**K L HYDERABAD**

**FRESHMAN ENGINEERING DEPARTMENT**

**A Project Based Lab Report**

**On**

**TITLE**

**Sorting Technique GUI**

**SUBMITTED BY:**

2310030011 Sree Krishna Ashrith

2310030019 Ashmit Rushikesh Pathak

2310030210 Digvijay Rajoria

**UNDER THE ESTEEMED GUIDANCE OF**

**Dr. K. Ramasubramanian**

**Associate Professor**



**KONERU LAKSHMAIAH EDUCATION FOUNDATION**

(Deemed to be University)

Moinabad Road, Aziz Nagar, Hyderabad – 500075

**DEPARTMENT OF BASIC ENGINEERING SCIENCES**

****

**CERTIFICATE**

This is to certify that the project based laboratory report entitled

“Sorting Techniques GUI” submitted by Mr. Digvijay Rajoriabearing Regd. No. 2310030210 to the **Department of Basic Engineering Sciences, KL University** in partial fulfilment of the requirements for the completion of a project in **“Computational Thinking for Object Oriented Design -23SC1203**”course in I Year B Tech II Semester, is a bonafide record of the work carried out by him/her under my supervision during the academic year 2023-24.

**PROJECT SUPERVISOR**  **HEAD OF THE DEPARTMENT**

Dr. K. Ramasubramanian Dr. G. REKHA

**ACKNOWLEDGEMENTS**

It is great pleasure for me to express my gratitude to our honourable President **Sri. Koneru Satyanarayana**, for giving the opportunity and platform with facilities in accomplishing the project based laboratory report.

I express the sincere gratitude to our Principal **Dr. A. RamaKrishna** for his administration towards our academic growth.

I express sincere gratitude to our Coordinator and HOD - BES **Dr. G. Rekha** for her leadership and constant motivation provided in successful completion of our academic semester. I record it as my privilege to deeply thank for providing us the efficient faculty and facilities to make our ideas into reality.

I express my sincere thanks to our project supervisor **Dr.K.Ramasubramanian** for his novel association of ideas, encouragement, appreciation and intellectual zeal which motivated us to venture this project successfully.

Finally, it is pleased to acknowledge the indebtedness to all those who devoted themselves directly or indirectly to make this project report success.

Digvijay Rajoria

2310030210

**ABSTRACT**

In this project, we aim to develop an interactive digital learning tool designed to enhance the educational experience of students studying computer science, specifically focusing on various sorting algorithms. As students at our university have shown a keen interest in digital learning, this tool leverages a graphical user interface (GUI) to provide an engaging and intuitive method for understanding sorting techniques.

The tool will enable students to input a set of numbers and observe the step-by-step execution of multiple sorting algorithms, including Bubble Sort, Shell Sort, Insertion Sort and Quick Sort. For each algorithm, the tool will visually display each step of the sorting process, allowing students to see how the algorithm manipulates data to achieve a sorted array. This visual representation aims to clarify the operational mechanics and logic behind each algorithm, making complex concepts more accessible.

In addition to step-by-step visualisation, the tool will provide a comparative analysis of the time complexities of the different sorting algorithms. Students will be able to see not only how each algorithm performs but also understand the theoretical underpinnings of their efficiency. The tool will highlight best-case, average-case, and worst-case scenarios for each sorting technique, providing a comprehensive understanding of their performance in various contexts.

By integrating interactive and visual elements into the learning process, this project seeks to bridge the gap between theoretical knowledge and practical understanding. It aims to foster a deeper comprehension of fundamental computer science principles and enhance students' problem-solving skills. Ultimately, this project demonstrates the power of digital learning tools in transforming traditional educational methods and improving student engagement and retention of complex concepts.

