DAA Algorithm Project

Session 2024 - 2025

Sorting Algorithm Visualizer

Deployed project link

Submitted by:

Name: **Yash Raj Singh**

Roll No: **2205606**

Section: **CSE 32**

Submitted to:

Dr. Sricheta Parui

Course:

Design and Analysis of Algorithms

Abstract

This project provides an interactive, web-based visualization of two widely studied sorting algorithms: Merge Sort and Bubble Sort. Designed with the intent of engaging and educating users, the project uses React to showcase real-time animations of these algorithms, illustrating their step-by-step execution. The visualization includes features such as sliders to adjust both the array size and animation speed, allowing users to observe the impact of these algorithms on datasets of varying sizes and complexity. Through these tools, users gain a deeper understanding of how sorting algorithms function, emphasizing the importance of algorithmic efficiency in computational tasks. This project highlights the advantages of educational tools that leverage interactivity to simplify complex computer science concepts, and it is poised for future expansion with plans to add path-finding and tree traversal algorithms to further extend its educational value.

Introduction to the Topic

Sorting algorithms form a cornerstone of computer science, enabling efficient data handling in fields as diverse as database management, network routing, and machine learning. The goal of this project is to provide a web-based platform that visualizes **Merge Sort** and **Bubble Sort**, demonstrating key differences between a simple, iterative sorting method and a recursive, divide-and-conquer approach. **Bubble Sort** operates by iteratively comparing and swapping adjacent elements, yielding O(n^2) time complexity, which becomes increasingly inefficient as data size grows. Conversely, **Merge Sort** recursively divides the array into smaller sub-arrays, sorting and merging them back together efficiently, achieving O(n log n) complexity.

React's component-based framework was selected to bring this project to life, offering dynamic state management and high responsiveness to user inputs. Users can adjust the data-set size and sorting speed through sliders, observe color-coded comparisons and swaps, and reset arrays to try different data sets. This interactive format allows learners to see algorithms in action, fostering an intuitive grasp of their inner workings and comparative performance. The project emphasizes the role of interactivity in learning and the potential for future expansions, including additional sorting algorithms, path-finding algorithms, and side-by-side comparisons to further enhance its educational impact.

Literature Review

Merge Sort, credited to John von Neumann in 1945, is one of the earliest divide-and-conquer sorting algorithms and continues to be widely used in situations requiring stable, predictable performance (Lowry et al., 1951).

With a time complexity of O(n log n), Merge Sort is theoretically faster for large datasets compared to algorithms with quadratic time complexity, such as Bubble Sort (Storer, 2002).

However, its memory requirements may not always be suitable for constrained environments, highlighting the need for adaptability in algorithm selection.

Bubble Sort, by contrast, is often dismissed in practical applications due to its quadratic time complexity but remains foundational for understanding sorting logic due to its straightforward comparison and swap mechanism.

Research increasingly shows that interactive visualizations can bridge the gap between abstract algorithmic concepts and practical understanding.(Algorithm visualization: a report on the state of the field, n.d)

This project builds on this premise by providing users with real-time control over algorithm parameters, showcasing React's interactive capabilities and reinforcing the principles behind both sorting algorithms.

Methodology

This project was developed using **React**, which allows for a responsive, interactive web application structured around independent components. The application consists of several key components that work together to manage array generation, user interaction, and algorithm visualization.

- **User Interaction Components**:

Element Slider: Users can adjust the number of elements in the array, allowing them to test the sorting algorithms on different dataset sizes.

Speed Slider: Users can control the animation speed, slowing down the process for detailed analysis or speeding it up for faster comprehension of the overall sorting approach.

Randomize Button: A reset button regenerates a randomized array of values, giving users a new dataset for each sorting test.

Algorithm Selection Buttons: Users can select either Merge Sort or Bubble Sort, initiating a color-coded animation of the chosen algorithm.

- **Visualization and Sorting Components**:

Sorting Logic: Merge Sort and Bubble Sort algorithms are implemented with interactive animations to visualize the logic behind each comparison and swap.

React State Management: State variables handle array values, animation timing, and user-selected settings. **useState** and **useEffect** allow for controlled re-rendering and responsiveness to changes.

Optimization with useMemo: The **useMemo** hook is applied to memoize values that do not change frequently, enhancing performance by preventing unnecessary recalculations during re-renders.

- **Styling and User Experience**:

Color Coding: The application highlights elements under comparison in red, reverting to their original color once sorted, making it easy to follow the sorting process visually.

Component Reusability: By structuring the application into reusable components, the project remains modular and easy to expand, facilitating future feature additions, such as adding new algorithms or visual elements.

React’s flexibility and reactivity make it an ideal framework for building this project. The resulting application provides a user-friendly interface that balances educational value with intuitive controls, allowing users to explore algorithm behavior dynamically and adjust parameters in real time.

Results and Analysis

The project successfully demonstrates the functionality and comparative efficiency of Merge Sort and Bubble Sort, providing valuable insights into each algorithm's performance on datasets of varying sizes. Bubble Sort, due to its $O(n^2)$ complexity, requires significantly more steps to sort large datasets, and this is evident when users increase the dataset size or slow down the animation speed. Users can see each individual swap and comparison, offering a clear view of why Bubble Sort is rarely suitable for large-scale applications.

Conversely, Merge Sort’s divide-and-conquer approach handles larger datasets much more efficiently. The visualization showcases Merge Sort’s recursive nature, with the array being split into progressively smaller subarrays until sorted and merged back together. This sorting strategy becomes particularly apparent when users speed up the sorting process, making Merge Sort’s speed advantages obvious.

Feedback from initial users indicates that this interactive approach significantly enhances comprehension of algorithm mechanics and complexity. The color-coded animations and adjustable parameters make it easy to observe both algorithms' performance, offering valuable insights into the practical importance of algorithm selection in real-world scenarios.

Conclusion and Future Work

The project has proven effective in making sorting algorithms accessible and engaging through a visually appealing, interactive format. The site's ability to dynamically adjust dataset sizes and animation speeds enables users to explore sorting mechanics in a way that traditional explanations often fail to achieve. By focusing on React's reactivity and state management capabilities, the project delivers a robust educational tool that could be expanded in numerous ways.

Future Enhancements:

Additional Algorithms: Pathfinding and tree algorithms will be incorporated to broaden the site's educational scope, allowing users to explore a wider range of fundamental algorithms.

Side-by-Side Comparisons: A new page will be added for comparing two algorithms in real time, using identical datasets to highlight the differences in time complexity and sorting approaches. Complexity metrics will be displayed alongside each algorithm, reinforcing the relationship between algorithm design and performance.

Performance Analytics: Integrating additional performance metrics, such as memory usage and time complexity, will provide users with a deeper understanding of each algorithm's computational costs and efficiency.

This project lays a strong foundation for interactive algorithm education, and with the planned expansions, it aims to become a comprehensive platform for exploring algorithm behavior and complexity in a visually intuitive manner.