Project Abstract: Algorithm Playground

An Interactive Visualizer for Algorithm Design and Analysis

Samjith Raj Bondla (23891A66B4)
Samerla Deekshith (23891A66B3)
Gangi Manognya (23891A6675)
Vanam Sahaja (23891A66C6)

Department of CSE(AI&ML)
Vignan Institute of Technology and Science

October 22, 2025

Abstract

The "Algorithm Playground" is an interactive web application developed for the Design and Analysis of Algorithms (DAA) innovative project. Its primary objective is to demystify complex algorithms by providing a dynamic, visual, and educational platform where users can observe their step-by-step execution and understand their performance characteristics.

This single-page application, built entirely with HTML, Tailwind CSS, and modern JavaScript, features several key modules:

- **Diverse Algorithm Categories:** The tool provides visualizations for a wide range of algorithms, including Sorting (e.g., Bubble Sort, Merge Sort, Quick Sort), Searching (Linear, Binary), Pathfinding (e.g., Dijkstra's, A*, BFS, DFS), and Dynamic Programming (e.g., Fibonacci).
- **Interactive Visualizer:** Users can generate random data, control visualization speed, and, in the case of pathfinding, interact with the grid by drawing custom "wall" barriers and moving the start/end nodes.
- **Dynamic Complexity Analysis:** A core feature is the real-time analysis panel. As an algorithm is selected, this panel immediately updates to display its Best, Average, and Worst-case Time Complexity, as well as its Space Complexity, complete with a clear, concise explanation for each.

This project serves as a practical, hands-on learning tool. It successfully bridges the gap between theoretical algorithmic concepts and their practical, visual implementation, making complex topics in Design and Analysis of Algorithms more accessible and intuitive to understand.

Project Overview and Design

1. System Architecture

The application is intentionally designed as a single-page application (SPA) to ensure portability, zero-setup, and ease of use. It runs entirely in the client's web browser, requiring no backend server or dependencies. The architecture is modular, even within the single-file structure:

- **View Management:** A core JavaScript function handles switching between the four main "views" (Sorting, Searching, Pathfinding, DP). This function dynamically hides and shows the relevant control panels and visualizer containers.
- **State Management:** A global JavaScript object holds the application's state, including the current algorithm, visualization speed, array/grid data, and boolean flags (e.g., 'isSorting').
- **Algorithm Modules:** Each algorithm is implemented as a separate 'async' JavaScript function (e.g., 'bubbleSort()', 'dijkstra()'). This allows the use of 'await sleep()' for visualization delays without blocking the main browser thread.
- **DOM Manipulation:** Helper functions (e.g., 'generateArray()', 'createGrid()') are responsible for creating, updating, and styling the HTML elements (bars and grid cells) that form the visualization.

System Architecture Diagram Placeholder

This diagram would show the relationship between the UI, State Management, Visualization Engine, and Algorithm Modules.

Figure 1: High-level architecture of the Algorithm Playground.

2. Implementation Details

User Interface (UI): The UI is built with HTML and styled using **Tailwind CSS**. Tailwind’s utility-first approach allows for rapid development of a clean, modern, and responsive interface without writing custom CSS files. A tabbed navigation system provides a clear and intuitive way to switch between algorithm categories.

Visualization Logic:

- **Sorting/Searching:** Visualized using ‘div‘ elements (”bars”). The ‘height‘ property of each bar is dynamically updated to reflect its value in the array. Colors are changed by adding/removing Tailwind classes (e.g., ‘bg-red-500‘, ‘bg-green-500‘) to highlight comparisons, swaps, and sorted elements.

- **Pathfinding:** A CSS Grid represents the node map. Each cell is a ‘div‘ that can have classes like ‘.wall‘, ‘.visited‘, and ‘.path‘ added to it, which apply different background colors and animations.

Complexity Analysis: A key component is the ‘complexityData‘ JavaScript object. This object acts as a database, mapping algorithm IDs (e.g., ‘bubbleSort‘) to their title, time/space complexity, and a detailed HTML string for the explanation. A single function, ‘updateComplexityInfo()‘, reads from this object based on the currently selected algorithm and injects the correct data into the analysis panel.

Algorithm Execution Flowchart Placeholder

This flowchart would illustrate the process: User selects algorithm → Clicks 'Start' → Controls are disabled → Async algorithm function is called → Loop (Update array, Update DOM, `await sleep()`) → Controls are enabled.

Figure 2: Flow of control for a typical visualization.