**INDEX**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TITLE** | **PAGE NO** |
| 1 | Introduction | 6 |
| 2 | Aim of the Project | 7 |
| 2.1 | Advantages & Disadvantages | 7 |
| 3 | Software & Hardware Details | 8 |
| 4 | Class Diagram | 9 |
| 5 | Implementation | 10 |
| 6 | Outputs/ScreenShots | 21 |
| 7 | Conclusion & Future Scope | 27 |

**INTRODUCTION**

In the rapidly evolving field of computer science, mastering fundamental concepts such as sorting algorithms is crucial for students. Sorting algorithms are not only essential tools for organising data but also provide valuable insights into algorithmic thinking and efficiency. Traditional teaching methods often struggle to make these abstract concepts tangible and engaging for students. To address this gap, our project aims to harness the power of digital learning to enhance the understanding and retention of sorting algorithms through an interactive graphical user interface (GUI).

This project will develop a comprehensive digital learning tool that allows students to explore various sorting techniques through dynamic, step-by-step visualisations. By inputting a set of numbers, students can observe how algorithms such as Bubble Sort, Insertion Sort, Quick Sort, and Shell Sort manipulate and order data. Each step in the sorting process will be displayed graphically, providing a clear and intuitive representation of the algorithm’s operations. Additionally, the tool will compare the time complexities of these algorithms, helping students appreciate their performance characteristics in various scenarios. This visual and interactive approach aims to create a more engaging and effective learning experience, equipping students with a solid foundation in computer science and enhancing their problem-solving skills.

**AIM**

The aim of this project is to develop an interactive digital learning tool that visually demonstrates the step-by-step execution of various sorting algorithms and compares their time complexities, enhancing students' understanding and retention of these fundamental computer science concepts.

**Advantages:**

1. **Enhanced Understanding**: The visual step-by-step execution of sorting algorithms helps students grasp the underlying mechanics and logic more effectively than traditional methods.
2. **Interactive Learning**: The interactive nature of the tool engages students, making learning more enjoyable and immersive.
3. **Comparative Analysis**: By comparing the time complexities of different algorithms, students can develop a deeper understanding of algorithm efficiency and performance in various scenarios.
4. **Self-Paced Learning**: Students can use the tool at their own pace, allowing them to revisit and review concepts as needed, which supports diverse learning styles.
5. **Practical Skills**: The hands-on experience with sorting algorithms helps students build practical problem-solving skills that are essential in computer science and programming.

**Disadvantages:**

1. **Resource Intensive**: Developing and maintaining the interactive tool can be resource-intensive, requiring significant time and technical expertise.
2. **Limited Scope**: While the tool focuses on sorting algorithms, it may not cover other important areas of computer science, necessitating additional resources for a comprehensive education.

**SYSTEM REQUIREMENTS**

* **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

Language : JAVA

Operating system **:** Windows 11 , M1&M2 .

Tools : Eclipse IDE, intelliJ IDE

* **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM : 8GB

Processor : Intel 32/64 Bit with 1GHZ

**CLASS DIAGRAM**

****

**IMPLEMENTATION**

import javax.swing.\*;

import java.awt.\*;

import java.awt.event.\*;

import java.util.ArrayList;

import java.util.Arrays;

public class SortingTechniquesGUI extends JFrame implements ActionListener {

private JPanel panel;

private JLabel labelInput, labelOutput;

private JTextField textFieldInput, textFieldOutput;

private JButton buttonBubbleSort, buttonQuickSort, buttonShellSort, buttonInsertionSort;

private JButton buttonPrevious, buttonNext;

private JTextArea textAreaSteps;

private JProgressBar progressBar;

private SortVisualizer sortVisualizer;

private ArrayList<int[]> sortingSteps;

private int currentStep;

private boolean isSorting;

private String[][] timeComplexities = {

{"O(n^2) ", "O(n^2) ", "O(n^2) Worst"}, // Bubble Sort

{"O(n log n) ", "O(n log n) ", "O(n^2) "}, // Quick Sort

{"O(n log n) ", "O(n^2) ", "O(n^2) "}, // Shell Sort

{"O(n) ", "O(n^2) ", "O(n^2) "} // Insertion Sort

};

public SortingTechniquesGUI() {

setTitle("Sorting Techniques");

setSize(800, 600);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLayout(new BorderLayout());

initComponents();

setVisible(true);

