Interactive Sorting Algorithm Visualizer

Program: BS Software Engineering

Semester:: 3rd - Regular

Course: Data Structures & Algorithms

Team Members:

17 - Sawera Mumtaz (Group Leader)

02 - Hanzla Rani

03 - Shahmeer Hassan

42 - Alishba

50 - Huzaifah Iftikhar

---

1. Project Overview

Problem Statement

Sorting algorithms are fundamental to computer science, yet understanding their internal mechanics —specifically the difference between comparison-based strategies and non-comparison based efficiency—is difficult to grasp through static code. Students often struggle to visualize how memory is manipulated in algorithms like Merge Sort or how time complexity scales with input size.

Purpose

The **Interactive Sorting Algorithm Visualizer** is a robust educational tool designed to visualize the step-by-step execution of 9 distinct sorting algorithms. By utilizing a high-performance C++ backend for execution and a React frontend for rendering, the system provides accurate, real-time performance metrics and visualizes datasets ranging from small arrays to massive, scrollable datasets.

2. System Architecture

The system utilizes a **Decoupled Architecture** designed for high data throughput and memory efficiency.

2.1 Core Components

Backend (C++17): The computational engine. It generates datasets, executes algorithms, and serializes the execution history. It features a custom **Delta Compression** engine to minimize memory usage during logging.
Frontend (React/TypeScript): The visualization layer. It parses the compressed logs and uses a **Scalable Canvas Renderer** to draw frame-by-frame animations.
Data Interchange (JSON): Acts as the interface between the producer (C++) and consumer (Web).

2.2 Data Flow Pipeline

```
1. Generation: User selects Algorithm & Size (CLI) | 2. Execution: C++
Engine runs sort | 3. Optimization: StepRecorder calculates 'Diffs'
(Only changed indices recorded, not full array) | 4. Serialization:
Compressed JSON log generated | 5. Reconstruction: Frontend loads JSON |
6. Rendering: Canvas applies Diffs to local state & renders
```

3. Algorithmic Implementation

The project implements a comprehensive suite of algorithms, categorized by their sorting strategy. All implementations are instrumented to record Comparisons, Swaps, and Time Complexity.

Comparison-Based Algorithms

Algorithm	Complexity	Visualization Mechanic
Bubble Sort	$O(n^2)$	Standard adjacent swapping. Optimized with an early-exit flag.
Selection Sort	$O(n^2)$	Highlights the linear scan for the minimum element (Amber) before swapping.
Insertion Sort	$O(n^2)$	Visualizes the "shift and overwrite" mechanism for the sorted subarray.
Shell Sort	O(n log n)	Visualizes gap-based insertion, showing how elements move long distances initially.
Merge Sort	O(n log n)	Recursive Divide & Conquer. Visualizes the merging of auxiliary arrays.
Quick Sort	O(n log n)	Lomuto Partition scheme. Explicitly highlights the **Pivot** (Violet) and partition boundaries.
Heap Sort	O(n log n)	Visualizes the Max-Heap structure and the "sift-down" process.

Non-Comparison Algorithms (New)

Algorithm	Complexity	Visualization Mechanic
Radix Sort (LSD)	O(nk)	Visualizes distribution into buckets based on digit position (units, tens, etc.).
Bucket Sort	O(n + k)	Visualizes scattering elements into ranges and gathering them back.

4. Technical Optimization & Performance

> **Feature 1: Delta Compression Protocol**
> To support large datasets (N > 1,000), the backend does not save the full array for every frame. Instead, the `StepRecorder` logs only the *indices that changed* (e.g., `"op": "swap", "indices": [4, 5]`). This reduces the output file size by approximately **90%** compared to full-state logging, allowing for massive logs to be generated and loaded instantly.

> **Feature 2: Scalable Canvas Rendering**
> The frontend implementation handles high-density arrays.
> **Dynamic Width Calculation:** Bar width is calculated as floating-point to ensure sub-pixel rendering accuracy on high-DPI displays.
> **Scroll Support:** For extremely large datasets (N > 2,000), the canvas enables a horizontal scroll mode, preventing bars from becoming too thin to see.

5. Conclusion

This project fulfills the requirements of an advanced educational tool. By implementing both standard comparison sorts and complex non-comparison sorts, and by optimizing the data pipeline with delta compression, the system demonstrates high technical proficiency. It allows users to gain a deep intuition for algorithmic efficiency through interactive exploration.