iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 13 **Issue:** XI **Month of publication:** November 2025

**DOI:** https://doi.org/10.22214/ijraset.2025.75446

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue XI Nov 2025- Available at www.ijraset.com*

# Code and Algorithm Visualizer

Divya Dharsni L M[1], Fathima Zuhra A B[2], Mrs. A. Lavanya[3]

[1, 2]*Bachelor of Engineering in Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution)*
[3]*Assistant Professor, Department of CSE, Adhiyamaan College of Engineering (An Autonomous Institution), Anna University, Chennai*

***Abstract:*** ***In the field of computer science education, understanding algorithms and data structures poses a significant challenge for many learners due to their abstract and logical nature. The Code Visualizer project aims to bridge this gap by developing an interactive algorithm visualization platform that allows students and beginners to write, execute and visualize algorithms through step-by-step, real-time animations. The system provides a dynamic learning environment where users can observe how algorithms process data internally, such as during sorting, searching and graph traversal operations. By translating algorithmic logic into visual representations, the platform enhances conceptual clarity and fosters deeper comprehension. The integration of automatic algorithm detection enables the system to recognize code patterns and generate contextual visualizations along with explanations and complexity analysis. To make the learning experience engaging and effective, the application incorporates interactive controls—allowing users to pause, replay or adjust execution speed—and educational overlays that provide theoretical insights. The platform's responsive and modular design, built using Flutter and integrated with Firebase, ensures accessibility across devices, real-time data handling and scalability for future algorithm additions.***

***Beyond individual learning, the Code Visualizer also supports educators through tools that facilitate custom demonstrations, assignment sand performance tracking. By transforming complex algorithmic logic into intuitive visual feedback, this project empowers learners to connect theoretical knowledge with practical understanding. Ultimately, the Code Visualizer serves as a modern educational tool that promotes active learning, algorithmic thinking and computational problem-solving skills for the next generation of programmers.***

## I. INTRODUCTION

### *A. Overview*

Our project introduces an interactive algorithm visualization platform designed for computer science students and programming beginners. This comprehensive application enables users to write, execute and visualize algorithms through dynamic, step-by-step animations, transforming abstract concepts into intuitive learning experiences.

Understanding algorithms can be challenging due to their abstract and logical nature. This platform bridges that gap by providing real-time visual feedback that reveals how algorithms operate internally. Users can input code, observe each stage of execution and follow algorithmic decision-making through color-coded animations that highlight comparisons, swaps, node visits and variable updates. The platform supports a wide range of core algorithms-including sorting, searching, graph traversal and recursion-allowing learners to see how data structures evolve during execution. Interactive controls let users navigate through algorithm steps, pause, rewind or adjust speed, promoting deep comprehension and self-paced exploration.

A key innovation is the automatic algorithm detection system, which identifies common code patterns and generates appropriate visualizations automatically. This intelligent feature acts as a virtual teaching assistant, offering contextual explanations, complexity insights and conceptual guidance tailored to each algorithm. Integrated educational resources such as theoretical overviews, time and space complexity analysis and best practices-support both independent learning and classroom instruction.

The platform's responsive and modular architecture ensures smooth performance across devices and easy scalability for new algorithms. Through intuitive interaction and seamless user experience, the platform fosters both engagement and understanding. By turning abstract algorithmic logic into concrete visual experiences, it empowers learners to bridge theory and practice, cultivating deeper algorithmic thinking and inspiring the next generation of programmers.

### *B. Objective*

The main objectives of this project are:

*1)* To enhance the understanding of algorithmic concepts by providing an interactive visualization platform that enables students to write, execute and observe algorithms through real-time, step-by-step animations, making abstract ideas more concrete and easier to grasp.

2) To bridge the gap between theory and practice by allowing learners to visualize the internal workings of algorithms, helping them develop a deeper conceptual understanding, improved analytical thinking and stronger problem-solving skills.
3) To create an engaging, accessible and user-friendly learning environment through intuitive visual interfaces, responsive controls and integrated learning materials, supporting both self-paced learning and classroom-based instruction.
4) To incorporate intelligent assistance and adaptive learning features that can automatically detect algorithm types, provide contextual explanations, highlight errors and perform performance analyses, thereby functioning as a virtual tutor to guide learners.
5) To empower educators by offering customization tools for designing interactive demonstrations, lessons and assignments, enabling a more dynamic and effective teaching experience in computer science education.
6) To promote inclusive and continuous learning through features such as multi-language support, offline functionality and personalized progress tracking, ensuring the platform is accessible and beneficial to learners from diverse educational and linguistic backgrounds.
7) To encourage curiosity and lifelong learning by fostering an interactive, experimental and feedback-driven approach, motivating students to explore algorithms beyond textbooks and classroom boundaries.

## II. LITERATURE SURVEY

The literature survey reviews recent studies related to algorithm visualization, interactive code execution and educational technologies aimed at improving programming comprehension.

1) R. Pang and X. Lu, "AI-Driven Algorithm Visualization Tool for Python Learning," IEEE Transactions on Learning Technologies, 2025.

This paper developed an AI-based visualization system that converts Python code into dynamic, interactive visuals. The tool enhances conceptual clarity by showing variable and data structure changes in real-time.

2) Vinueza-Morales et al., "Visualization-Based Programming Education: A Bibliometric Study," Springer Education Informatics, 2025.

This study analyzed global research trends in programming education, revealing that visualization tools significantly improve learner engagement, participation, and understanding of algorithmic logic.

3) S. K. Sharma and N. Gupta, "Interactive Data Structure Visualization Environments for Teaching Algorithms," Journal of Educational Computing Research, 2025.

The authors proposed a dynamic visualization platform for teaching sorting and searching algorithms. Their results show that interactive environments enhance learning outcomes compared to static or text-based instruction.

4) Yan Zhang, "Enhancing Student Understanding of Data Structures through Algorithm Animation," Elsevier Computers & Education, 2024.

This study demonstrated that visualizing algorithm steps improves students conceptual understanding data structure operations such as stacks, queues, and trees.

5) Varun Singh et al., "A Review of Visualization Techniques in Algorithm Education," ACM Computing Surveys, 2024.

The authors reviewed multiple algorithm visualization approaches, concluding that visual interaction, adaptability, and accessibility are essential features of effective educational tools.

6) Smriti Chawla and Sneha Jindal, "Bridging Theory and Practice through Real-Time Algorithm Visualization," International Journal of Computer Science Education, 2024.

This paper introduced a web-based tool that synchronizes code execution with real-time animation, helping students understand how theoretical algorithms translate into executable logic.

7) Rodrigo Mourato and André Santos, "Program Visualization Framework Using Execution Information," IEEE Access, 2024.

The study presented a visualization framework that collects execution data and transforms it into step-by-step visuals. It promotes self-directed learning by allowing learners to explore runtime behavior interactively.

*8)* S. K. Mehta et al., "Visualization System for Comparing Pathfinding and Sorting Algorithms," Elsevier Procedia Computer Science, 2024.
This system visually compares algorithmic efficiency in real-time. Students can observe how different algorithms handle identical input, fostering analytical understanding of time and space complexity.

*9)* A. Shaffer et al., "Active Learning with Algorithm Animation: Revisiting Educational Impacts," Springer Educational Technology Research, 2023.
This paper reaffirmed that hands-on manipulation of visualizations — such as pausing or rewinding — significantly strengthens student comprehension and memory retention.

*10)* T. Naps et al., "User Control Features in Algorithm Visualization Systems," ACM SIGCSE Bulletin, 2023.
The authors emphasized the importance of user control features like pause, step, and rewind functions, which allow learners to progress at their own pace and improve conceptual absorption.

*11)* P. Brusilovsky et al., "Adaptive Visualization Systems for Personalized Learning," IEEE Transactions on Learning Technologies, 2023.
This work explored intelligent visualization systems that adapt content delivery based on user behavior, promoting personalized learning experiences in programming education.

*12)* K. E. Hundhausen et al., "Animation Combined with Explanation in Programming Education," Journal of Computing Sciences in Colleges, 2023.
The authors found that combining algorithm animation with instructor or system-based explanations enhances comprehension compared to animation-only teaching methods.

*13)* S. H. Edwards and J. Pérez Alvarez, "Accessibility in Algorithm Visualization Tools: Device-Agnostic Approaches," Journal of Educational Technology Systems, 2023.
This study discussed the importance of designing visualization systems that support multiple devices and languages, ensuring inclusive and accessible learning environments.

*14)* K. Patel and R. Deshmukh, "Real-Time Algorithm Execution Visualizer with Interactive Controls," Springer Advances in Computing, 2022.
The research developed a visualizer supporting live execution with interactive control buttons. It concluded that student comprehension time improved when learners directly interacted with algorithm flow.

*15)* M. Lee and H. Park, "Unified Visualization Platform for Sorting and Graph Algorithms," International Journal of Advanced Computer Applications, 2023.
This research developed an integrated visualization platform capable of simulating sorting, searching, and graph algorithms in one environment. It proved effective in improving comprehension speed and facilitating comparative algorithm study.

*16)* L. Tao et al., "Multi-Language Visualization Framework for Sorting Algorithm Education," IEEE Computer Applications in Engineering Education, 2022.
This study proposed a platform where learners can view sorting algorithms in multiple programming languages, improving code logic understanding and language comparison.

*17)* R. Patel and J. Thomas, "Visual Debuggers for Algorithmic Learning," ACM Transactions on Computing Education, 2022.
The authors introduced a visual debugging interface that maps source code execution to real-time variable and memory changes. Their findings indicate that this method significantly enhances students' debugging and reasoning skills.

*18)* P. Kaur et al., "Augmented Reality-Based Algorithm Animation for Programming Education," Elsevier Computers & Graphics, 2021.

The authors designed an AR-based system where algorithms are visualized in 3D environments, improving student engagement and comprehension through immersive learning.

*19)* R. Mishra and A. Sahu, "Complexity Analysis with Animated Feedback for Learners," International Journal of Educational Technology, 2021.
This system visualizes algorithm complexity during execution, allowing students to see time and space usage dynamically, fostering deeper understanding of performance trade-offs.

*20)* S. A. Ali et al., "Impact of Visualization Interfaces on Cognitive Load and Learning Efficiency," Journal of Computer-Assisted Learning, 2021.
This research analyzed how interface design elements like color coding and animation speed affect cognitive load, finding that clear visualization significantly improves learning speed.

*21)* N. Gupta, R. Mehra, and S. Talwar, "Adaptive Visualization Systems for Programming Education," IEEE Transactions on Learning Technologies, 2021.
This paper presented an adaptive visualization engine that modifies animation speed and detail based on user expertise. The results showed improved engagement for beginners while maintaining efficiency for advanced learners.

*22)* L. Zhang and K. Wong, "Cross-Platform Visualization Framework for Algorithm Education," International Journal of Computer Science Education, 2020.
The authors proposed a responsive, web-based visualization system supporting sorting and searching algorithms. Their research highlighted how cross-device accessibility promotes consistent learning experiences for students across different operating environments.