}

private void initComponents() {

panel = new JPanel();

panel.setLayout(new GridLayout(0, 2, 10, 10)); // 2 columns, 10px horizontal and vertical gap

panel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

labelInput = new JLabel("Enter numbers (comma separated):");

textFieldInput = new JTextField();

textFieldInput.setPreferredSize(new Dimension(200, 30)); // Set preferred size for text field

labelOutput = new JLabel("Sorted numbers:");

textFieldOutput = new JTextField();

textFieldOutput.setPreferredSize(new Dimension(200, 30)); // Set preferred size for text field

buttonPrevious = new JButton("Previous");

buttonPrevious.addActionListener(this);

buttonNext = new JButton("Next");

buttonNext.addActionListener(this);

textAreaSteps = new JTextArea("Sorting Steps:\n");

textAreaSteps.setEditable(false);

JScrollPane scrollPane = new JScrollPane(textAreaSteps);

progressBar = new JProgressBar();

progressBar.setStringPainted(true);

panel.add(labelInput);

panel.add(textFieldInput);

panel.add(labelOutput);

panel.add(textFieldOutput);

buttonBubbleSort = createButton("Bubble Sort", timeComplexities[0]);

buttonQuickSort = createButton("Quick Sort", timeComplexities[1]);

buttonShellSort = createButton("Shell Sort", timeComplexities[2]);

buttonInsertionSort = createButton("Insertion Sort", timeComplexities[3]);

panel.add(buttonBubbleSort);

panel.add(buttonQuickSort);

panel.add(buttonShellSort);

panel.add(buttonInsertionSort);

panel.add(scrollPane);

panel.add(progressBar);

panel.add(buttonPrevious);

panel.add(buttonNext);

add(panel, BorderLayout.CENTER);

sortVisualizer = new SortVisualizer();

add(sortVisualizer, BorderLayout.SOUTH);

}

private JButton createButton(String text, String[] timeComplexities) {

JButton button = new JButton(text);

button.addActionListener(this);

button.setToolTipText(getToolTipText(timeComplexities)); // Set tooltip

return button;

}

private String getToolTipText(String[] timeComplexities) {

return String.format("<html><b>Time Complexities:</b><br>Best Case: %s<br>Average Case: %s<br>Worst Case: %s</html>",

timeComplexities[0], timeComplexities[1], timeComplexities[2]);

}

@Override

public void actionPerformed(ActionEvent e) {

if (e.getSource() == buttonPrevious) {

showPreviousStep();

} else if (e.getSource() == buttonNext) {

showNextStep();

} else {

handleSortingAction(e);

}

}

private void handleSortingAction(ActionEvent e) {

String input = textFieldInput.getText();

if (!isValidInput(input)) {

JOptionPane.showMessageDialog(this, "Invalid input. Please enter numbers separated by commas.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

int[] originalArray = Arrays.stream(input.split(","))

.map(String::trim)

.mapToInt(Integer::parseInt)

.toArray();

sortingSteps = new ArrayList<>();

currentStep = 0;

isSorting = true;

if (e.getSource() == buttonBubbleSort) {

sortingSteps = SortingAlgorithms.bubbleSort(originalArray.clone());

} else if (e.getSource() == buttonQuickSort) {

sortingSteps = SortingAlgorithms.quickSort(originalArray.clone());

} else if (e.getSource() == buttonShellSort) {

sortingSteps = SortingAlgorithms.shellSort(originalArray.clone());

} else if (e.getSource() == buttonInsertionSort) {

sortingSteps = SortingAlgorithms.insertionSort(originalArray.clone());

}

if (!sortingSteps.isEmpty()) {

updateGUI(sortingSteps.get(currentStep));

}

}

private boolean isValidInput(String input) {

return input.matches("^\\d+(,\\s\*\\d+)\*$");

}

private void updateGUI(int[] array) {

SwingUtilities.invokeLater(() -> {

StringBuilder stepsText = new StringBuilder("Sorting Steps:\n");

for (int num : array) {

stepsText.append(num).append(" ");

