Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
"Real-Time Sorting Algorithm Visualizer"

[Code No : COMP 202]
(For partial fulfillment of II/I Year/Semester in Computer Science/Engineering)

Submitted by
Sanchit Baral (Roll No. 03)

Submitted to
Er. Sagar Acharya
Department of Computer Science and Engineering

Submission Date: 25/02/2026

Acknowledgements

I would like to express my sincere gratitude to everyone who contributed to the successful completion of this mini project, “Real-Time Sorting Algorithm Visualizer”.

First and foremost, I extend my deepest thanks to my project supervisor, Er. Sagar Acharya, from the Department of Computer Science and Engineering, Kathmandu University. His guidance on data structures and algorithm design was very helpful throughout the project. His encouragement to think visually about algorithms helped me understand sorting, graphics programming, and real-time rendering much better.

I am also thankful to the Department of Computer Science and Engineering, Kathmandu University, for providing the necessary resources and a good learning environment that made this project possible.

Finally, I would like to thank my family and friends for their constant support and motivation throughout the duration of this project.

Abstract

This mini project was developed to improve understanding of sorting algorithms through visual animation. This project presents a Real-Time Sorting Algorithm Visualizer that uses the concepts of Data Structures and Algorithms (DSA) to make sorting easy to understand. The system displays an array of colored bars that are sorted step by step on screen. It uses six algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. Each comparison is highlighted in yellow and each swap in red so users can clearly see what is happening. By combining real-time rendering with practical data structures, the project shows how DSA concepts can be turned into a working and useful tool.

Keywords: *Sorting Algorithms, Algorithm Visualization, Raylib, C++17, Data Structures, Real-Time Rendering, Bubble Sort, Merge Sort, Quick Sort, Heap Sort, Step Engine*

Table of Contents

List of Figures

Acronyms/Abbreviations

DSA: Data Structures and Algorithms

FPS: Frames Per Second

GUI: Graphical User Interface

CPU: Central Processing Unit

RAM: Random Access Memory

IDE: Integrated Development Environment

UI: User Interface

API: Application Programming Interface

Chapter 1 Introduction

This system “Real-Time Sorting Algorithm Visualizer” is a simple desktop application that helps users understand how sorting algorithms work by showing them step by step on screen. It displays a row of colored bars that get sorted in real time. Each step is highlighted so the user can see exactly what the algorithm is doing at every moment. The project applies fundamental concepts of Data Structures and Algorithms (DSA) to build a tool that makes learning sorting algorithms easier and more interesting.

1. Background

Sorting is one of the most important topics in computer science. Students often find it hard to understand sorting algorithms by just reading code. The problem is that code shows only the final result and not the process. Traditional textbook diagrams also show only a few steps and miss a lot of detail. This makes it difficult to understand why one algorithm is faster than another.

Visualization tools solve this problem by showing every single step in real time. When a student can see comparisons and swaps happening live, they build a much stronger understanding of the algorithm. DSA concepts like arrays, functions, and step-by-step logic are the foundation of this project and have been used to build a smooth and accurate visualizer.

2. Objectives

- To understand and implement six classic sorting algorithms and show each step in real time.
- To design a color system that makes comparisons, swaps, and sorted bars clearly different.

- To track and display live statistics such as comparison count, swap count, and progress.
- To let the user change speed, array size, and algorithm using keyboard controls.

3. Motivation and Significance

The main motivation behind this project was to apply what I have learned in class to something that actually helps people learn better. As a student myself, I know that reading about Bubble Sort or Quick Sort from a textbook is often confusing. The idea behind this project was to make those algorithms visible. When you can see every comparison and swap happening on screen, the concept becomes much clearer. Using arrays to store bar values and function objects to store steps, the project shows how classical data structures can be used to create a real and useful educational tool.

Chapter 2 Related Works

Various sorting visualizer tools have been developed over the years and have helped students learn algorithms better. Some of the most well-known ones are listed below.