*23)* D. K. Yadav et al., "Web-Based Visualization Tool for Graph Algorithms," Springer Advances in Intelligent Systems, 2020.
The study built a visual learning platform for graph algorithms, concluding that visual traversal representation enhances understanding of BFS, DFS, and pathfinding methods.

*24)* A. K. Sinha and M. Reddy, "Evaluating Visualization Platforms for Data Structures and Algorithms," Journal of Interactive Learning Research, 2020.
The paper evaluated several online visualization systems and found that students using interactive platforms showed higher motivation and test performance than those using text-based materials.

*25)* A. Shaffer et al., "Student-Constructed Visualizations in Computer Science Education," ACM Transactions on Computing Education, 2019.
This study found that students who create their own algorithm visualizations gain better conceptual retention and long-term understanding than those who passively observe.

*26)* T. Naps et al., "Integrating Visualization into Algorithmic Pedagogy," Journal of Computing Education, 2019.
The authors emphasized the integration of reflection activities such as post-visualization quizzes — to reinforce algorithmic concepts effectively.

*27)* A. Shaffer et al., "Interactive Algorithm Animation: Enhancing Engagement Through Visual Learning," Journal of Educational Computing Research, 2019.
This study demonstrated that allowing learners to manipulate visualizations during algorithm execution increases motivation and conceptual clarity.

*28)* P. Brusilovsky, "Adaptive Hypermedia and Intelligent Tutoring Systems," Springer Computer Science Education Series, 2018.
This foundational work laid the basis for modern adaptive educational tools by exploring intelligent hypermedia and user modeling for personalized algorithm visualization.

*29)* K. Hundhausen et al., "Meta-Analysis of Visualization in Algorithm Learning," ACM Computing Surveys, 2017.
The meta-study concluded that visualization alone is not enough; active engagement, explanation, and interaction are critical for achieving meaningful algorithm understanding.

*30)* S. H. Edwards et al., "Mobile-First Visualization Tools for Programming Education," IEEE Transactions on Mobile Learning, 2016.
This research highlighted the importance of designing responsive, mobile-compatible algorithm visualization tools to increase accessibility for learners worldwide.

## III. SYSTEM ANALYSIS

### *A. Existing System*

The current approach to teaching algorithms and data structures primarily depends on traditional classroom instruction, static diagrams and text-based explanations. Students typically learn through lectures, textbooks or coding exercises without real-time visualization of algorithmic operations. This makes it difficult for beginners to understand complex concepts such as recursion, sorting and graph traversal.

Existing visualization tools available online offer limited interactivity, customization and language support. Most provide only pre-defined demonstrations, preventing users from experimenting with their own code or controlling the execution flow. As a result, learners remain passive observers instead of active participants.

Furthermore, the lack of real-time feedback, intelligent assistance and contextual explanations restricts deeper understanding and personalized learning. These systems often fail to bridge the gap between theoretical knowledge and practical implementation.

Many platforms also lack scalability and extensibility, focusing only on basic algorithms while neglecting advanced topics such as tree balancing, graph algorithms and optimization techniques. This limits their effectiveness in modern computer science education.

Disadvantages of the Existing System:

- Limited interactivity and customization
- Absence of real-time visualization and feedback
- Lack of intelligent assistance or contextual explanations
- Minimal support for advanced algorithmic concepts
- Low engagement and poor conceptual understanding
- Restricted scalability and adaptability

### *B. Proposed System*

The proposed system, Code and Algorithm Visualizer, is an interactive platform that simplifies the learning and understanding of algorithms. It allows users to write code, execute it step by step and visualize internal processes through real-time, color-coded animations.

Unlike traditional methods that rely on static diagrams or theoretical explanations, the system transforms abstract logic into dynamic visual representations. Key features include automatic algorithm detection, interactive execution control and real-time complexity analysis. Users can switch between programming languages, customize themes and adjust or replay execution steps.

The system bridges the gap between theory and practice, serving as both a learning tool for students and a teaching aid for educators. With its intuitive interface, responsive design and modular architecture, it ensures scalability, maintainability and smooth accessibility across devices.

Advantages of the Proposed System:

- Enhanced Learning: Converts complex algorithms into clear, interactive visualizations for better understanding.
- Interactivity: Enables real-time, step-by-step execution to promote active engagement.
- Educational Value: Bridges theoretical knowledge and practical implementation.
- Scalability: Modular design allows easy integration of new algorithms and features.
- Accessibility: Works across devices with multi-language support and responsive design.
- Customization: Allows users to adjust visualization settings for a personalized learning experience.

### C. *Proposed Solution*

The proposed solution introduces an interactive and modular algorithm visualization platform designed to bridge the gap between theory and implementation. It enables users to write, execute and visualize algorithms in real time, making abstract logic structures more intuitive and engaging.

When users input code or select an algorithm from the built-in library, the system performs real-time parsing and analysis to identify the algorithm type. The visualization engine then translates logical operations into animated visual cues, such as highlighting comparisons, swaps and recursive calls.

Core Components of the Solution

- Algorithm Parser: Recognizes common algorithm patterns (e.g., sorting, searching, graph traversal, trees) from user code.
- Visualization Engine: Generates step-by-step visual feedback using color-coded animations for clear comprehension.
- Execution Controller: Provides playback control with options like pause, resume, next-step and reverse execution.
- Educational Overlay: Displays explanations, pseudo-code and performance metrics such as time and space complexity.
- Performance Analyzer: Evaluates algorithm efficiency and presents comparison charts for various algorithm.

This solution enhances understanding by combining visual learning, hands-on interaction and contextual explanations. It allows both beginners and advanced learners to explore how algorithms behave dynamically, promoting analytical thinking and deeper conceptual clarity.

### D. *Ideation & Brainstorming*

The ideation phase involved identifying the primary difficulties faced by students in learning algorithmic concepts. Through brainstorming sessions, several ideas were explored to make algorithm learning more intuitive and engaging.

The team analyzed existing algorithm visualization tools and discovered several shortcomings, such as limited interactivity, lack of real-time control and poor adaptability to user-written code. The brainstorming sessions led to the concept of a code-driven visualizer—an application that can not only animate pre-defined algorithms but also visualize any custom code written by the user.

Key Ideas Generated

- Integration of a real-time code editor linked with a visualization engine.
- Implementation of automatic algorithm recognition to detect algorithm type.
- Inclusion of step navigation controls to move forward, backward or replay algorithm operations.

These collective ideas formed the foundation of the Code Visualizer, ensuring the system provides not only visual representation but also educational context and learner engagement.

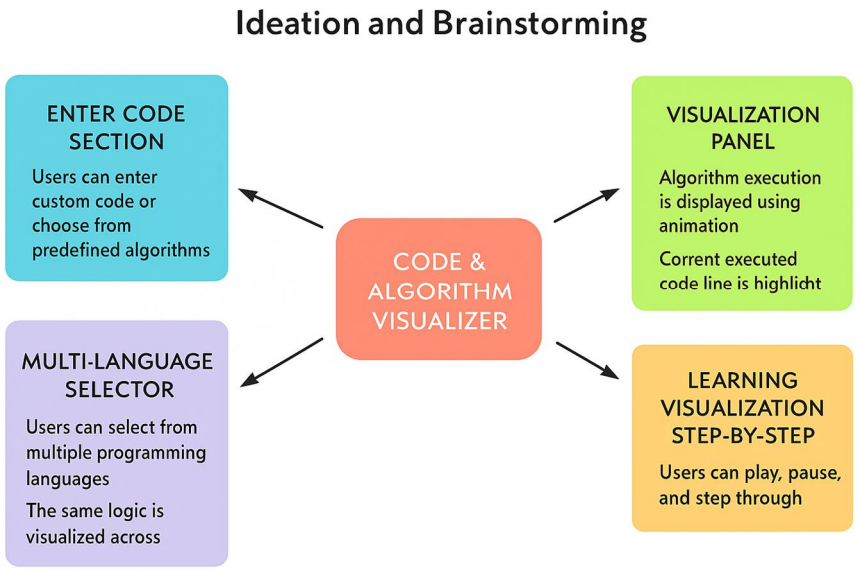



Figure 3.4: Ideation and Brainstorming

*E. Problem Solution Fit*

*1) Identified Problems*

*a)* Students often struggle to understand algorithm flow and logic from static materials.
*b)* Existing tools lack interactivity and do not support user-defined algorithms.
*c)* Limited visualization customization and poor real-time feedback make learning less engaging.
*d)* Educators face challenges in demonstrating algorithm behaviour effectively during lectures.
*e)* Most existing visualizers do not provide performance metrics like time and space complexity for comparison.
*f)* Learners often face difficulty connecting theoretical concepts with actual code execution.
*g)* Visualization tools are not adaptive to different learning speeds.

*2) Proposed Solutions*

*a)* The Code Visualizer addresses these challenges by providing real-time algorithm execution visualization.
*b)* It allows learners to interact with each step of the algorithm, making the experience active rather than passive.
*c)* The system's automatic detection and contextual explanation modules personalize the learning experience.
*d)* Teachers can create custom algorithm demonstrations and exercises, making classroom sessions more interactive.
*e)* Designed for scalability to include advanced algorithms and data structures in future updates.
*f)* Facilitates collaborative demonstrations between teachers and students in real-time.

This alignment between problems and solutions ensures that the Code Visualizer meets both educational.