}

textAreaSteps.setText(stepsText.toString());

sortVisualizer.updateArray(array);

progressBar.setValue((int) (((double) (currentStep + 1) / (sortingSteps.size())) \* 100));

if (currentStep == sortingSteps.size() - 1) {

isSorting = false;

StringBuilder sortedArrayText = new StringBuilder();

for (int num : array) {

sortedArrayText.append(num).append(", ");

}

if (sortedArrayText.length() > 0) {

sortedArrayText.delete(sortedArrayText.length() - 2, sortedArrayText.length());

}

textFieldOutput.setText(sortedArrayText.toString());

}

});

}

private void showPreviousStep() {

if (currentStep > 0) {

currentStep--;

updateGUI(sortingSteps.get(currentStep));

}

}

private void showNextStep() {

if (currentStep < sortingSteps.size() - 1) {

currentStep++;

updateGUI(sortingSteps.get(currentStep));

}

}

public static void main(String[] args) {

new SortingTechniquesGUI();

}

}

class SortingAlgorithms {

public static ArrayList<int[]> bubbleSort(int[] array) {

ArrayList<int[]> steps = new ArrayList<>();

steps.add(array.clone());

for (int i = 0; i < array.length - 1; i++) {

for (int j = 0; j < array.length - i - 1; j++) {

if (array[j] > array[j + 1]) {

int temp = array[j];

array[j] = array[j + 1];

array[j + 1] = temp;

}

steps.add(array.clone());

}

}

return steps;

}

public static ArrayList<int[]> quickSort(int[] array) {

ArrayList<int[]> steps = new ArrayList<>();

quickSortHelper(array, 0, array.length - 1, steps);

return steps;

}

private static void quickSortHelper(int[] array, int low, int high, ArrayList<int[]> steps) {

if (low < high) {

int pi = partition(array, low, high);

steps.add(array.clone());

quickSortHelper(array, low, pi - 1, steps);

quickSortHelper(array, pi + 1, high, steps);

}

}

private static int partition(int[] array, int low, int high) {

int pivot = array[high];

int i = low - 1;

for (int j = low; j < high; j++) {

if (array[j] < pivot) {

i++;

int temp = array[i];

array[i] = array[j];

array[j] = temp;

}

}

int temp = array[i + 1];

array[i + 1] = array[high];

array[high] = temp;

return i + 1;

}

public static ArrayList<int[]> shellSort(int[] array) {

ArrayList<int[]> steps = new ArrayList<>();

steps.add(array.clone());

int n = array.length;

for (int gap = n / 2; gap > 0; gap /= 2) {

for (int i = gap; i < n; i++) {

int temp = array[i];

int j;

for (j = i; j >= gap && array[j - gap] > temp; j -= gap) {

array[j] = array[j - gap];

}

array[j] = temp;

steps.add(array.clone());

}

}

return steps;

}

public static ArrayList<int[]> insertionSort(int[] array) {

ArrayList<int[]> steps = new ArrayList<>();

steps.add(array.clone());

for (int i = 1; i < array.length; i++) {

int key = array[i];

int j = i - 1;

while (j >= 0 && array[j] > key) {

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = key;

steps.add(array.clone());

}

return steps;

}

}

class SortVisualizer extends JPanel {

private int[] array;

public SortVisualizer() {

setPreferredSize(new Dimension(600, 100));

setBackground(Color.WHITE);

}

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

if (array != null && array.length > 0) {

int barWidth = getWidth() / array.length;

int maxValue = Arrays.stream(array).max().orElse(1);

for (int i = 0; i < array.length; i++) {

int barHeight = (int) ((double) array[i] / maxValue \* getHeight());

int x = i \* barWidth;

int y = getHeight() - barHeight;

g.setColor(getColorForValue(array[i], maxValue));

g.fillRect(x, y, barWidth, barHeight);

g.setColor(Color.BLACK);

g.drawRect(x, y, barWidth, barHeight);

