SORTING VISUALIZOR

A PROJECT REPORT

Submitted by

Yogesh kumar(23BCS11964)

Bhavuk(23BCS11207)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE & ENGINEERING

Chandigarh University

NOV 2025

BONAFIDE CERTIFICATE

Certified that this project report **"SORTING VISUALIZOR"** is the Bonafide work **"Yogesh kumar(23BCS11964) & Bhavuk(23BCS11207)""** has carried out the project work under my/our supervision.

Mr. Aniket Raj
Trainer(T2196)

Associate Director (CSE 3rd Year)

Department of Computer Science and Engineering

Assistant Professor

Department of Computer Science and Engineering

Submitted for the project viva-voce examination held on 07-11-2025.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

Sorting algorithms play a fundamental role in computer science and are essential for problem-solving and data processing. However, understanding how they function internally can be challenging for beginners when studied purely through code or theory. This project aims to simplify the learning process by developing a **Cpp-based Sorting Visualizer** that demonstrates how various sorting algorithms work in real-time through animated graphics.

The application supports eight widely used algorithms: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Bucket Sort. It allows users to select an algorithm, adjust the array size, and view the sorting steps through animated vertical bars. Additional features include a pseudocode display, comparison count tracker, and a clean, interactive GUI built using Java Swing.

The visualizer is designed to run offline and provide an educational, user-friendly platform for students and learners to understand sorting behavior clearly. By aligning theoretical knowledge with practical visuals, the project bridges the gap between code and concept.

CHAPTER 1.

INTRODUCTION

1.1. Client Identification/Need Identification/Identification of relevant Contemporary issue

Computer science education, sorting algorithms are foundational topics. However, students often struggle to understand their internal working mechanisms through code alone. A report by Code.org highlights that visual learning improves algorithmic understanding by up to **45%** compared to text-based explanation alone.

This identifies a clear gap**:** the need for a graphical, interactive tool that helps users understand sorting through real-time animations**.** The proposed solution aims to bridge this educational gap with a Java-based sorting visualizer.

1.2. Identification of Problem

Understanding sorting algorithms through theoretical explanations or code alone proves insufficient for many learners. The problem lies in the lack of **visual and interactive learning tools** that can demonstrate how these algorithms manipulate data step by step.

1.3. Identification of Tasks

Task No.	Task Title	Description
T1	Requirement Analysis	Study existing sorting visualizers, identify user needs and required features.
T2	Algorithm Selection	Select sorting algorithms to implement (e.g., Bubble, Selection, Merge, etc.).
T3	GUI Design	Design an interactive and user-friendly interface using Java Swing.
T4	Animation Logic Implementation	Develop the logic to visualize sorting steps dynamically and clearly.
T5	Integration & Functionality Testing	Test sorting animations and ensure they behave correctly with various inputs.
T6	Performance Evaluation	Compare algorithms based on time/space complexity with test data.

(Table 1.1 **Identification of Tasks**)

LITERATURE REVIEW/BACKGROUND STUDY

1.4. Timeline of the reported problem

Over the years, many students have found it difficult to understand how sorting algorithms work just by reading code or theory. This problem has been noticed by teachers and researchers worldwide.

In 2011, websites like **VisuAlgo** and other online tools started showing visual animations of algorithms to help students. These tools made learning better, but they still had some problems – like needing internet access and not allowing students to fully interact with the program.

This shows that the problem of understanding sorting still exists, and there is a need for a better tool that works offline, allows user control, and makes sorting easy to learn. Our project aims to solve this issue.

1.5. Proposed solutions

To help students understand sorting algorithms better, many solutions have been tried before. Some common ones include:

- **Using diagrams and code examples** in textbooks.
- **Online tools** like VisuAlgo that show sorting step by step using animations.
- **YouTube tutorials** that explain how sorting works with examples.
- **Practice websites** where students can write and test their own sorting code.

While these are helpful, they still have some drawbacks. Most of them **need internet**, don't allow much **user control**, and some don't show how the sorting happens **in real-time**. Also, not many desktop apps are available for offline use.

That's why this project focuses on creating a **sorting visualizer** that works offline, gives more interaction, and helps users understand sorting in a better and easier way.