*F. Architectural Design*

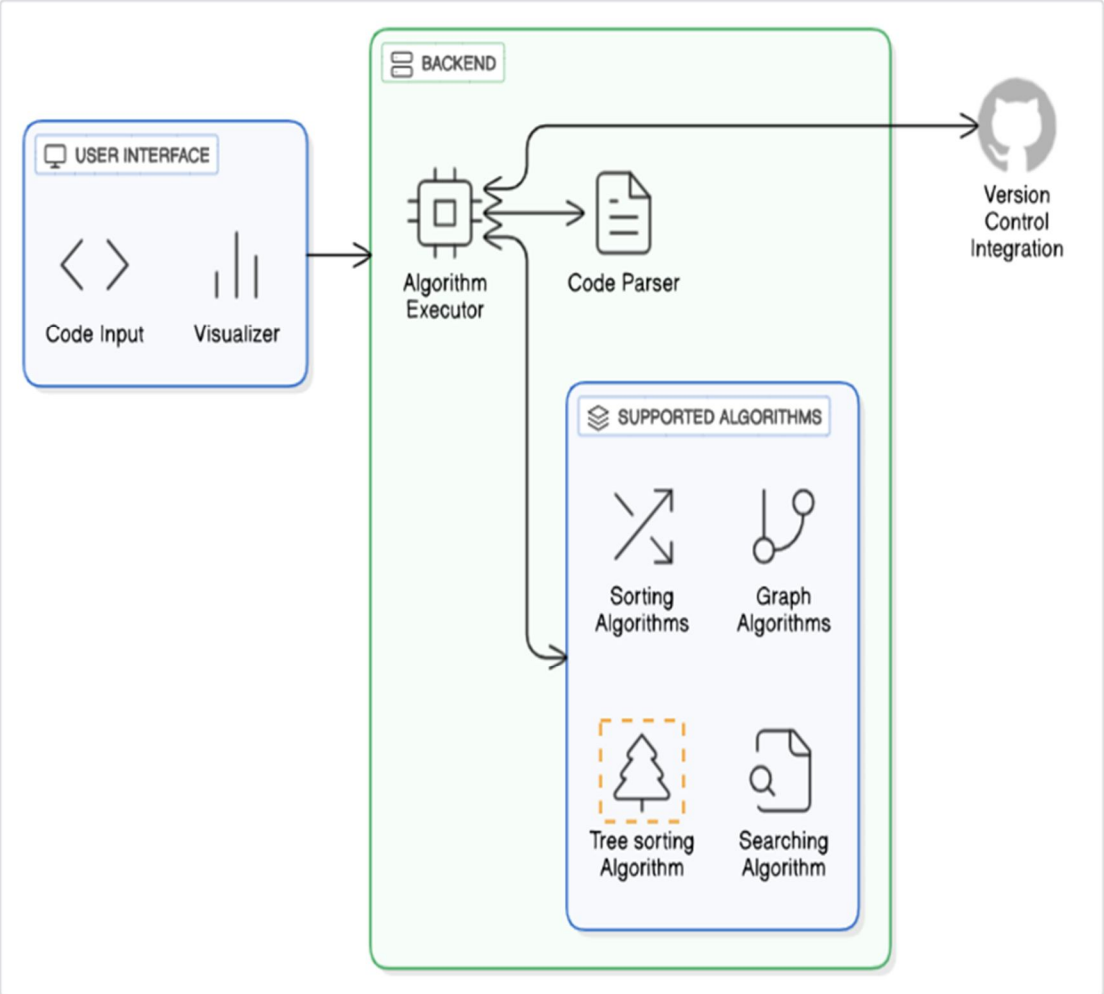



Figure 3.6: Architecture Design

The architecture of the Code and Algorithm Visualizer outlines a seamless interaction between the Frontend (User Interface), Backend Engine and Version Control Integration, creating an intuitive and intelligent learning ecosystem.

*1) User Interface (UI)*

The User Interface component is responsible for interacting with the end user.

It includes two major modules:

- Code Input: Allows users to write or paste code for different algorithms. Users can modify, test, and submit their code for execution.
- Visualizer: Displays a visual representation of the algorithm's execution process. This helps users understand how data structures and steps evolve over time (e.g., how sorting or graph traversal happens).

*2) Backend Engine*

The Backend handles all processing, parsing and algorithm execution.

It consists of several key modules:

a) Algorithm Executor
- Executes the user's submitted algorithm.
- Communicates with both the Code Parser and the Supported Algorithms module.
- Sends execution results to the Visualizer for display.
- Ensures safe and efficient execution of user-submitted code.

b) Code Parser
- Analyzes and interprets the user's input code.
- Converts the code into an internal format that can be processed by the Algorithm Executor.
- Ensures syntax and logic compatibility with supported algorithms.

c) Supported Algorithms
- Sorting Algorithms:
  These algorithms arrange data elements in a specific order, typically ascending or descending. They help improve efficiency in searching, organizing and processing large datasets.
  Example: Bubble Sort, Merge Sort, Quick Sort.
- Graph Algorithms:
  Used to solve problems related to network structures like routes, paths and connectivity. They analyze relationships between nodes and edges in graphs effectively.
  Example: Dijkstra's, BFS, DFS.
- Tree Sorting Algorithm:
  This algorithm uses a binary tree structure to sort data elements hierarchically. It provides efficient insertion and retrieval operations during the sorting process.
  Example: Binary Tree Sort (highlighted, possibly recently added or under development).
- Searching Algorithms:
  These algorithms are designed to locate specific elements within a dataset quickly. They reduce the time complexity of finding data, enhancing program efficiency.
  Example: Linear Search, Binary Search.

Each category represents a set of predefined algorithm templates that the executor can run and visualize.

*3) Version Control Integration*

For collaboration and version tracking, the system connects to external repositories like GitHub, enabling users to manage, share and evolve their algorithms over time.
- Provides integration with GitHub or similar platforms.
- Enables users to push, pull or manage their algorithm code versions directly from the system.
- Facilitates collaboration and backup of work.

Together, these modules create a responsive, scalable and interactive environment that bridges the gap between algorithm theory and real-time understanding.

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue XI Nov 2025- Available at www.ijraset.com*

*G. Dataflow of The System*

The data flow of the Code and Algorithm Visualizer starts with the user entering or selecting code in the input section. The code is then processed and visualized in real time through animated steps in the visualization panel. Users can control execution using the control panel and learn each stage interactively through the step-by-step module. Finally, the multi-language selector allows switching between programming languages, making the learning experience dynamic and versatile.

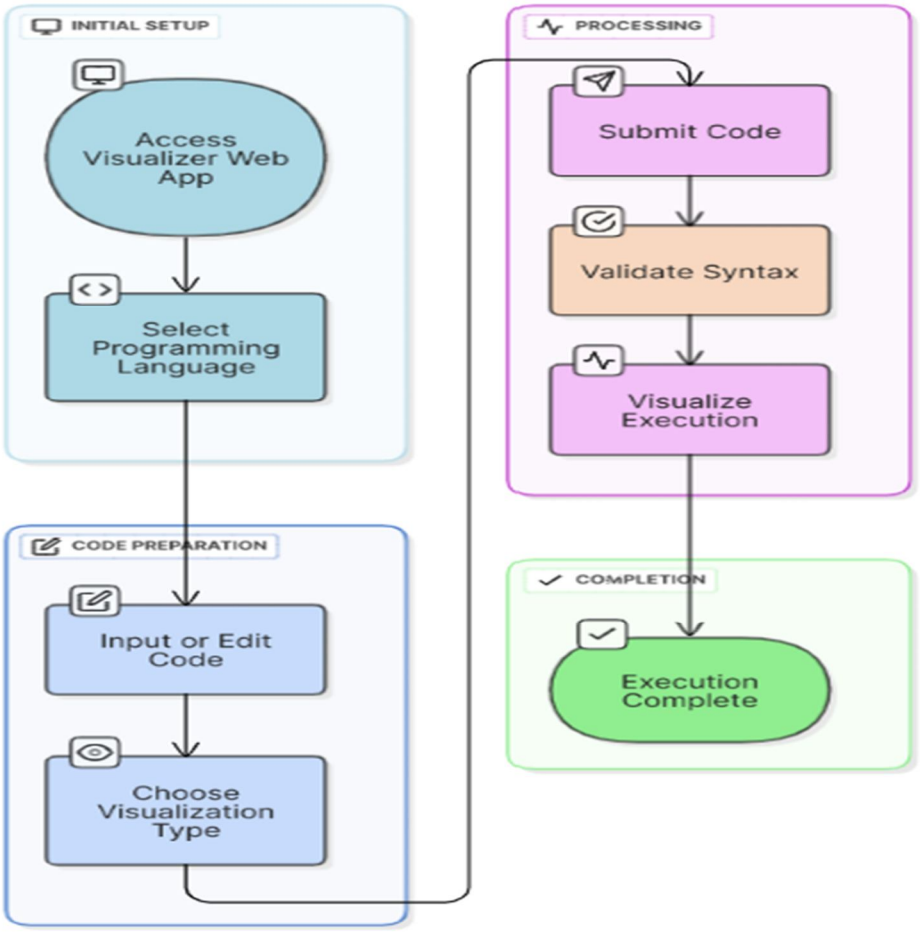



Figure 3.8: Dataflow Diagram

## IV. SYSTEM REQUIREMENTS

*A. Hardware Requirement*

- SYSTEM – A computer or laptop with at least an Intel i5 processor (or equivalent) and 8 GB RAM.
- GRAPHICS – Integrated or dedicated graphics support.
- STABLE INTERNET CONNECTION – Required for loading dependencies, hosting the web application, accessing APIs and synchronizing data.
- WEB BROWSER – Google Chrome, Mozilla Firefox or Microsoft Edge.
- OPERATING SYSTEM – Windows 10 / 11, macOS or any Linux distribution.

*B. Software Requirements*

- IDE / CODE EDITOR – Visual Studio Code
- FRAMEWORK / LIBRARY – React.js
- PROGRAMMING LANGUAGES – HTML5, CSS3 and JavaScript (ES6+)
- BACKEND / DATABASE – Node.js / Firebase / Python Flask for handling user data, algorithm submissions and progress tracking.

- VISUALIZATION LIBRARIES – D3.js, Chart.js, p5.js or custom JavaScript visualization modules for algorithm animations.
- PACKAGE MANAGER – npm or yarn for dependency installation and version management.
- VERSION CONTROL TOOLS – Git with GitHub
- DEPLOYMENT PLATFORM – Firebase Hosting, Netlify, Vercel or GitHub Pages for deploying the web application.
- TESTING TOOLS – Jest, Cypress or Puppeteer for automated testing and validation of the web interface and visualization accuracy.

### *C. Software Description*

#### *1) Visual Studio Code (VS Code)*

Visual Studio Code is a lightweight yet powerful source code editor developed by Microsoft, widely used for modern web and software development. It supports multiple programming languages and frameworks, making it an ideal tool for building the Code and Algorithm Visualizer.

- User Interface: VS Code offers a clean, intuitive interface with customizable themes and layouts. It supports split views, integrated terminal access and quick navigation tools that streamline coding and debugging workflows.
- Code Editing: The editor includes intelligent features like syntax highlighting, auto-completion and real-time linting. These make it easy for developers to write efficient, error-free code in JavaScript, React or Python.
- Extension Support: VS Code provides an extensive library of extensions, including Prettier, ESLint, React Developer Tools and Git integration, allowing customization for front-end visualization and backend processing.
- Integrated Git Control: Built-in Git support simplifies version control, enabling developers to commit, push, pull and resolve merge conflicts directly from the editor.
- Debugger: The integrated debugger helps in identifying logic errors, testing code flow and monitoring variables in real-time during the visualization build process.
- Cross-Platform Compatibility: Available on Windows, macOS and Linux, VS Code ensures a seamless experience across development environments.

#### *2) ReactJS*

ReactJS is a popular JavaScript library developed by Facebook for building dynamic and interactive user interfaces. It forms the core of the visualization front-end in this project.

- Component-Based Architecture: React allows building reusable UI components, making the visualization structure modular and easier to maintain. Each visualization panel, editor and control bar is implemented as an independent component.
- Virtual DOM: The Virtual DOM in React enhances rendering speed by updating only the necessary parts of the user interface, ensuring smooth animation and transitions during algorithm visualization.
- Declarative UI: Developers can define the visualization logic declaratively, allowing the system to manage updates efficiently and reduce the chances of bugs.
- Integration with Libraries: React easily integrates with libraries such as Framer Motion (for animations), PrismJS (for syntax highlighting) and Redux (for state management).
- Cross-Platform Efficiency: The same React codebase can be adapted for both web and desktop applications using frameworks like Electron.
- Developer Tools: React Developer Tools in Chrome/Edge browsers provide insights into component behavior and performance, aiding efficient debugging.