- **VisuAlgo (Web-Based Sorting Visualizer)**

VisuAlgo is a web-based tool built using JavaScript and HTML5 Canvas that animates sorting algorithms and other DSA topics. It was developed at the National University of Singapore and is widely used in universities across the world. The tool supports algorithms like Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. It shows each step one at a time with a clear description of what is happening. However, the tool runs only in a web browser and depends on an internet connection. It also does not give the user control over array size or playback speed in a very fine way. Users cannot change the number of elements freely, and the interface has a limited set of controls. Despite these drawbacks, VisuAlgo is one of the best free educational resources for learning DSA visually (Halim, 2013).

- **Sorting.at (Interactive Sorting Comparison Tool)**

Sorting.at is another browser-based sorting visualizer that allows users to watch multiple sorting algorithms running at the same time for comparison. It is built with JavaScript and runs directly in the browser without any installation. The tool is useful for comparing how different algorithms behave on the same dataset. However, it has a fixed array size and does not show detailed statistics like comparison count or swap count. The interface is clean but gives very little control to the user. It is also not available offline. For students who want to study algorithm performance differences, this tool provides a good visual comparison but lacks the depth needed for a full learning experience (Bednar, 2014).

- **SDL-Based Native Sorting Visualizers**

Several desktop-based sorting visualizers have been built using SDL (Simple DirectMedia Layer), a C library for graphics and input. These tools run natively on the computer without needing a browser and typically perform better than web-based tools. They can handle larger arrays and smoother animations. However, SDL requires a more complex build setup compared to Raylib. Users need to configure include paths, link libraries, and manage the window and event loop manually. This makes it harder for beginners to get started. The resulting visualizers also tend to have simpler interfaces without stat cards or progress bars. This project chose Raylib instead because it provides a much simpler API while still delivering hardware-accelerated 2D rendering at smooth frame rates (Lantinga, 1998).

Chapter 3 Design and Implementation

The design and implementation of this project was planned carefully so that every part was clean, easy to understand, and easy to extend. The system was built in six stages, with each stage adding new features on top of the previous one.