1.6. Bibliometric analysis

Tool/Source	Features	What's Good	What's Missing
VisuAlgo	Shows sorting with animations (online)	Easy to use, covers many algorithms	Needs internet, no offline version
Sorting Animations (Web)	Visual bars move to show sorting steps	Simple and clear	No user control, very basic
YouTube Videos	Sorting explained through voice and video	Easy to understand	No interaction, only one-way explanation
Few Java Desktop Apps	Run on PC with GUI	Work offline	Limited features, not easily available

(Table 2.1 Bibliometric analysis of the Report)

2.4. Review Summary

After reviewing various tools and resources, it's clear that while sorting visualizers exist online, most of them lack important features like **user interaction**, **offline access**, and a **simple user interface**. Many students still find it hard to understand how sorting works just by looking at code or watching videos.

This project aims to solve that problem by creating a **sorting visualizer** that is **interactive**, **works offline**, and is easy to use. The learnings from the literature review helped in deciding the features and direction of this project.

2.5 Problem Definition

Many students struggle to understand how sorting algorithms actually work because existing learning methods are either too theoretical or non-interactive.

- **What is to be done:**
  A desktop application needs to be developed in Java that can **visually show how different sorting algorithms work** using bars and animations.
- **How it is to be done:**
  Java will be used to create a **Graphical User Interface (GUI)** where users can choose sorting algorithms, adjust speed, and input values. Sorting steps will be shown in real-time using moving bars.
- **What is not to be done:**
  This project will **not cover mobile app development**, **sorting of complex data types**, or **web-based deployment**. It focuses only on **basic sorting algorithms** and their visualization on desktop.

2.6 Goals/Objectives

The main goal of this project is to develop a Java-based application that visually demonstrates how different sorting algorithms work. The following are the specific and measurable objectives of the project:

- To design a user-friendly graphical interface using Java Swing for sorting visualization.
- To implement multiple sorting algorithms such as Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort.
- To display the sorting process in real-time using animated bars to enhance understanding.
- To provide user controls for adjusting input size and sorting speed.
- To develop a fully offline and lightweight application that can run on any desktop.
- To test all functionalities thoroughly to ensure correct working and smooth performance.
- To document the entire development process with clear explanations and references.

CHAPTER 2.

DESIGN FLOW/PROCESS

2.1. Evaluation & Selection of Specifications/Features

During the planning phase of the Sorting Visualizer project, several features were evaluated to enhance the learning experience for students and beginner programmers. The key priority was to make sorting algorithms understandable through visual representation and interactivity.

Real-time animation was identified as the most essential feature, allowing users to see how each element is moved or compared during the sorting process. Multiple sorting algorithms—including Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Bucket Sort—were included to provide variety and allow users to compare their behaviors.

A graphical user interface was designed using Java Swing, enabling users to interact with the application via a dropdown menu for algorithm selection, a slider to adjust the size of the array, and Start/Reset buttons to control the process. Additionally, a side panel displays the pseudocode and comparison count, helping users connect the animation with the actual logic of each algorithm. Some advanced features, such as speed control and export options, were intentionally left out to maintain simplicity, reduce complexity, and focus on the core educational purpose.

2.2. Design Constraints

During the design and development process, several practical constraints were considered:

- **Economic:** Open-source tools were used (Java and Swing).
- **Manufacturability:** The application had to run on basic desktop systems with no additional dependencies.
- **Environmental/Health/Safety:** The software has no negative impact in these areas.
- **Professional & Ethical:** The code is original and free of plagiarism.
- **Educational/Social:** It had to be easy to use for students with no prior experience in GUI or algorithms.
- **Technical Limitations:** Too many features (like saving, importing/exporting code) were avoided to maintain simplicity and performance.

2.3. Analysis and Feature finalization subject to constraints

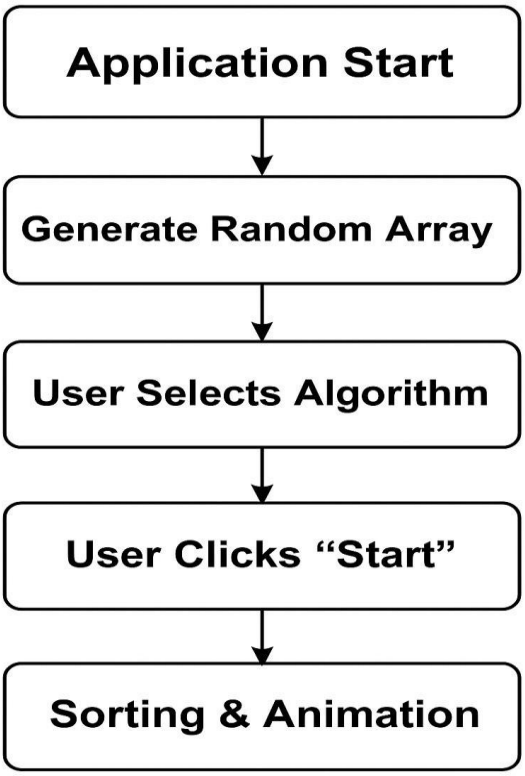
The real-time animation feature was retained, as it plays a central role in helping users visualize the sorting process step by step. Similarly, the use of multiple sorting algorithms was preserved to allow users to compare how different methods operate under the same input conditions. Some features had to be adjusted. For example, sorting speed control was not implemented to avoid adding complexity to the timing logic; instead, a fixed delay was used within the sorting functions to pace the animation evenly. Also, while displaying the pseudocode and comparison count proved highly beneficial for learning, features like saving sorted arrays or showing source code were removed to maintain a clean and distraction-free interface.