#### *3) Node.js and Express.js:*

Node.js is a JavaScript runtime environment built on Chrome's V8 engine and Express.js is a lightweight web application framework for Node.js. Together, they form the backend engine for handling algorithm execution and user interaction.

- Server-Side Processing: Node.js handles input from the visualization interface, processes algorithm data and returns results in real time.
- Asynchronous and Event-Driven: Node.js supports non-blocking I/O operations, allowing multiple users to execute code visualizations simultaneously.
- Express.js Framework: Express simplifies backend routing, API creation and communication between the visualization interface and backend logic.

- Performance and Scalability: The combination of Node.js and Express.js ensures high performance, making the system responsive even under heavy load.
- Security: Middleware functions in Express help in validating user inputs and securing API endpoints.
- Cross-Platform Deployment: Applications built with Node.js can be deployed on various platforms like AWS, Azure or local servers easily.
- Integration: The backend seamlessly integrates with databases and visualization libraries, ensuring real-time updates and smooth execution.

*4) Git and GitHub*

Git is a distributed version control system and GitHub is a cloud-based hosting service for Git repositories. Both play a crucial role in collaborative development and version management of the Code Visualizer project.

- Version Tracking: Git allows developers to track every change made to the source code, ensuring that earlier versions can be easily retrieved when needed.
- Collaboration: Multiple developers can work on the project simultaneously using branching and merging features, minimizing conflicts.
- Backup and Security: GitHub provides secure, cloud-based storage for source code, protecting it from local machine failures.
- Integration with VS Code: Built-in integration within VS Code allows direct commits, push/pull actions and branch management from the editor itself.
- Documentation and Issues: GitHub's README and Issues sections help maintain documentation and track development progress.
- Continuous Development: GitHub Actions can automate testing and deployment, ensuring consistent updates to the visualization system.
- Open-Source Collaboration: By hosting the project on GitHub, the Code Visualizer can be shared and improved by the developer community worldwide.

*5) Framer Motion and PrismJS*

Framer Motion and PrismJS are two major libraries used for enhancing the visual and interactive experience in the Code and Algorithm Visualizer.

- Framer Motion: Used for creating smooth animations and transitions, it brings the visualization to life by animating algorithm execution, variable updates and flow transitions.
- PrismJS: This library provides syntax highlighting for multiple programming languages like JavaScript, Python, Java and C++, making the code editor both interactive and educational.
- Customization: Both libraries offer extensive customization, allowing fine-tuning of colors, motions and transitions for better learning visualization.
- Integration: Seamlessly integrates with React components and enhances user engagement with minimal performance overhead.
- Performance: Lightweight and optimized for rendering visual elements in real time.
- These technologies collectively ensure that users not only understand algorithm logic but also visually experience its step-by-step execution.

## V. CODE IMPLEMENTATION

*A. Code Panel*

The Code Visualizer begins with an intuitive code input panel, where users can type or paste their algorithm code. The editor provides syntax highlighting, indentation and error checking for a smooth coding experience. Users can also choose predefined algorithms such as sorting, searching or tree traversal. This section acts as the starting point for visualization, allowing learners to understand the structure and logic of their code before execution. A simple "Run" button initiates the process, sending the code to the visualization engine for analysis and animation.

Here's the code to setup the registration page:

```
<div>
  <textarea  className="code-textarea"  value={code}
    onChange={(e) => onCodeChange(e.target.value)}
    placeholder="Enter your code here..."  />
```

```
<SyntaxHighlighter
  language={language}
  style={theme === 'dark' ? vscDarkPlus : vs}
  wrapLines
  lineProps={(lineNumber) => ({
    style: {    backgroundColor:
       lineNumber === currentLine ? 'rgba(33,150,243,0.1)' : 'transparent',
     borderLeft: lineNumber === currentLine ? '4px solid #2196F3' : 'none',
    },  })}>
  {code || ' '}
 </SyntaxHighlighter>
 <span>Executing line {currentLine}</span>
</div>
```

*B. Visualization panel*

After execution, the visualization panel animates the algorithm's operations in real time. Each comparison, swap or traversal step is shown through color-coded movements and transitions. The corresponding code line is highlighted simultaneously for better clarity. This creates a direct link between written logic and visual understanding.

Here's the code for visualization:

```
const [currentStep, setCurrentStep] = useState(0);
useEffect(() => {
  if (steps. length > 0 && isPlaying) {
    const interval = setInterval (() => {
      setCurrentStep((prev) => (prev < steps.length - 1? prev + 1: prev));
    }, speed);
    return () => clearInterval(interval);
  }
},
[isPlaying, speed, steps]);
const currentData = steps[currentStep];
updateVisualization(currentData);
highlightCodeLine (currentData.lineNumber);
```

*C. Learn: Step-by-Step Execution (Execution Trace)*

The system divides execution into small, manageable steps for detailed observation. Users can study how variables change and loops iterate with every step. This feature makes algorithm flow transparent, encouraging hands-on exploration and independent learning. It helps users grasp complex concepts with visual evidence rather than abstract reasoning**.**

Here's the code to step by step execution:

```
      const [stepIndex, setStepIndex] = useState(0);
    const handleNextStep = () => {
    if (stepIndex < steps.length - 1) {
    setStepIndex(stepIndex + 1);
     const current = steps[stepIndex + 1];
     updateVariables(current.variables);
     highlightCodeLine(current.line);
       }
};
const handlePrevStep = () => {
  if (stepIndex > 0) {
    setStepIndex(stepIndex - 1);
```

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue XI Nov 2025- Available at www.ijraset.com*

```
    highlightCodeLine(steps[stepIndex - 1].line);
  }
}
```

*D. Control Panel (Playback & Interaction Controls)*

The control panel gives users full command over the visualization process. They can play, pause, move forward or backward, change speed or reset execution at any time. These controls make it easy to focus on specific sections or replay complex parts. The interactive nature of the panel enhances engagement and active participation.

Here's the code to step by step control.

```
const [stepIndex, setStepIndex] = useState(0);
const handleNextStep = () => {
  if (stepIndex < steps.length - 1)
 {
    setStepIndex(stepIndex + 1);
    const current = steps[stepIndex + 1];
    updateVariables(current.variables);
    highlightCodeLine(current.line);
  }
};
const handlePrevStep = () => {
  if (stepIndex > 0) {
    setStepIndex(stepIndex - 1);
    highlightCodeLine(steps[stepIndex - 1].line);
}
};
```

*E. Multi-Language Selector*

To support diverse learners, the platform offers a multi-language selector for JavaScript, Python, Java and C++. Users can switch languages seamlessly and visualize the same algorithm across different syntaxes. This feature strengthens understanding of algorithmic logic while improving cross-language coding skills.

Here's the code for multiple language selector.

```
const [isOpen, setIsOpen] = useState(false);
const currentLang = languages[currentLanguage];
<motion.button
  className="language-toggle"
  onClick={() => setIsOpen(!isOpen)}
  whileHover={{ scale: 1.05 }}
  whileTap={{ scale: 0.95 }} >
  <span className="language-icon">{currentLang.icon}
</span>
  <span className="language-name">{currentLang.name}
</span>
 <ChevronDown size={16}
 />
</motion.button>
{
isOpen && (
  <motion.div className="language-dropdown">
    {Object.entries(languages).map(([key, lang]) => (
      <button    key={key}
```

```
      className={`language-option ${currentLanguage === key ? 'active' : ''}`
}
      onClick={() => { onLanguageChange(key); setIsOpen(false);}} >
      <span className="language-icon">{lang.icon}
      </span>
      <span className="language-name">{lang.name}
      </span>
      </button>
   )
)}
  </motion.div>
)
}
```

*F. Result*

The results of the Code and Algorithm Visualizer project highlight its effectiveness as a modern educational tool that simplifies the understanding of program execution and algorithmic logic. The system provides a practical and visual approach to learning by displaying how variables, loops and data structures change during runtime. Through interactive animations and real-time feedback, the application enhances conceptual clarity, improves problem-solving skills and makes learning more engaging for students and developers. This project successfully bridges the gap between theoretical programming and practical visualization, enabling learners to observe the exact flow of logic, function calls and recursive processes as they occur in code. By integrating frontend and backend technologies such as React.js, Node.js and JavaScript, the platform delivers a smooth and responsive user experience that supports multiple algorithm categories including sorting, searching, tree and graph algorithms.

Additionally, the system's modular design and version control integration provide scalability and collaboration opportunities, making it ideal for both individual learners and classroom environments. The visualizer not only aids in debugging and performance analysis but also encourages experimentation, allowing users to modify parameters and instantly observe outcomes. Overall, the project demonstrates how interactive visualization can transform algorithm learning into an intuitive, engaging, and highly effective experience.

THE APP FLOW:

*1) Home Page / Algorithm Selection:*

When the user opens the application, they are welcomed with a clean and interactive interface. Users can easily select algorithm categories such as Sorting, Searching, Tree or Graph Visualization.

Each category is visually represented for quick understanding and accessibility.

The interface ensures smooth navigation and intuitive learning flow.

It serves as the starting point for all algorithm exploration.

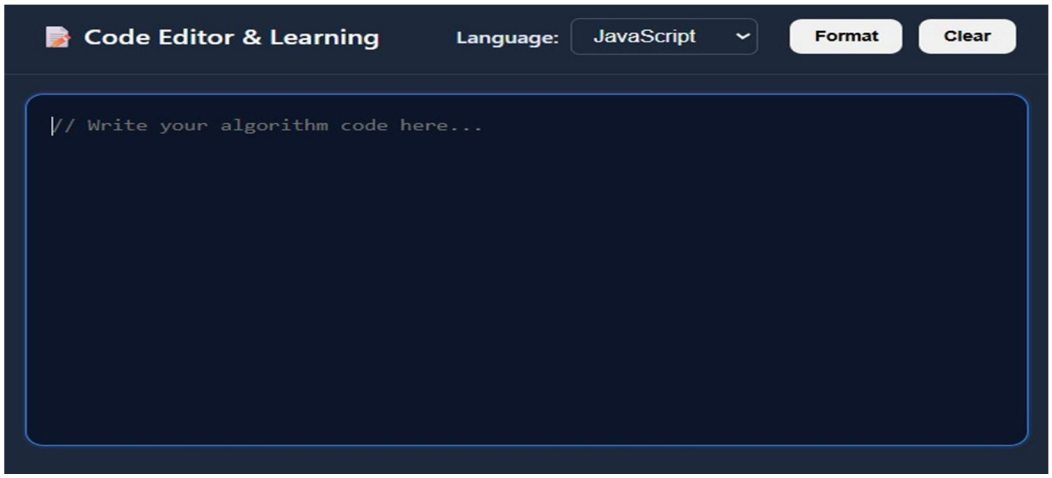



Figure 5.6.1: Code Editor Interface

*2) Algorithm Input Interface*

After selecting an algorithm, users are prompted to provide input values for execution. They can manually enter data such as numbers, nodes or elements as needed. Sample data is also available for quick testing and demonstration.

