();
                    }
        return null;
    }
};bubblesort.execute();
```

The bubble sort algorithm takes the array and similarly to the shuffle method, it also uses SwingWorker. This means that it also animates through the same way. Bubble sort is a sorting algorithm that loops through an array while comparing adjacent indexes. If they are larger or smaller, they are swapped accordingly. It is one of the simplest sorting algorithms It has an O(n^2) time complexity as seen through the two for loops. That said, it is also one of the slowest sorting algorithms due to high number of swaps needed to be done and high number of temporary variables needed.

Insertion sort method

```
//Insertion Sort initialization, taken from GeeksForGeeks.org
void insertionsort(int arr[]) {
    insertionsort = (SwingWorker) () -> {
            int n = arr.length;
            for (int i = 1; i < n; ++i) {
                int key = arr[i];
                int j = i - 1;
    /* Move elements of arr[0..i-1], that are
       greater than key, to one position ahead
       of their current position */
                while (j >= 0 && arr[j] > key) {
                     arr[j + 1] = arr[j];
                     j = j - 1;
                }
                arr[j + 1] = key;
                Thread.sleep( millis: 10);
                 repaint();
            }
            return null;
    };insertionsort.execute();
}
```

The insertion sort algorithm takes an array and splits it apart by defining the values as sorted an unsorted. It places sorted values in a completely separate temporary array while the unsorted values are swapped to sort. Insertion sort has a time complexity O(n) in the best-case scenario which would be when the array is already sorted but roughly the same time complexity as bubble sort. Insertion sort is also faster even though it has the same time complexity due to the aforementioned problems with bubble sort. Similarly, it also uses the same SwingWorker method but has a longer sleep time due to being much quicker than bubble sort.

Selection Sort method

```
    //Selection Sort initialization, taken from GeeksForGeeks.org
    void selectionsort(int arr[]) {
        selectionsort = new SwingWorker<>() {
            @Override
            protected Void doInBackground() throws InterruptedException {
                // One by one move boundary of unsorted subarray
                int n = arr.length;
                for (int i = 0; i < n - 1; i++) {
                    // Find the minimum element in unsorted array
                    int min_idx = i;
                    for (int j = i + 1; j < n; j++)
                        if (arr[j] < arr[min_idx])
                            min_idx = j;

                    // Swap the found minimum element with the first
                    // element
                    int temp = arr[min_idx];
                    arr[min_idx] = arr[i];
                    arr[i] = temp;
                    Thread.sleep( millis: 10);
                    repaint();
                }
                return null;
            }
        };selectionsort.execute();
    }
}
```

The selection sort divides the array in two parts by putting sorted indexes on the left whereas the unsorted indexes are on the right. This process continues until the entire array is sorted. It also uses SwingWorker and like InsertionSort also has a longer delay due to it being much faster than bubble sort.

Main Sorting method

```
//Main Function
public Sorting(){
    this.add(bubble);
    this.add(insert);
    this.add(select);
    this.add(reset);
    setBackground(Color.black);
    //Initializing array and its indexes being as much as the numLines variable
    array = new int[numLines];
    for (int i = 0; i < numLines; i++) {
        array[i] = i;
    }
    //To start of shuffled
    shuffle();
    //Executing the action listener through the button
    bubble.addActionListener(e -> bubbleSort(array));
    insert.addActionListener(e -> insertionsort(array));
    reset.addActionListener(e -> shuffle());
    select.addActionListener(e -> selectionsort(array));
}
```

This is the main function that calls the other methods. It also adds the aforementioned button attributes to the JPanel. We also have the background color as black as shown. The array is also initialized in this function by making the values of the array the same as the index. We call the shuffle method as the paintComponent initializes the graphics as already sorted due to it simply drawing rectangles in ascending order and we want to start it off shuffled as otherwise, we would be unable to see the sorting algorithms in action. We also have action

listeners for the buttons, however I decided to use lambda functions as it shortens the code and there is no need for the implement of ActionListener in the Sorting Class.

Figure III: Sorted

Conclusion

This project and program were created through the combination of the material of two different classes. Data Structures taught me the wonders of sorting algorithms while Object Oriented Programming taught me the importance of inheritance and classes in general. Overall, I am thankful for the material given and I cannot help but admit that I have improved as a developer in the aftermath of this project. All in all, I just want to say:

Thank you, Sir Jude.

Project Link:

https://github.com/CH1MP5T0N/OOP_Project_DataVisualizerSortingAlgorithms