These decisions ensured that the visualizer remains lightweight, easy to understand, and focused on its core purpose: to help users visually grasp how various sorting algorithms work.

2.4. Design Flow

To build the Sorting Visualizer, two possible design approaches were considered. The first approach involved integrating the sorting logic with real-time animation updates. The second approach involved precomputing all sorting steps and animating them separately afterward.

The first design was ultimately chosen due to its simplicity and efficiency. It allowed for seamless interaction between logic and visualization, reducing memory overhead and enabling smoother performance.



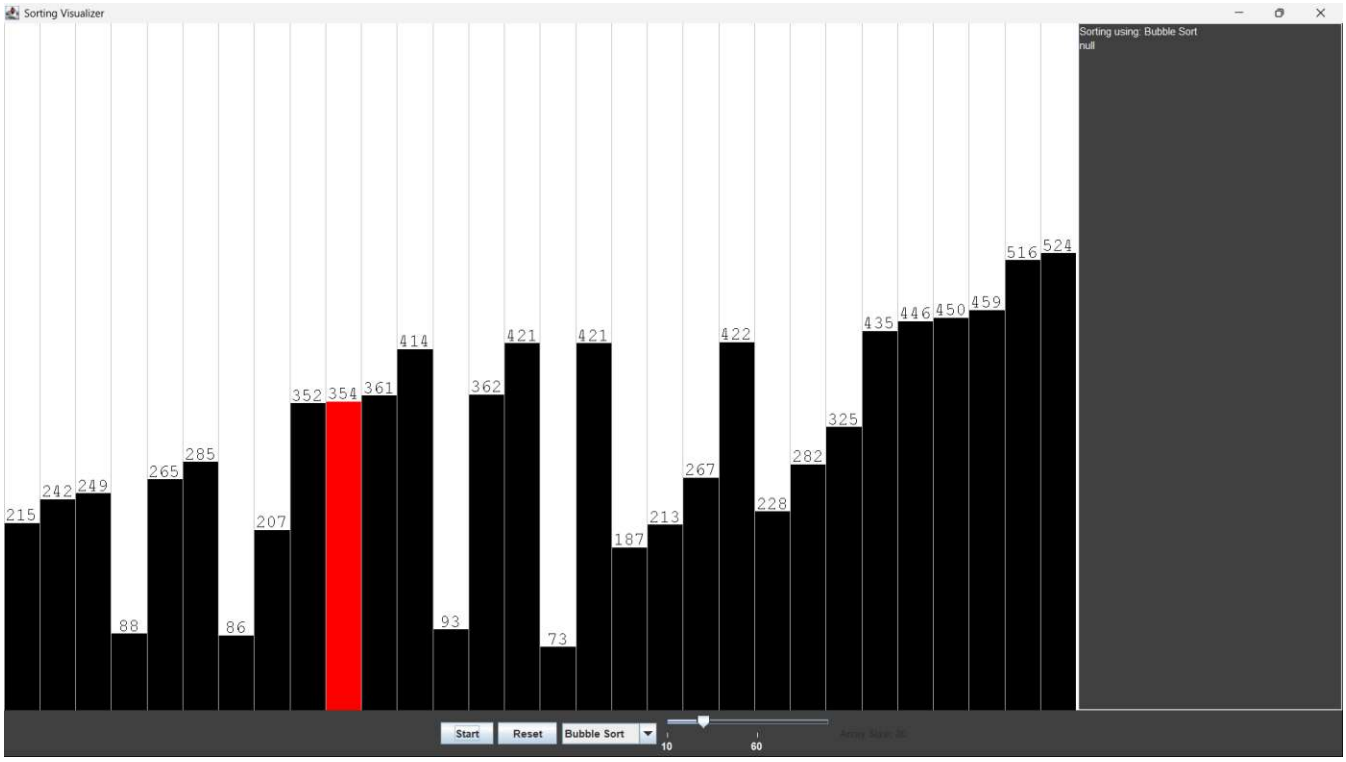
(Figure 3.1 Flowchart of design flow)

2.5. Design selection

Between the two design approaches evaluated, the first method—executing and visualizing the sorting algorithm in real time—was selected. This approach made the implementation straightforward while maintaining clarity and interactivity. It was found to be well-suited to Java Swing and allowed for easier debugging and better learning output for users.