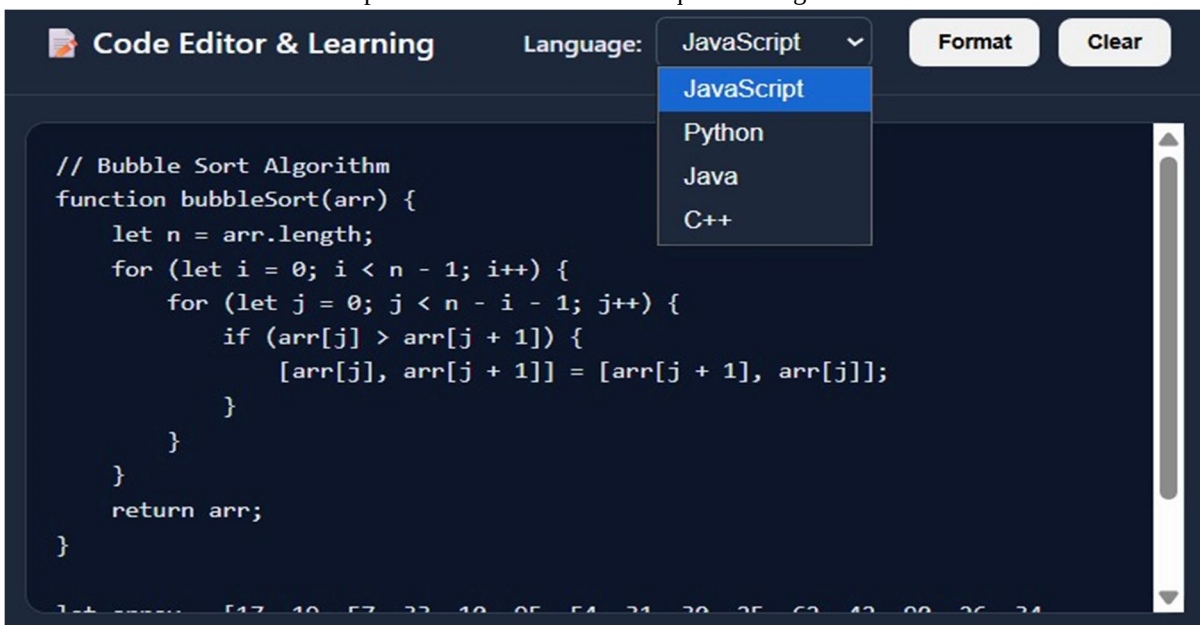



Figure 5.6.2: Algorithm Input Interface

*3) Execution and Visualization*

Once the algorithm is selected and input is given, the visualization engine executes the algorithm step by step.

- Each step of the code is highlighted in real-time.
- The corresponding animation is displayed showing data changes (like swaps in sorting, traversals in graphs or node insertions in trees).
- Users can pause, resume or step through the algorithm manually.
- This enhances understanding of internal logic and data transformations.

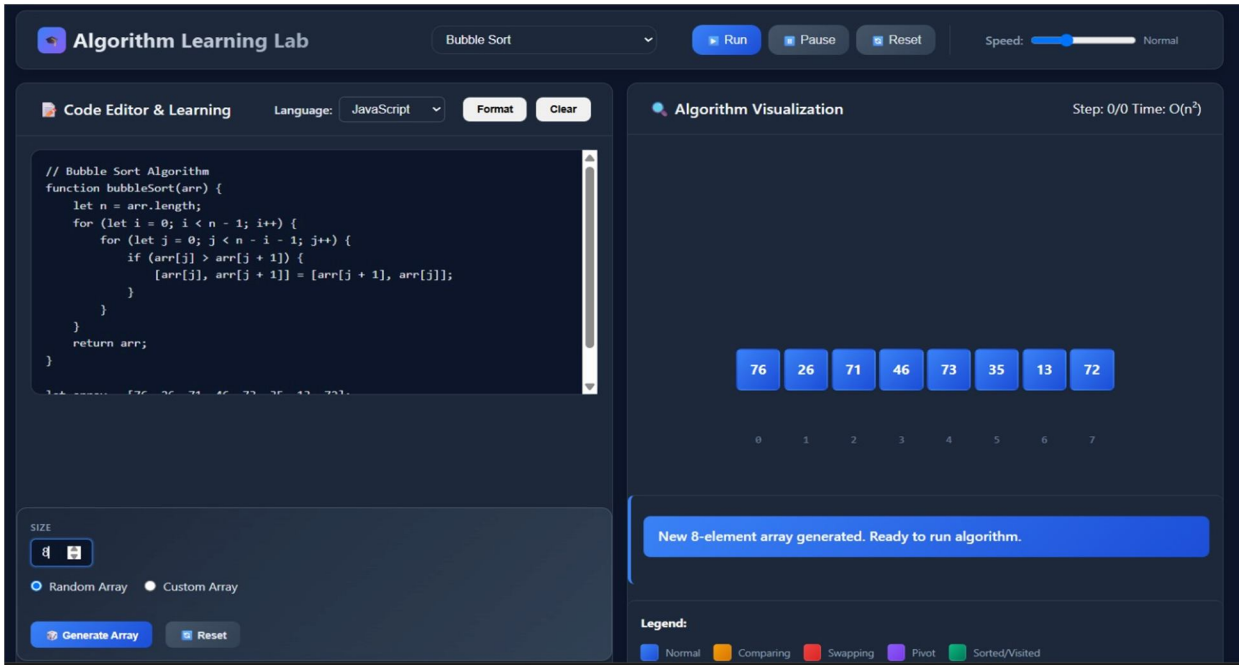



Figure 5.6.3: Visualization Interface

*4) Output Display:*

The final output of the algorithm executed within the application. The interface displays visual and textual results for various algorithm categories such as Sorting, Searching and Tree Traversal, providing users with clear insights into the step-by-step process and overall execution summary.

*a) Sorting Algorithm Output*

The output of sorting algorithms (e.g., Bubble Sort, Merge Sort, Quick Sort) shows the transformation of an unsorted array into a sorted sequence.

Each comparison, swap and iteration is visually highlighted, allowing users to track progress in real-time. The final sorted list is displayed along with metrics such as total swaps and execution time.

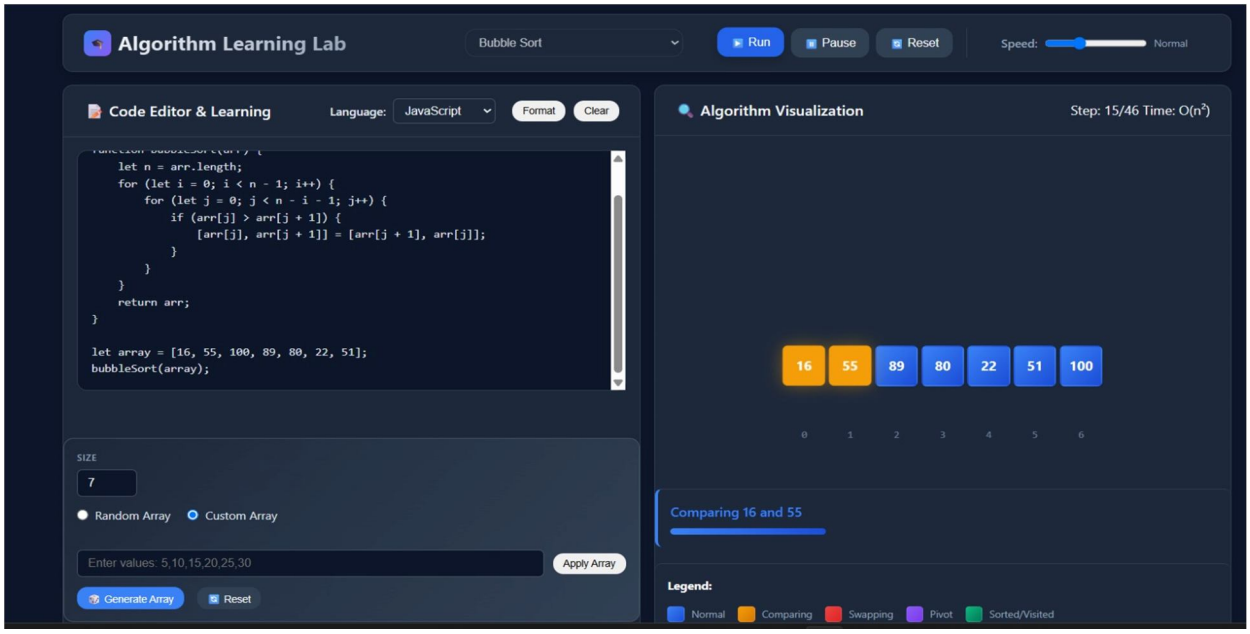



Figure 5.6.4(a): Sorting Algorithm Output

*b) Searching Algorithm Output*

For searching algorithms (e.g., Linear Search, Binary Search), the interface highlights the process of locating a target element within the dataset.

The visualization clearly marks comparisons and the element's position when found. The output includes search result status, total comparisons made and time complexity.

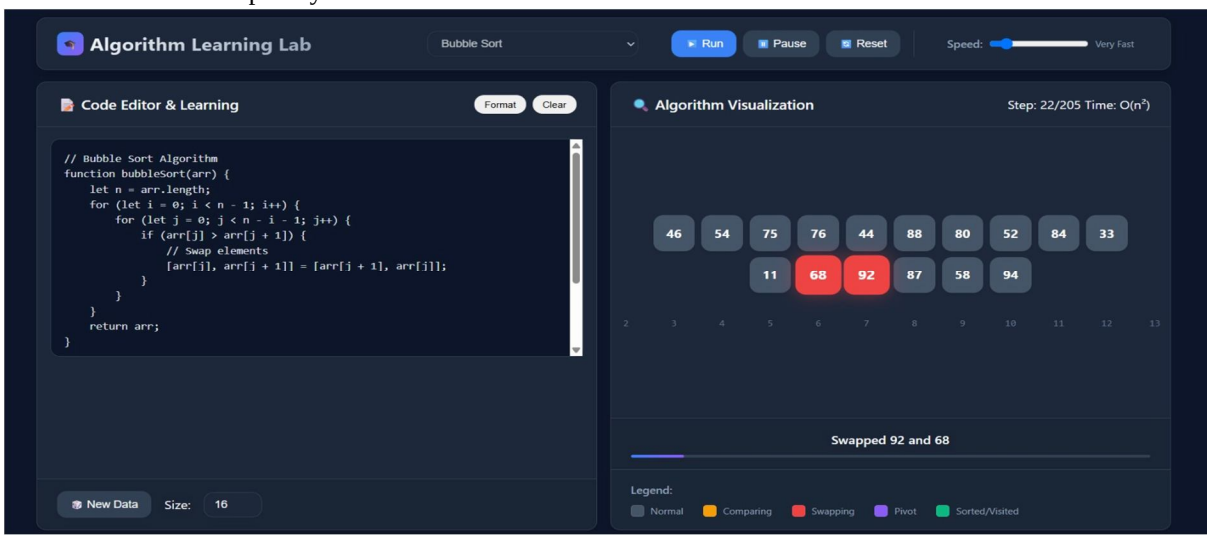



Figure 5.6.4(b): Searching Algorithm Output

*c) Tree Algorithm Output*

Tree based algorithms ( e.g., Binary Search Tree traversal, insertion or deletion) are represented through dynamic tree structures. Each node's insertion and traversal order (Inorder, Preorder, Postorder) are visually animated. The output view summarizes the traversal result and illustrates parent-child relationships in the tree.