}

}

}

public void updateArray(int[] array) {

this.array = array;

repaint();

}

private Color getColorForValue(int value, int maxValue) {

float hue = 0.5f; // Constant hue

float saturation = 0.6f; // Constant saturation

float brightness = (float) value / maxValue; // Adjust brightness based on value

return Color.getHSBColor(hue, saturation, brightness);

}

public void clear() {

array = null;

repaint();

}

}

**OUTPUTS**

Step1:

A screenshot of a computer

Description automatically generated

Step 2:

A screenshot of a computer

Description automatically generated

Step 3:

A screenshot of a computer

Description automatically generated

Step 4:

A screenshot of a computer

Description automatically generated

Step 5:

**A screenshot of a computer

Description automatically generated**

Step 6:

**A screenshot of a computer

Description automatically generated**

Step 7:

**A screenshot of a computer

Description automatically generated**

Time Complexity of Bubble Sort:

A screenshot of a computer

Description automatically generated

Time Complexity of Quick Sort :

A screenshot of a computer

Description automatically generated

Time Complexity of Shell Sort:

A screenshot of a computer

Description automatically generated

Time Complexity of Insertion Sort:

A screenshot of a computer

Description automatically generated

**CONCLUSION & FUTURE SCOPE**

This project has successfully illustrated the powerful potential of interactive digital tools in enhancing the educational experience of computer science students. By developing a comprehensive GUI that visually demonstrates various sorting algorithms, the project has made complex and abstract concepts more tangible and easier to understand. Students can interact with the tool to observe the step-by-step execution of sorting techniques like Bubble Sort, Quick Sort, Shell Sort, and Insertion Sort, gaining a deeper and more intuitive understanding of these algorithms. Additionally, the tool's feature to compare the time complexities of different sorting algorithms equips students with valuable insights into algorithm efficiency and performance, which is crucial for their academic and professional growth.

The interactive nature of this learning tool engages students in a way that traditional methods often cannot, making the learning process more enjoyable and effective. By supporting self-paced learning and catering to diverse learning styles, the tool helps students build strong foundational knowledge and practical problem-solving skills essential for their future careers in computer science. Overall, this project demonstrates how digital learning can significantly transform and enhance the educational landscape, making complex topics more approachable and comprehensible.

The future scope of this project includes several enhancements and expansions to further enrich the educational experience:

1. **Expansion of Algorithms**: Broaden the scope of the tool to include a wider range of algorithms, such as searching algorithms, graph algorithms, and more complex data structures like trees and graphs. This will provide students with a more comprehensive learning resource.

2. **Adaptive Learning Techniques**: Incorporate adaptive learning features to personalise the educational experience. The tool could adapt to individual student needs and learning paces, offering tailored feedback and additional resources to address knowledge gaps and reinforce understanding.

3. **Real-World Applications**: Integrate real-world problem-solving scenarios and applications of algorithms to demonstrate their practical utility. Examples from fields such as data analysis, artificial intelligence, and software development could provide context and enhance student engagement.

4. **Collaborative Learning**: Expand the tool to support collaborative learning environments. Features such as peer reviews, group projects, and discussion forums could be integrated to facilitate collaboration and enhance learning outcomes.

5. **Gamification**: Introduce gamification elements, such as quizzes, challenges, and rewards, to make learning more engaging and motivating. This can help maintain student interest and encourage continuous learning.

6. **Mobile Compatibility**: Develop a mobile-friendly version of the tool to make it accessible on various devices, allowing students to learn and practice on-the-go.

7. **Performance Analytics**: Implement analytics to track student performance and progress. Educators can use this data to identify areas where students struggle and provide targeted support and interventions.

8. **Continuous Improvement**: Regularly update and refine the tool based on student feedback and the latest educational research. This will ensure that the tool remains effective, user-friendly, and aligned with current technological advancements and educational standards.

By pursuing these future developments, the project aims to become a more versatile and comprehensive educational resource, continuously improving the way sorting algorithms and other computer science concepts are taught and understood.