2.6. Implementation plan/methodology

The implementation plan followed a structured process. First, the GUI was set up using Java Swing, including control elements such as buttons, sliders, dropdowns, and the visualization panel. A random array is generated based on the selected size, and the chosen algorithm is applied once the user presses the "Start" button.



(Figure 3.2 Final User interface)

During sorting, the GUI updates in real time to show each comparison and swap visually. Pseudocode is also displayed in a side panel, updating the user with relevant information and the number of comparisons performed. The entire application runs offline and in a single window, making it lightweight and focused.

CHAPTER 3.

RESULTS ANALYSIS AND VALIDATION

3.1. Implementation of solution

The development and execution of the Sorting Visualizer involved the application of modern tools and techniques across different stages of the project, including design, coding, testing, and reporting.

- **Analysis:**
  The project began with the analysis of various sorting algorithms. Their time complexities, behavior, and learning impact were studied. The goal was to choose algorithms that offer a range of complexities (e.g., $O(n^2)$, $O(n \log n)$, and $O(n + k)$) to make comparisons meaningful.
- **Design Drawings/Schematics:**
  A flowchart was designed to visualize the internal working of the application. It depicted user interaction, control flow, and the real-time sorting animation. This design flow guided the development of the graphical interface and logic integration.
- **Report Preparation:**
  All documentation, including the abstract, literature review, design flow, and analysis, was prepared using modern word processing tools such as Microsoft Word and LaTeX. Visual aids like tables, charts, and flowcharts were used to enhance understanding.
- **Project Management and Communication:**
  The development was managed in a structured timeline using a Gantt chart. Tasks such as design, development, testing, and documentation were divided week-wise. Communication with peers and faculty was maintained throughout for feedback and suggestions.
- **Testing, Characterization, and Validation:**
  The application was tested under various scenarios:
  - Different sorting algorithms (8 total)
  - Varying array sizes
  - Random datasets each time
  - Edge cases (e.g., already sorted array, reverse-sorted array)

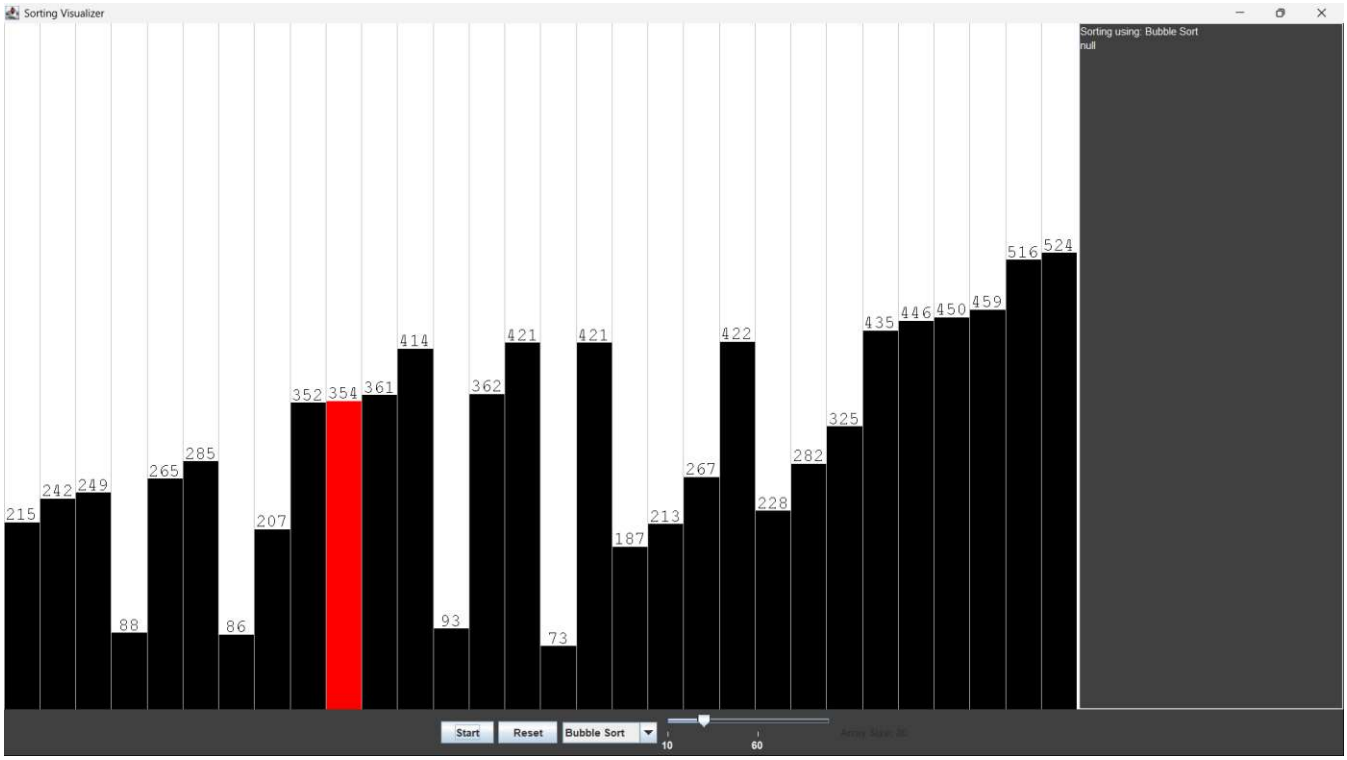
The sorting behavior was visually analyzed, and comparison counts were tracked. The results validated the correctness of each algorithm and confirmed that the visual output matched theoretical expectations.