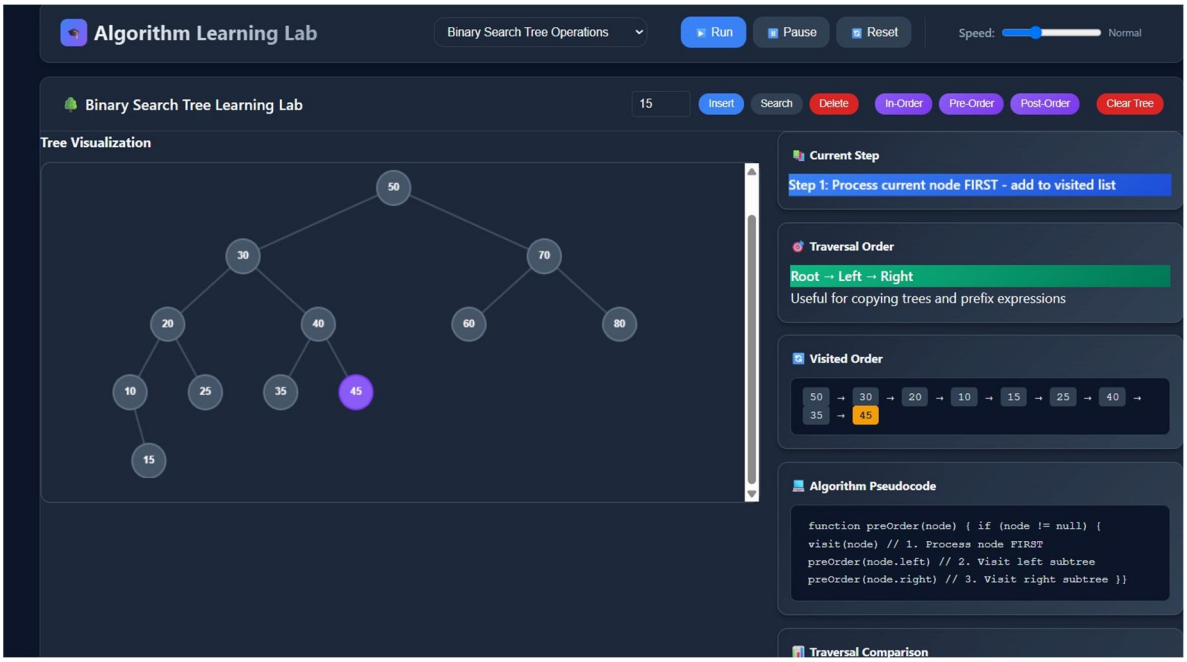



Figure 5.6.4(c): Tree Algorithm Output

*d) Graph Algorithm Output*

Graph algorithms help analyze relationships between connected data points using nodes and edges. They are essential for solving problems in networks, paths and connectivity.

Common examples include BFS, DFS, Dijkstra's and Minimum Spanning Tree algorithms.

These algorithms visually represent how nodes are explored and connected.

They enhance understanding of traversal, routing, and shortest path concepts.

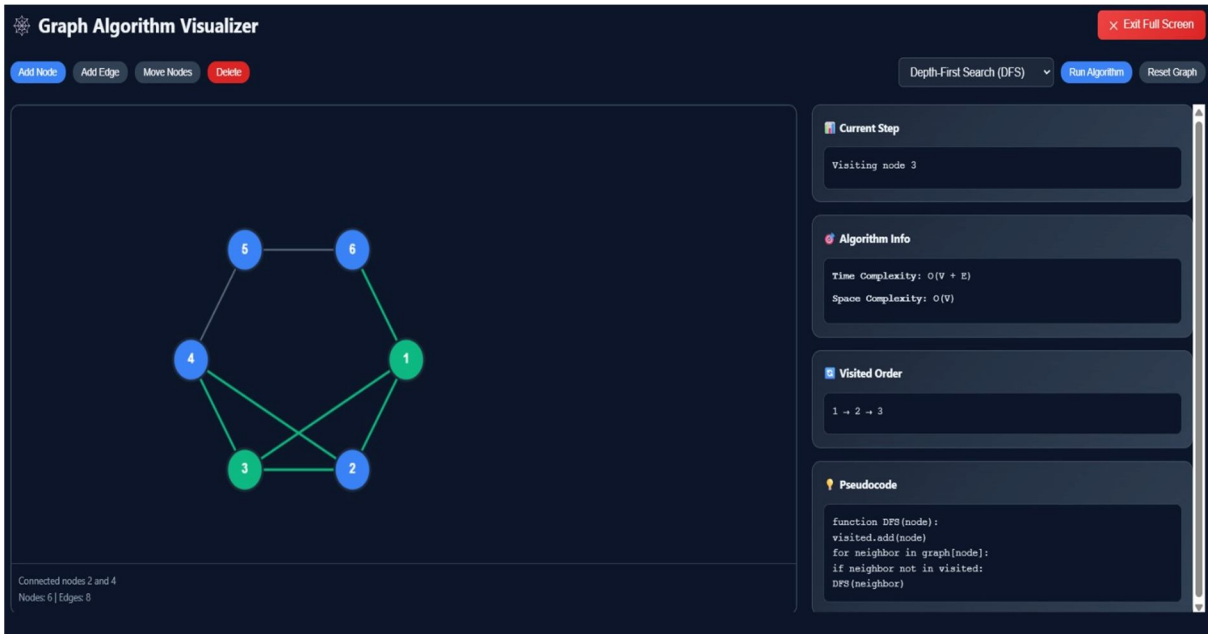



Figure 5.6.4(d): Graph Algorithm Output

## VI. CONCLUSION AND FUTURE ENHANCEMENT

### A. Conclusion

In conclusion, the Code Visualizer project represents a transformative step toward enhancing computer science education through interactive and visual learning. By providing a comprehensive and engaging platform for algorithm visualization, the project addresses the challenges faced by students in understanding abstract algorithmic concepts. Through real-time execution, step-by-step animation and dynamic feedback, it bridges the gap between theoretical knowledge and practical implementation.

The system not only focuses on teaching algorithms but also on making learning intuitive, accessible and enjoyable. With features such as automatic algorithm detection, complexity analysis, educational overlays and interactive execution controls, learners gain a deeper and more meaningful understanding of how algorithms function internally. The platform encourages active participation and exploration, transforming passive learners into engaged problem-solvers. Moreover, the project emphasizes accessibility and inclusivity through its multi-language support, responsive interface and cross-platform compatibility, ensuring that learners from diverse backgrounds can benefit equally.

By developing this application, the project contributes to the advancement of modern digital education and the empowerment of students with strong computational and analytical thinking skills. It signifies a commitment to fostering curiosity, creativity and understanding in the field of computer science. The Code Visualizer stands as a step forward in reimagining how algorithms are taught and learned—making complex concepts simpler, clearer and more accessible to learners everywhere.

### B. Future Scope

1) Integration of Artificial Intelligence: Future versions of the application can incorporate AI-driven virtual assistants capable of providing intelligent feedback, real-time code suggestions and personalized learning support. This would transform the system into an intelligent tutoring platform that guides learners based on their progress and learning patterns.
2) Expansion of Algorithm Library: The repository of algorithms can be continuously expanded to include advanced topics such as dynamic programming, greedy algorithms, backtracking and graph optimization techniques. This will ensure that learners at all levels—from beginners to advanced programmers—can benefit from comprehensive coverage.
3) Adaptive Learning System: By implementing adaptive learning algorithms, the platform can analyze user interactions and performance data to automatically adjust the pace, difficulty level and type of algorithm demonstrations presented. This personalization would make the learning journey more effective and engaging for each individual learner.
4) Cloud-Based Integration and Offline Mode: Enhancing cloud integration will allow users to store their progress, visualization sand notes securely online. Additionally, improved offline functionality can ensure uninterrupted learning, even without internet connectivity—making the platform accessible to a wider audience.
5) Educator Dashboard and Analytics: Future updates could include advanced educator tools with analytics dashboards to monitor class performance, identify learning gaps and assign customized algorithm tasks.

## APPENDICES

SOURCE CODE

App.jsx:

```
import React, { useState, useEffect } from 'react';
import ControlPanel from './components/controls/ControlPanel';
import CodeExecutorVisualizer from './components/code/CodeExecutorVisualizer';
import LearningPanel from './components/learning/LearningPanel';
import { codeTemplates } from './utils/codeTemplates';
import './App.css';
function App() {
  const [currentAlgorithm, setCurrentAlgorithm] = useState('bubble');
  const [language, setLanguage] = useState('javascript');
  const [code, setCode] = useState('')
  // Initialize code when algorithm or language changes
  useEffect(() => {
                const template = codeTemplates[currentAlgorithm]?.[language] ||
                 codeTemplates.default[language];
```

```
    setCode(template);
  }, [currentAlgorithm, language]);
  const getExampleCodes = () => {
    return {
      fibonacci: `// Fibonacci Sequence - Watch variables change in real-time!
let n = 6;
let a = 0;
let b = 1;
let next = b;
let count = 1;
console.log("Fibonacci sequence:");
while (count <= n) {
  console.log("Fibonacci number:", next);
  count = count + 1;
  a = b;
  b = next;
  next = a + b;
}`,
      bubbleSort: `// Bubble Sort - Watch the array get sorted!
let arr = [64, 34, 25, 12, 22, 11, 90];
let n = arr.length;
console.log("Original array:", arr);
for (let i = 0; i < n - 1; i++) {
  console.log("--- Outer loop iteration:", i + 1, "---");
  for (let j = 0; j < n - i - 1; j++) {
    console.log("Comparing", arr[j], "and", arr[j + 1]);
    if (arr[j] > arr[j + 1]) {
      // Swap elements
      let temp = arr[j];
      arr[j] = arr[j + 1];
      arr[j + 1] = temp;
      console.log("Swapped! New array:", arr);
    }
  }
  console.log("After iteration", i + 1, ":", arr);
}
console.log(" Final sorted array:", arr);`,
      factorial: `// Factorial Calculation - See recursion in action!
function factorial(n) {
  console.log("Calculating factorial of", n);
  if (n === 0 || n === 1) {
    console.log("Base case: factorial(", n, ") = 1");
    return 1;
  }
  let result = n * factorial(n - 1);
  console.log("factorial(", n, ") =", result);
  return result;
}
let numbers = [10, 23, 45, 67, 89, 12, 34];
let target = 67;
```

```
console.log("Starting linear search...");
let index = linearSearch(numbers, target);
console.log("Search completed. Result: index", index);`
    };
  };
  const loadExample = (exampleName) => {
    const examples = getExampleCodes();
    setCode(examples[exampleName] || code);
  };
  return (
    <div className="app-container">
      <header className="app-header">
        <div className="logo">
          <div className="logo-icon">□</div>
          <h1>Algorithm Learning Lab - Real Code Execution</h1>
        </div>
      </header>
      <main className="main-content">
        <div className="examples-panel">
          <h3>□ Live Code Examples</h3>
          <div className="example-buttons">
            <button onClick={() => loadExample('fibonacci')} className="example-btn">
              □ Fibonacci
            </button>
            <button onClick={() => loadExample('bubbleSort')} className="example-btn">
              □ Bubble Sort
            </button>
            <button onClick={() => loadExample('factorial')} className="example-btn">
              □ Factorial
            </button>
            <button onClick={() => loadExample('linearSearch')} className="example-btn">
              □ Linear Search
            </button>
          </div>
        </div>
        <div className="content-area">
          <CodeExecutorVisualizer
            code={code}
            language={language}
          />
        </div>
        <LearningPanel
          algorithm={currentAlgorithm}
        />
      </main>
    </div>
  );
}
```