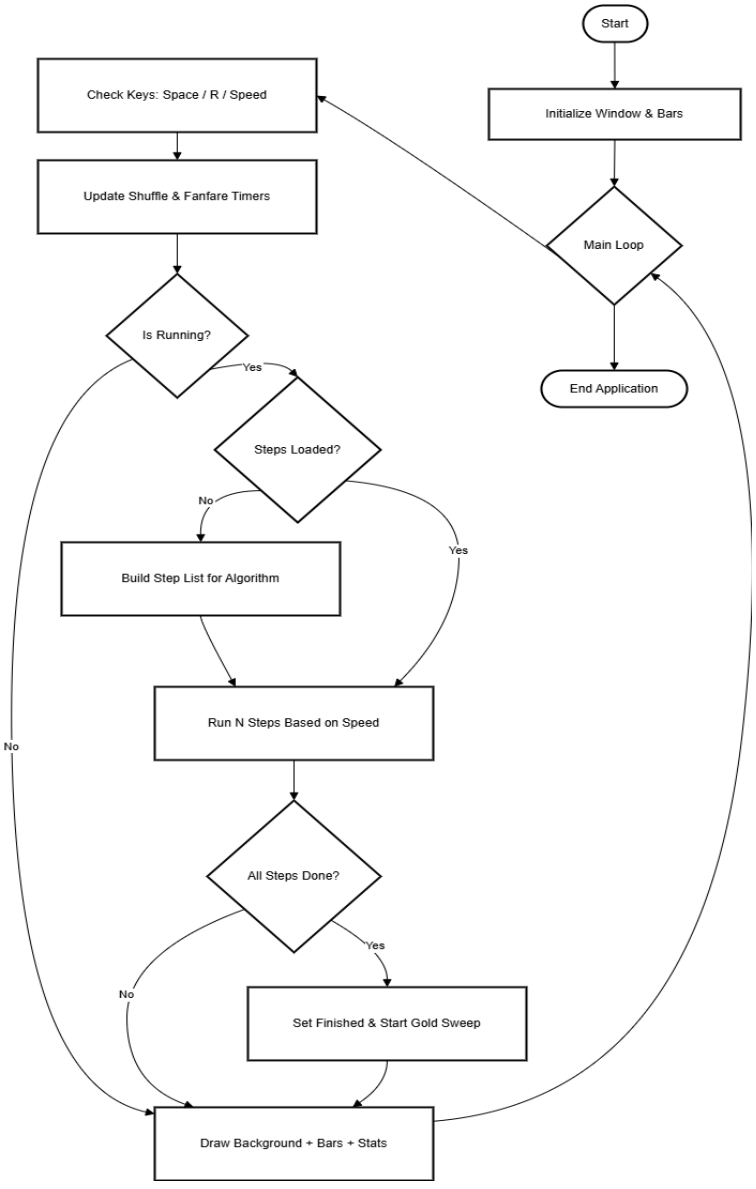

Fig 3.1: System Architecture Diagram

3.1 System Requirement Specifications

This system was developed using C++17 and the Raylib graphics library. It runs on a standard desktop computer without needing any internet connection. The system provides a real-time visualization of six sorting algorithms and allows the user to interact using simple keyboard controls.

3.3.1 Software Specifications

The following points describe the software requirements of this system.

- **Functional Requirements:**
    a. **Real-Time Sorting Display:** The system shall animate each step of sorting on screen with colored bars.
    b. **Color Coding:** The system shall highlight comparisons in yellow and swaps in red at every step.
    c. **Algorithm Selection:** The system shall allow the user to choose any of the six sorting algorithms using number keys.
    d. **Live Statistics:** The system shall display comparison count, swap count, step number, and element count in real time.
    e. **User Controls:** The system shall let the user start, pause, shuffle, change speed, and change array size using the keyboard.

- **Non-Functional Requirements:**
    f. **Performance:** The system shall run at a smooth 60 frames per second for all array sizes up to 200 elements.
    g. **Reliability:** The system shall complete sorting correctly for all six algorithms without crashing.
    h. **Usability:** The system shall provide a clear and easy-to-read interface with labels for every control.

i. **Scalability:** The system shall support array sizes from 25 to 200 elements without changing any core logic.
j. **Portability:** The system shall run on any desktop that supports Raylib and a C++17 compiler.

3.1.2 Hardware Specifications

The following hardware components are needed for the system to run well.

- **Processor:** Any dual-core or better CPU (e.g., Intel i3 or equivalent) to handle rendering and step execution.
- **Memory (RAM):** At least 2 GB RAM to run the application and hold the step vector in memory.
- **Storage:** At least 100 MB of free disk space for the compiled binary and Raylib library files.
- **Display:** A monitor with at least 1600 x 900 resolution to see the full interface correctly.
- **Input Devices:** A standard keyboard for all user controls.

Chapter 4 Discussion on the Achievements

The sorting visualizer was built to make algorithm learning easier and more visual. The step-engine design kept the sorting logic clean and separate from the rendering code. All six algorithms work correctly and animate smoothly. Future versions could explore larger array sizes with lazy step generation. The project shows how a visual tool can make abstract DSA concepts much easier to understand. The Real-Time Sorting Algorithm Visualizer achieved all of its main goals: correct algorithm animation, a clean and easy-to-use interface, and useful live statistics.

4.1 Features:

1. Six Sorting Algorithms

The system includes Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. All six are pre-built and ready to run. The user can switch between them instantly using number keys 1 through 6. Each algorithm uses the same step-engine pattern so they all work in the same consistent way.

2. Real-Time Color-Coded Animation

Bars are shown in four colors: blue for unsorted, yellow for bars being compared, red for bars being swapped, and green for bars that are already in their final sorted position. This color system makes it immediately clear what the algorithm is doing at every step. Even a first-time learner can understand the animation without any explanation.

3. Live Statistics Panel

The stats row at the top shows four cards in real time: comparisons made, swaps made, current step out of total steps, and the number of elements in the array. There is also a progress bar showing how far through the sort the algorithm is, and a speed

indicator that changes color from green to red as the speed increases. These numbers help users understand the cost of each algorithm clearly.

4. User Controls

The user can press SPACE to start or pause sorting, R to shuffle and reset the array, UP and DOWN arrows to change the speed, and A or D to decrease or increase the array size. All controls are shown permanently in the header so the user never needs to guess. The algorithm buttons at the top highlight the currently active algorithm so the selection is always clear.

5. Animations

When the user shuffles the array, a wave animation makes each bar pop in from left to right with an ease-out curve. When sorting is complete, a gold highlight sweeps across all the bars to show that the array is fully sorted. These small animations make the experience more satisfying and give clear visual feedback about what state the visualizer is in.

Chapter 5 Conclusion and Recommendation

The system “Real-Time Sorting Algorithm Visualizer” built as a mini project to apply the knowledge of DSA was able to achieve its primary goal of designing and implementing a step-by-step sorting animation system using C++17 and Raylib. The system was built to show six sorting algorithms running live, with colors that clearly mark every comparison and swap. The step-engine design kept the algorithm logic separate from the rendering code, making the project easy to maintain and extend. The live statistics panel lets users directly measure the cost of each algorithm in terms of comparisons and swaps, reinforcing key DSA lessons.

5.1 Limitations

- The system slows down the startup when handling very large arrays because all steps are generated at once before playback begins.
- The system only supports integer arrays and cannot sort strings or custom data types.
- There is no way to save or replay a sort session after it has finished.
- The visualizer requires a 1600 x 900 screen and does not scale to smaller displays.

5.2 Future Enhancement

- Add a lazy step generation mode so that very large arrays above 200 elements can be handled without a startup delay.
- Add a side-by-side comparison mode that runs two algorithms at the same time on the same data.
- Support user-defined input so the user can type in their own array values to sort.
- Add more algorithms such as Shell Sort, Radix Sort, and Tim Sort.

References

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press. (Chapter 6: Heapsort; Chapter 7: Quicksort; Chapter 8: Sorting in Linear Time).

Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.). Addison-Wesley.

Raysan5. (2013–2024). Raylib: A simple and easy-to-use library to enjoy videogames programming. Retrieved from https://www.raylib.com

Data Structures and Algorithm Lecture Notes, Kathmandu University.

APPENDIX

Fig 5.1: Unsorted Array View

Fig 5.2: Sorting in Progress

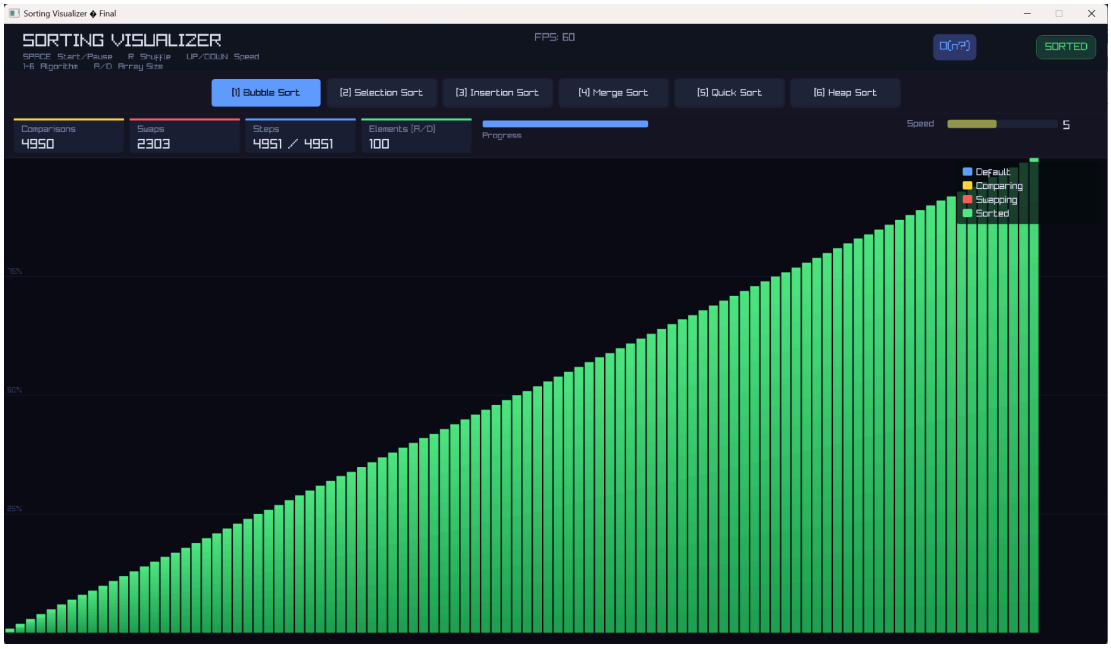

Fig 5.3: Sorted Array View