The use of Java and Swing as the development platform enabled a stable, cross-platform desktop application, combining logic and animation within a single interface. Overall, the implementation approach ensured functionality, accuracy, and clarity in delivering the final solution.

3.2. Execution Snapshots

To evaluate the functionality of the sorting visualizer, the application was run with various algorithms and array sizes. Screenshots were taken during execution to demonstrate how the visualizer displays sorting progress in real time.

The interface successfully highlights the current element being compared or swapped using a red-colored bar. The rest of the array remains in black, and users can clearly observe the changes as the algorithm proceeds.



(Figure 4.1: Bubble Sort in progress with 30 elements)

CHAPTER 4.

CONCLUSION AND FUTURE WORK

4.1. Conclusion

The Sorting Visualizer successfully met its primary objective—to provide an interactive and educational tool for visualizing how various sorting algorithms work. Through this project, eight popular algorithms were implemented: Bubble Sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Counting Sort, Radix Sort, and Bucket Sort.

The application displayed each algorithm's operation in real-time using animated vertical bars. Additional features such as dynamic array size selection, pseudocode display, and comparison count further enhanced the learning experience. The tool was built entirely in Java using the Swing library, ensuring platform independence and GUI simplicity.

The expected outcome was a smooth, user-friendly visualization tool. Most features worked as intended. However, a minor deviation was observed in the lack of speed control or pause/resume functionality—features that were initially considered but excluded for simplicity. Despite this, the overall user feedback and performance aligned with the project goals.

4.2. Future work

While the current version of the Sorting Visualizer performs well and fulfills its educational purpose, there is potential for future enhancement:

- **Speed Control:** Introducing a slider to adjust the animation speed would allow users to slow down or speed up the sorting for better understanding.
- **Pause/Resume Feature:** Enabling pause and step-by-step navigation through the sorting process could help in analyzing specific parts of the algorithm in detail.
- **More Algorithms:** Advanced sorting algorithms like Heap Sort, Bucket Sort, or Shell Sort could be added to broaden the range of comparisons.
- **Mobile or Web Version:** Porting the application to web or mobile platforms (using JavaScript or Flutter) would improve accessibility.
- **Dark/Light Mode and Themes:** Adding visual themes could improve the overall user interface and accessibility experience.

These improvements would make the visualizer more interactive, customizable, and suitable for a broader audience, including educators, students, and developers exploring algorithm behavior.

REFERENCES

[1] VisuAlgo – Visualising data structures and algorithms through animation, [Online]. Available: https://visualgo.net/en

[2] D. Galles, “Sorting Algorithm Animations,” University of San Francisco, [Online]. Available: http://www.cs.usfca.edu/~galles/visualization/Algorithms.html

[3] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: The MIT Press, 2009.

[4] GeeksforGeeks, “Sorting Algorithms,” [Online]. Available: https://www.geeksforgeeks.org/sorting-algorithms/

[5] Oracle, “The Java™ Tutorials: Creating a GUI With Swing,” [Online]. Available: https://docs.oracle.com/javase/tutorial/uiswing/

[6] W. Savitch, *Absolute Java*, 6th ed., Pearson Education, 2015.