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue XI Nov 2025- Available at www.ijraset.com*

LearningPanel.jsx:

```
import React, { useState, useEffect } from 'react';
import ControlPanel from './components/controls/ControlPanel';
import CodeExecutorVisualizer from './components/code/CodeExecutorVisualizer';
import LearningPanel from './components/learning/LearningPanel';
import { codeTemplates } from './utils/codeTemplates';
import './App.css';
function App() {
  const [currentAlgorithm, setCurrentAlgorithm] = useState('bubble');
  const [language, setLanguage] = useState('javascript');
  const [code, setCode] = useState('');
  // Initialize code when algorithm or language changes
  useEffect(() => {
    const template = codeTemplates[currentAlgorithm]?.[language] || codeTemplates.default[language];
    setCode(template);
  }, [currentAlgorithm, language])
  const getExampleCodes = () => {
    return {
      fibonacci: `// Fibonacci Sequence - Watch variables change in real-time!
let n = 6;
let a = 0;
let b = 1;
let next = b;
let count = 1;
console.log("Fibonacci sequence:");
while (count <= n) {
  console.log("Fibonacci number:", next);
  count = count + 1;
  a = b;
  b = next;
  next = a + b;
}`,
      bubbleSort: `// Bubble Sort - Watch the array get sorted!
let arr = [64, 34, 25, 12, 22, 11, 90];
let n = arr.length;
console.log("Original array:", arr);
for (let i = 0; i < n - 1; i++) {
  console.log("--- Outer loop iteration:", i + 1, "---");
  for (let j = 0; j < n - i - 1; j++) {
    console.log("Comparing", arr[j], "and", arr[j + 1]);
    if (arr[j] > arr[j + 1]) {
      // Swap elements
      let temp = arr[j];
      arr[j] = arr[j + 1];
      arr[j + 1] = temp;
      console.log("Swapped! New array:", arr);
    }
}
  console.log("After iteration", i + 1, ":", arr);
}
```

*ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538*
*Volume 13 Issue XI Nov 2025- Available at www.ijraset.com*

```
console.log("□ Final sorted array:", arr);`,
    factorial: `// Factorial Calculation - See recursion in action!
function factorial(n) {
 console.log("Calculating factorial of", n);
 if (n === 0 || n === 1) {
  console.log("Base case: factorial(", n, ") = 1");
  return 1;
 }
 let result = n * factorial(n - 1);
 console.log("factorial(", n, ") =", result);
 return result;
}
 return (
  <div className="app-container">
   <header className="app-header">
    <div className="logo">
     <div className="logo-icon">□</div>
     <h1>Algorithm Learning Lab - Real Code Execution</h1>
    </div>
   </header>
   <main className="main-content">
    <ControlPanel
     currentAlgorithm={currentAlgorithm}
     onAlgorithmChange={setCurrentAlgorithm}
     onLanguageChange={setLanguage}
    />
    <div className="examples-panel">
     <h3>□ Live Code Examples</h3>
     <div className="example-buttons">
      <button onClick={() => loadExample('fibonacci')} className="example-btn">
       □ Fibonacci
      </button>
      <button onClick={() => loadExample('bubbleSort')} className="example-btn">
       □ Bubble Sort
      </button>
      <button onClick={() => loadExample('factorial')} className="example-btn">
       □ Factorial
      </button>
      <button onClick={() => loadExample('linearSearch')} className="example-btn">
       □ Linear Search
      </button>
     </div>
    </div>
    <div className="content-area">
     <CodeExecutorVisualizer
      code={code}
      language={language}
     />
    </div>
    <LearningPanel
```

```
        algorithm={currentAlgorithm}
       />
      </main>
    </div>
  );
}
export default App;
```

LanguageSwitcher.jsx:

```
import React, { useState } from 'react';
import { motion, AnimatePresence } from 'framer-motion';
import { ChevronDown } from 'lucide-react';
import './LanguageSwitcher.css';

const LanguageSwitcher = ({ languages, currentLanguage, onLanguageChange }) => {
  const [isOpen, setIsOpen] = useState(false);
  const currentLang = languages[currentLanguage];

  return (
    <motion.div className="language-switcher" initial={{ opacity: 0 }} animate={{ opacity: 1 }}>
      <motion.button
        className="language-toggle"
        onClick={() => setIsOpen(!isOpen)}
        whileHover={{ scale: 1.05 }}
        whileTap={{ scale: 0.95 }}
      >
        <span className="language-icon">{currentLang.icon}</span>
        <span className="language-name">{currentLang.name}</span>
        <motion.div
          animate={{ rotate: isOpen ? 180 : 0 }}
          transition={{ duration: 0.2 }}
        >
          <ChevronDown size={16} />
        </motion.div>
      </motion.button>

      <AnimatePresence>
        {isOpen && (
          <motion.div
            className="language-dropdown"
            initial={{ opacity: 0, y: -10 }}
            animate={{ opacity: 1, y: 0 }}
            exit={{ opacity: 0, y: -10 }}
          >
            {Object.entries(languages).map(([key, lang]) => (
              <motion.button
                key={key}
                className={`language-option ${currentLanguage === key ? 'active' : ''}`}
                onClick={() => {
                  onLanguageChange(key);
```

```
                setIsOpen(false);
              }
            }
              whileHover={{ x: 5 }}
              whileTap={{ scale: 0.95 }}
            >
              <span className="language-icon">{lang.icon}</span>
              <span className="language-name">{lang.name}</span>
            </motion.button>
          )
            )}
        </motion.div>
      )
      }
    </AnimatePresence>
  </motion.div>
 );
};
export default LanguageSwitcher;
```

Visualization.js:

```
class Visualizer {
  constructor(containerId, indexContainerId) {
    this.container = document.getElementById(containerId);
    this.indexContainer = document.getElementById(indexContainerId);
    this.currentArray = [];
    this.currentGraph = null;
    this.currentTree = null;
    this.currentStep = null;
    this.visualizationType = 'array';
  }
  initializeArray(array) {
    this.currentArray = [...array]; // Create a copy
    this.visualizationType = 'array';
    this.renderArray();
  }
  initializeGraph(graph) {
    this.currentGraph = graph;
    this.visualizationType = 'graph';
    this.renderGraph();
  // ========== ARRAY VISUALIZATION ==========
  renderArray(highlight = {}) {
    // Clear containers
    this.container.innerHTML = '';
    this.indexContainer.innerHTML = '';
    // Create array elements and indices
    this.currentArray.forEach((value, index) => {
      // Create array element (box with number)
      const element = document.createElement('div');
      element.className = 'array-element';
```

```
            element.textContent = value; // This shows the actual number
            element.dataset.index = index;
            element.dataset.value = value;
            // Apply highlighting based on current step
            this.applyArrayHighlight(element, index, highlight);
            this.container.appendChild(element);
            // Create index element
            const indexElement = document.createElement('div');
            indexElement.className = 'index-item';
            indexElement.textContent = index;
            this.indexContainer.appendChild(indexElement);
        }
      );
    }
    }
     // ========== GRAPH VISUALIZATION ==========
    renderGraph(step = {}) {
        this.container.innerHTML = '';
        this.container.innerHTML = '<div style="text-align: center; padding: 20px; color: #64748b;">Graph Visualization - Coming Soon</div>';
    }
    // ========== TREE VISUALIZATION ==========
    renderTree(step = {}) {
        this.container.innerHTML = '';
        this.container.innerHTML = '<div style="text-align: center; padding: 20px; color: #64748b;">Tree Visualization - Use BST for tree operations</div>';
    }
    // ========== MAIN VISUALIZATION METHOD ==========
    visualizeStep(step) {
        this.currentStep = step;
            if (step.array && this.visualizationType === 'array') {
            this.currentArray = [...step.array]; // Update with the actual swapped array
        }
        switch (this.visualizationType) {
            case 'array':
                this.renderArray(step.highlight || {});
                break;
            case 'graph':
                this.renderGraph(step);
                break;
            case 'tree':
                this.renderTree(step);
                break;
        }
    // Method to get current visualization state
    getVisualizationState() {
        return {
            array: [...this.currentArray],
            graph: this.currentGraph ? JSON.parse(JSON.stringify(this.currentGraph)) : null,
            tree: this.currentTree ? [...this.currentTree] : null,
```

```
            step: this.currentStep,
            type: this.visualizationType,
            container: this.container,
            indexContainer: this.indexContainer
        };
    }
}
// BST Visualization Controller
class BSTVisualizer {
    constructor(containerId) {
        this.container = document.getElementById(containerId);
        if (!this.container) {
            console.error('BST container not found:', containerId);
            return;
        }
        this.canvas = document.createElement('canvas');
        this.ctx = this.canvas.getContext('2d');
        this.container.appendChild(this.canvas);
        this.currentBST = { root: null, size: 0 };
        this.highlightedNodes = new Set();
        this.currentOperation = null;
        this.nodePositions = new Map();
        this.setupCanvas();
        this.setupEventListeners();
    }
    setupEventListeners() {
        // Add any event listeners needed for BST interactions
        this.canvas.addEventListener('click', (e) => this.handleCanvasClick(e));
    }
    initializeBST(bst) {
        this.currentBST = bst;
        this.highlightedNodes.clear();
        this.render();
    }
    visualizeStep(step) {
        if (!step) return;
        this.currentOperation = step.type;
        this.highlightedNodes.clear();
        // Update current BST if provided in step
        if (step.bst) {
            this.currentBST = step.bst;
        }
        // Handle different step types
        if (step.current) {
            this.highlightedNodes.add(step.current);
        }
        if (step.inserted) {
            this.highlightedNodes.add(step.inserted);
        }
```

```
    if (step.found) {
        this.highlightedNodes.add(step.found);
    }
    if (step.visited) {
        this.highlightedNodes.add(step.visited);
    }
    if (step.comparing) {
        this.highlightedNodes.add(step.comparing);
    }
    this.render();
}
     calculatePositions() {
    const nodePositions = new Map();
    const levelHeight = 80;
    if (!this.currentBST || !this.currentBST.root) {
        this.nodePositions = nodePositions;
        this.maxDepth = 0;
        return;
    }
    const calculate = (node, depth, minX, maxX) => {
        if (!node) return null;
        const x = (minX + maxX) / 2;
        const y = depth * levelHeight + 60;
        nodePositions.set(node.value, { x, y, node, depth });
        // Calculate positions for children with proper spacing
        const childWidth = (maxX - minX) / 2;
        if (node.left) {
            calculate(node.left, depth + 1, minX, x - 10);
        }
        if (node.right) {
            calculate(node.right, depth + 1, x + 10, maxX);
        }
    };
    calculate(this.currentBST.root, 0, 50, this.canvas.width - 50);
    this.nodePositions = nodePositions;
}
// ADDED: Draw the complete tree
drawTree() {
    // Draw edges first (so they appear behind nodes)
    this.drawEdges();
    // Then draw nodes (so they appear on top)
    for (const [value, pos] of this.nodePositions) {
        this.drawNode(pos);
    }
}
        // Draw edge to right child
        if (node.right) {
            const rightPos = this.nodePositions.get(node.right.value);
            if (rightPos) {
                this.ctx.beginPath();
```

```
                this.ctx.moveTo(x, y);
                this.ctx.lineTo(rightPos.x, rightPos.y);
                this.ctx.stroke();
            }
        }
        }
        }
    drawNode(pos) {
        const { x, y, node } = pos;
        const radius = 20;
        // Determine node color based on current operation and highlighting
        let fillColor = '#475569'; // Default gray
        let borderColor = '#64748b';
        }
        // Draw node circle
        this.ctx.fillStyle = fillColor;
        this.ctx.strokeStyle = borderColor;
        this.ctx.lineWidth = 2;
        this.ctx.beginPath();
        this.ctx.arc(x, y, radius, 0, 2 * Math.PI);
        this.ctx.fill();
        this.ctx.stroke();
        // Draw node value
        this.ctx.fillStyle = '#ffffff';
        this.ctx.font = 'bold 12px Arial';
        this.ctx.textAlign = 'center';
        this.ctx.textBaseline = 'middle';
        this.ctx.fillText(node.value.toString(), x, y);
    }
if (typeof module !== 'undefined' && module.exports) {
    module.exports = { Visualizer, AnimationController, BSTVisualizer };
}
```

## VII. ACKNOWLEDGEMENT

It is one of the most efficient tasks in life to choose the appropriate words to express one's gratitude to the beneficiaries. We are very much grateful to God who helped us all the way through the project and how molded us into what we are today.

We are grateful to our beloved Principal Dr. R. RADHAKRISHNAN, M.E., Ph.D., Adhiyamaan College of Engineering (An Autonomous Institution), Hosur for providing the opportunity to do this work in premises.

We acknowledge our heartful gratitude to Dr. G. FATHIMA, M.E., Ph.D., Professor and Head of the Department, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Hosur, for her guidance and valuable suggestions and encouragement throughout this project and made us to complete this project successfully.

We are highly indebted to Mrs. A. LAVANYA, M.E., Supervisor, Assistant Professor, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Hosur, whose immense support encouragement and valuable guidance were responsible to complete the project successfully.

We also extend our thanks to Project Coordinator and all Staff Members for their support in complete this project successfully.

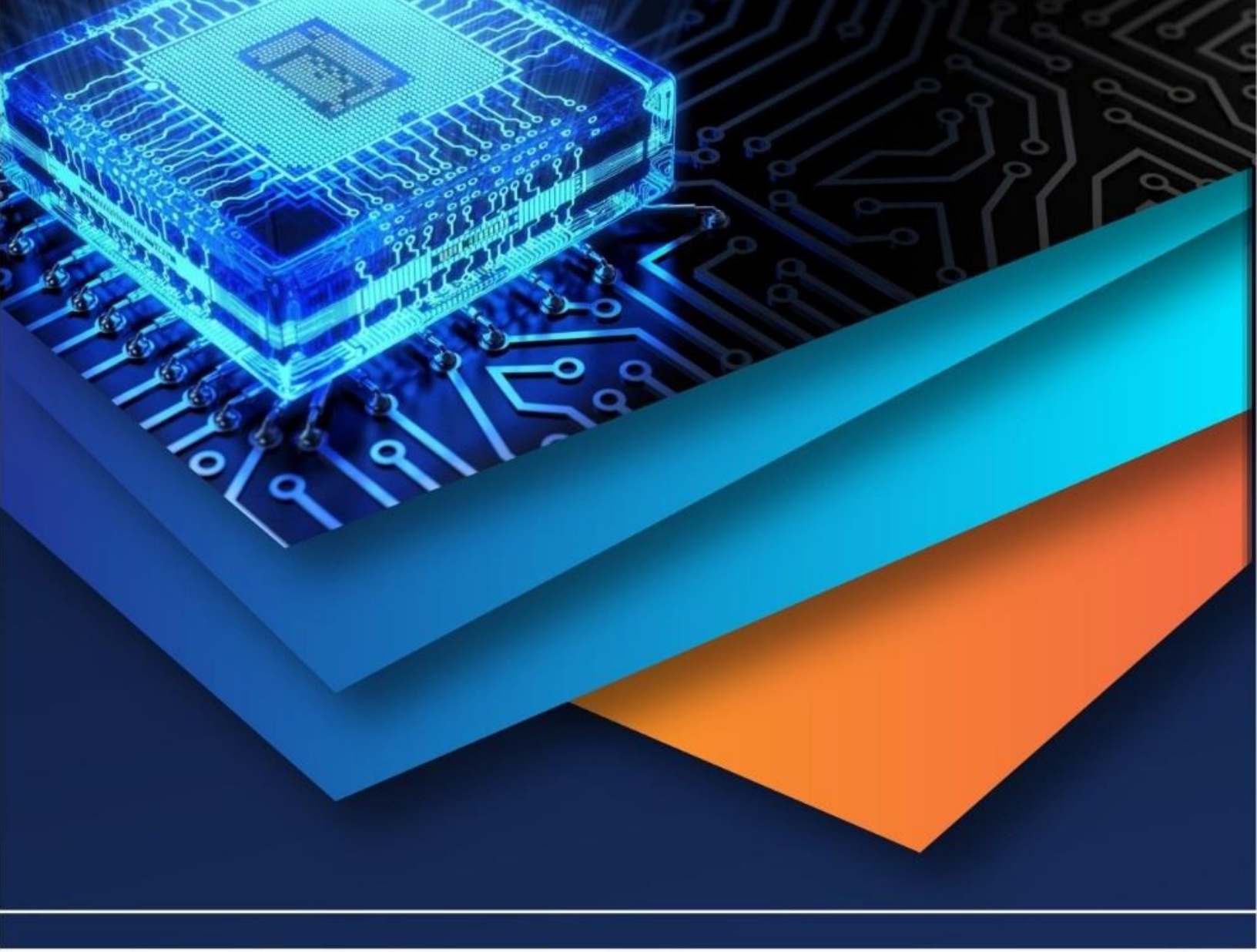
Finally, we would like to thank to our parents, without their motivational and support would not have been possible for us to complete this project successfully.

## REFERENCES

[1] S. R. Bhandari et al., "Algorithm Visualization on Data Structures," Journal of Computer Science Education, 2025.

[2] F. Vateha and S. Simoňák, "Comparative Visualization of Algorithms and Data Structures," International Journal of Educational Technology in Computer Science, 2025.

[3] K. R. Sharma and P. Verma, "Immersive Algorithm Animation in Augmented Reality for Data Structure Learning," International Journal of Interactive Multimedia and Artificial Intelligence, 2025.
[4] L. N. Patel, M. Q. Huang & S. Lee, "Visual Trace Mining: An Interactive Tool for Code-and-Data Structure Visualization in Real Time," Journal of Computer Science Education Research, 2025.
[5] J. A. Roberts, E. P. Gómez & R. Singh, "From Pseudocode to Visualization: A Gamified Approach to Algorithm Teaching," Journal of Educational Technology & Society, 2025.
[6] T. W. Kim, A. Brown & V. Singh, "Comparative Study of 3D and 2D Algorithm Visualizations in VR Environments for Undergraduate Learning," Computers & Education, 2025.
[7] H. Z. Ali, S. Munir & D. H. Lee, "Adaptive Visualization Framework for Dynamic Programming Using AI-Driven Interactive Dashboards," IEEE Access, 2025.
[8] S. R. Bhandari, A. Gaikwad, A. Huang, F. Vateha et al., "Modern Perspectives on Algorithm Visualization Techniques," IEEE Transactions on Learning Technologies, 2025.
[9] A. Gaikwad et al., "Visualizing Complexity: Navigating Algorithms with Algorithm Visualizer," ACM SIGCSE Bulletin, 2024.
[10] SIGCSE Demo Team, "Demo 3C: Algovision – An Algorithm Visualization Tool," Proceedings of the ACM Conference on Computer Science Education (SIGCSE), 2024.
[11] G. Kogan, H. Chassidim, and I. Rabaev, "The Efficacy of Animation and Visualization in Teaching Data Structures: A Case Study," Computer Applications in Engineering Education, 2024.
[12] D. H. Lee et al., "dpvis: A Visual and Interactive Learning Tool for Dynamic Programming," International Journal of Computing and Informatics, 2024.
[13] A. Ge Zhang et al., "CFlow: Supporting Semantic Flow Analysis of Students' Code in Programming Problems at Scale," IEEE Transactions on Learning Technologies, 2024.
[14] Y. Ye et al., "Generative AI for Visualization: State of the Art and Future Directions," Artificial Intelligence Review, 2024.
[15] R. Mourato and A. L. Santos, "Educational Program Visualizations Using Synthesised Execution Information," Journal of Computational Education, 2024.
[16] S. Chawla and S. Jindal, "Algorithm Visualization: Bridging the Gap Between Theory and Practice," International Journal of Computer Science Education, 2024.
[17] A. S. M. Venigalla et al., "Using Augmented Reality for Visualizing Control Flows in Programs," IEEE Access, 2023.
[18] M. Paredes-Velasco et al., "Augmented Reality with Algorithm Animation and Their Effect," Journal of Educational Multimedia and Hypermedia, 2023.
[19] B. E.-Okikere Ojugo, C. Ugwu, and L. U. Oghenekaro, "Visualization Tool for Data Structures in Real Time," Journal of Computer Applications Research, 2023.
[20] A. Huang et al., "Visualization of Sorting Algorithms in the Virtual Reality Environment," IEEE Transactions on Visualization and Computer Graphics, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY