IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Path-Finding And Sorting Algorithms Visualizer

Bhoomika E
Department of Information Science and Engineering
Global Academy of Technology, line 4: Bengaluru, India

Chahna G
Department of Information Science and Engineering
Global Academy of Technology
Bengaluru, India

Hithashree H S
Department of Information Science and Engineering
Global Academy of Technology
Bengaluru, India

Neha H S
Department of Information Science and Engineering
Global Academy of Technology
Bengaluru, India

Guided by: Manjula B S
Department of Information Science and Engineering
Global Academy of Technology
Bengaluru, India

***Abstract*—** Learning how algorithms work can be difficult when the explanation is limited to written code or theory. To make this easier, this project provides a web-based tool that shows the actions of different path-finding and sorting algorithms through clear, step-by-step visuals. The system uses simple colours and animations to display how nodes are visited, how choices are made during the search process, and how values are arranged while sorting. It supports widely used algorithms such as BFS, DFS, A*, Dijkstra's Algorithm, Bubble Sort, Merge Sort, Quick Sort, and others. The tool is built using standard web technologies and can run on any modern browser. Its main goal is to help learners understand algorithm behaviour more clearly and to support improved digital learning in line with educational and technological development goals.

***Keywords*—**Algorithm Visualization, Path-Finding Algorithms, Sorting Algorithms, Dijkstra's Algorithm, A* Search, Breadth-First Search (BFS), Depth-First Search (DFS), Bubble Sort, Quick Sort, Merge Sort, Educational Tool, Interactive Learning, Web-Based Application, HTML, CSS, JavaScript, SDG 4, SDG 9.

I. INTRODUCTION

Algorithms are an important part of computer science because they provide clear steps for solving different types of problems. They are used in many areas, such as data processing, networking, and artificial intelligence. Although algorithms are widely discussed, many learners find them difficult to understand when they are explained only through code, diagrams, or theory. These methods show the logic behind an algorithm, but they do not clearly show how values change or how decisions are made during execution. To make algorithm learning easier, this project introduces a web-based tool that shows how path-finding and sorting algorithms work through simple visual animations. The system uses colours and movement to display each step of an algorithm,helping learners see how nodes are visited, how comparisons are made, and how values are arranged. This makes it easier for users to understand the behaviour of algorithms instead of only reading about them.

The tool covers two main types of algorithms. The first type includes path-finding methods like Dijkstra's Algorithm, A* Search, Breadth-First Search (BFS), and Depth-First Search (DFS), which are used in areas such as navigation and robotics. The second type includes sorting methods like Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort, which are commonly used in data organization. Users can change settings such as grid size, data size, and animation speed to study how each algorithm works. This interactive approach helps learners see performance differences and understand key concepts more clearly. The project also supports the goals of improving digital learning and promoting technological development, which align with SDG 4 and SDG 9.

Overall, the Path-Finding and Sorting Algorithms Visualizer helps connect theoretical knowledge with practical understanding. By watching the algorithms run step by step, learners can better understand their logic, behaviour, and performance. This paper explains the design, implementation, and educational use of the system.

II. MAINTAINING THE INTEGRITY OF THE SPECIFICATIONS

The System is designed to give users a clear and constant visual experience across all algorithms. Each algorithm uses the same color pattern to show different states such as visited, unvisited, active and completed. This helps users understand the animations without confusion.

The tool also follows simple design rules such as fixed graph layouts, preset animation speeds and clear displays for execution time and the number steps. These rules ensure that the results are reliable and easy to compare. Since the tool works fully inside a web browser, it runs smoothly on many devices without any installation of any software. This makes the system convenient and more accessible for users.

III. ABBREVIATIONS AND ACRONYMS

The project uses the some of the abbreviations that are as follows:

For example, in this paper:

- **BFS** stands for *Breadth-First Search*.
- **DFS** stands for *Depth-First Search*.
- **A*** stands for *A-star Algorithm*.
- **IDE** stands for *Integrated Development Environment*.
- **UI** stands for *User Interface*.
- **SDG** stands for *Sustainable Development Goal* (specifically SDG 4: Quality Education and SDG 9: Industry, Innovation, and Infrastructure).

A. Units

Units Used in the Project

To ensure clarity and consistency, the project applies uniform measurement units when assessing the performance of different algorithms. Using the same unit system throughout the analysis improves readability and supports accurate comparisons.

• **Time:** Execution time is measured in milliseconds (ms). For example, the duration of a sorting operation or the time taken to complete a pathfinding task may be shown as "145 ms."
• **Memory:** Storage usage is stated in megabytes (MB), representing the amount of system memory consumed during execution.
• **Input Size:** For sorting algorithms, the dataset length is represented by the variable n.
• **Performance Indicators:** The tool displays algorithmic complexity using standard notations, including time complexity (such as O(n) or O(n log n)) and space complexity (such as O(1) or O(n)).
• **Display Specifications:** The interface is optimized for common screen dimensions—typically 1280 × 720 pixels or above—to achieve clear and stable visual output.

All quantitative values and metrics follow standard SI or computing conventions. To avoid confusion, the document maintains a single unit format within any given section, preventing inconsistencies such as mixing seconds and milliseconds.

B. Equations

In this project, a few basic mathematical expressions are used to explain how different algorithms estimate cost and performance. These equations help compare their behaviour and understand why they make certain decisions during execution.

1. Path-Finding Cost Estimation

Some path-finding algorithms use a cost value to decide which node to explore next. A* uses two values:

- the actual cost from the start to the current node, and
- an estimated cost from the current node to the goal.

Dijkstra's Algorithm uses only the actual cost because it does not rely on any estimate.

2. Sorting Algorithm Complexity

Sorting algorithms are often compared based on how their running time increases with larger inputs. Some take more time for bigger datasets, while others handle growth more efficiently. These complexity levels help explain why some methods finish sooner than others.

Some commonly observed complexities are:

- **Bubble Sort:** $O(n^2)$
- **Selection Sort:** $O(n^2)$
- **Insertion Sort:** $O(n^2)$
- **Merge Sort:** $O(n \log n)$
- **Quick Sort:** $O(n \log n)$in average cases
- **Heap Sort:** $O(n \log n)$

These expressions help explain why certain algorithms perform faster than others, especially with large inputs.

3. General Performance Expression

A simple expression can be used to represent approximate running time. The constants in the model depend on system speed, input size, and how the algorithm is written. This is not an exact measure but helps compare behaviours in a basic way.

T(n)=an+bnlogn+c

where:

- a, b, and $c$are constants based on the system, input type, and implementation
- $n$is the size of the input
- The formula gives an *approximate comparison*, not the exact execution time

C. Some Common Mistakes

During the development and writing of this project, a few issues were identified that can affect clarity:

- **Incorrect order of steps:** Algorithm actions must follow their real sequence to avoid confusion.
- **Wrong complexity notation:** Time and space complexity symbols should be used correctly and only when relevant.
- **Inconsistent colours:** Visual states should use the same colour meanings throughout the tool.
- **Mixing theory and results:** Analytical explanations should be separate from observations made during testing.
- **Different units:** All measurements should use the same unit within a section to avoid misunderstanding.
- **Lack of testing:** Algorithms should be verified with known examples before drawing conclusions.
- **Complex interface choices:** Visual components should remain simple and easy to understand.
- **Incorrect citations:** References must follow the correct order and format.

IV.MOTIVATION

The motivation for this project comes from the observation that learners often have a hard time turning theoretical algorithm steps into working mental models. Just reading code or pseudocode doesn't clearly show how variables change, how nodes are visited, or how sorting sequences progress. This visualizer was created to make those abstract changes visible, engaging, and easy to follow.

By using colour coding, movement, and step-by-step animation, the system tackles one of the biggest challenges in teaching algorithms. It keeps learners engaged while explaining complex logic. It also encourages exploration, allowing users to experiment, make mistakes, and see the results right away. The ultimate aim is to improve both understanding and retention of essential computational concepts.

V.OBJECTIVES

The primary objective of this project is to develop a **user-friendly and interactive platform** that visually demonstrates how commonly used algorithms operate. Through **real-time simulations**, the system presents each algorithm's decision points, comparisons, and logical steps for both **sorting** and **path-finding** techniques. This visual clarity enables users to analyse and compare the efficiency of algorithms based on factors such as **execution time**, **number of operations**, and **computational complexity**.

Another key objective is to align the project with **modern teaching methodologies** that emphasize experiential and visual learning. By providing **instant visual feedback**, the system helps learners identify patterns, detect inefficiencies, and understand the performance trade-offs between different algorithms. This interactive approach simplifies complex computational concepts, making them easier to comprehend and offering a more engaging and effective learning experience.

VI.PROJECT-REQUIREMENTS

The project needs minimal hardware to be accessible to many users. A computer with an Intel i3 processor, 4 GB of RAM, and a display resolution of 1280 × 720 pixels or higher can run the tool smoothly. This low demand for computers makes the visualizer suitable for both classroom and personal use without needing special equipment.

On the software side, the system was built using web technologies like HTML, CSS, and JavaScript. This approach ensures it works on different platforms and is easy to deploy. It can run directly in modern browsers such as Google Chrome, Mozilla Firefox, or Microsoft Edge. Using open-source frameworks helps with future improvements and scalability.

A. Hardware-Requirements

The application operates efficiently on standard computing devices. For optimal performance, a minimum of an **Intel Core i3 processor**, **4 GB RAM**, and **500 MB of free disk space** is recommended. A **1280 × 720 resolution** display ensures accurate rendering of grid and bar elements. Although dedicated GPUs enhance rendering speed, the tool functions effectively on integrated graphics.

B. Software-Requirements

The system is developed using **HTML**, **CSS**, and **JavaScript**, ensuring cross-browser compatibility with **Google Chrome**, **Mozilla Firefox**, and **Microsoft Edge**. **Visual Studio Code (VS Code)** serves as the primary IDE, offering real-time debugging and preview functionality.
The application architecture follows a **modular design**, separating algorithmic logic from visualization layers. Optional libraries such as **Chart.js** can be integrated for automated performance graph generation. As a browser-based tool, the system supports **Windows**, **Linux**, and **macOS** without installation requirements.

VII.EXPECTED-OUTCOME

The project aims to create a fully functional visualization system that can animate different algorithms in real time. It will help learners see how the algorithms work, review performance metrics, and identify algorithmic complexities such as $O(n^2)$ and $O(n \log n)$. By interacting with the system, users will feel more confident in grasping algorithmic principles.

More generally, this project hopes to bring an educational breakthrough that makes learning about algorithms more engaging and effective. The tool will act as an open platform for additional research and classroom use. It will allow instructors to present concepts in a dynamic way instead of relying on static examples.

VIII.APPLICATIONS

This visualizer has many academic and practical uses. In schools, it can serve as a teaching tool to show how different algorithms work with various datasets. Students can test their predictions, review outcomes, and improve their understanding through quick feedback.

Outside the classroom, the system can help developers debug or optimize algorithms by tracking their step-by-step execution. It can also be used in workshops, coding boot camps, and research labs for testing algorithms and comparing performance.

IX.RESULTS AND PERFORMANCE METRICS

The performance of the sorting algorithms was checked using three main factors: the number of comparisons, the number of swaps, and the time taken to complete the task. All algorithms were tested with the same input so that the results would be fair and easy to compare.

The algorithms with $O(n^2)$complexity, such as Bubble Sort, Selection Sort, and Insertion Sort, required more steps and generally took longer to finish. Algorithms with $O(n\log n)$complexity—such as Merge Sort, Quick Sort, and Heap Sort—worked faster and handled the data more efficiently. Although Quick Sort made more swaps in some cases, it still completed sooner because of the way it divides the input. Merge Sort showed stable performance because it splits the data evenly, while Heap Sort showed balanced behaviour in both swaps and comparisons.

The visualization also helped show that real performance depends on more than just the theoretical complexity. Input order, data patterns, and implementation details can all affect the results. These observations help learners understand how different algorithms behave in practical situations.

1)Algorithm Execution Flow

The execution sequence of each algorithm is built to reflect its theoretical steps while remaining clear for learners to follow. In the case of sorting routines, every comparison and swap is shown directly through bar colour changes and shifting positions on the screen. This helps users observe the internal decisions of the algorithm as they occur, one action at a time. For path-finding routines, the system highlights each node as it is discovered, visited, or marked as complete, revealing the traversal pattern in real time through animated cues.

This straightforward visual progression allows learners to connect what they see in textbooks—such as pseudocode and flowcharts—to the actual behaviour displayed in the visualizer. The structure is modular, meaning each algorithm is implemented as an independent function that plugs into the overall visualization environment. This modular approach simplifies testing, debugging, and the addition of new algorithms. It also ensures that every animation follows the correct sequence of operations, preventing confusion between theoretical expectations and on-screen activity.

2)Performance Parameters

The execution flow of each algorithm is designed to be both easy to understand and faithful to its theoretical logic. In sorting visualizations, every comparison and swap is represented through changes in the colour and position of bars, allowing users to track the decision-making process in real time. Similarly, in path-finding visualizations, each node is highlighted as it is explored, visited, or finalized, displaying the traversal order dynamically.

This clear visual representation helps learners connect what they observe on screen with the pseudocode and theory they study in textbooks. The system follows a modular design, where each algorithm operates independently within the visualization framework. This modularity simplifies debugging, testing, and the addition of new algorithms in the future. Moreover, it ensures that every animation accurately reflects the logical sequence of operations, removing any confusion between expected outcomes and the actual process being visualized.

3)Visualization Interface

The visualization interface connects algorithm logic to user perception. It has a grid or bar layout, color-coded states, and control elements like play, pause, and reset buttons. Each colour represents a specific algorithmic state, such as active, visited, sorted, or final. This makes complex transitions easier to understand. Animations are timed for clarity instead of speed, ensuring that each step is visible long enough to comprehend.

To make it more accessible, the interface has adjustable sliders for speed control and size selection, along with responsive scaling for different screen resolutions. The design focuses on simplicity and minimalism, avoiding clutter while keeping all necessary information in view. These design choices turn abstract algorithmic operations into an intuitive, interactive experience.

4)Data Input Options

The visualizer lets users customize input data before running any algorithm. For sorting, users can create random arrays, input custom numeric sequences, or choose between ascending and descending orders. For path-finding, users can set grid dimensions, manually select start and goal nodes, and create obstacles by clicking on cells. These options encourage experimentation and provide a hands-on learning environment.

This flexibility supports a wide range of educational and research applications. Students can test how different input distributions affect sorting efficiency. Researchers can observe how graph density impacts search algorithms. The ability to adjust and reset input instantly promotes an iterative learning process: observe, modify, and re-run. This makes the visualizer an effective teaching tool.

X. FIGURES AND TABLES

Fig 1.1 bubble sort

This figure shows the interface of the sorting module. The upper part of the screen includes controls such as an input box for entering values, an algorithm selection menu, and a slider to adjust the animation speed. The user can change these settings before starting the visualization. The lower part displays vertical bars that represent the data elements. As the Bubble Sort algorithm runs, the bars change colour and switch positions to show comparisons and swaps. This helps users clearly follow how the algorithm processes the data step by step.

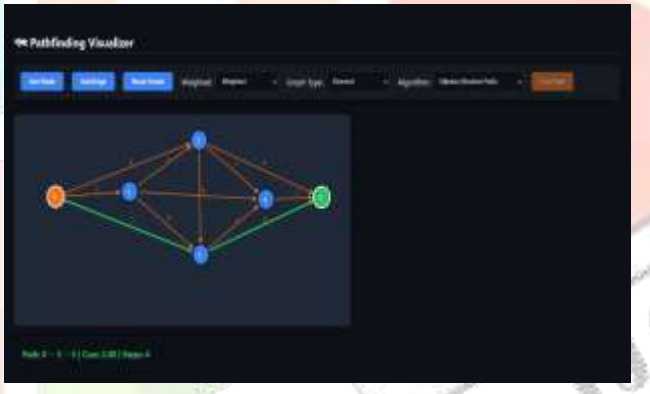

Fig 1.2 Dijkstra's pathfinding

This figure shows how the path-finding feature works. Users can add or remove nodes and connect them with edges. In this example, Dijkstra's Algorithm is used to find the shortest route between two points. The start and end nodes are shown in different colours. As the algorithm runs, the tool highlights the nodes and edges it checks. After the search finishes, the shortest path and its total cost are displayed. This helps users clearly see how the algorithm moves through the graph.

Table:

Table 1.1 Performance comparison of sorting algorithms in the visualization tool.

Algorithm	*Comparisons/sort*	*Time(ms)*	*Swaps*	*Complexity*
Bubble sort	*15*	*0.10*	*3*	*O(n²)*
Selection sort	*15*	*0.00*	*1*	*O(n²)*
Insertion sort	*3*	*0.00*	*3*	*O(n²)*
Merge sort	*8*	*0.10*	*0*	*O (n log n)*
Quick sort	*9*	*0.00*	*11*	*O (n log n)*
Heap sort	*8*	*0.10*	*13*	*O (n log n)*

Table I shows the performance of six sorting algorithms: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, and Heap Sort. The table includes the number of comparisons, the time taken in milliseconds, the number of swaps, and the expected time complexity for each algorithm. All results were obtained using the Sorting Visualizer, which tested every algorithm on the same input to make the comparison fair.

The first column lists the algorithm names. The next columns show how many comparisons were made, how long the algorithm took to finish, and how many swaps were required. The last column gives the theoretical Big-O complexity, which indicates how the algorithm behaves as the input size increases.

The results show that Bubble Sort, Selection Sort, and Insertion Sort have $O(n^2)$complexity, so they take longer on larger inputs. Merge Sort, Quick Sort, and Heap Sort follow $O(n\log n)$, allowing them to perform better. Quick Sort has more swaps but still runs quickly because of its efficient splitting method. Merge Sort finishes with no swaps, while Heap Sort maintains a balanced number of comparisons and swaps. These results match common expectations and show that the visualizer presents algorithm performance accurately.

System Architecture

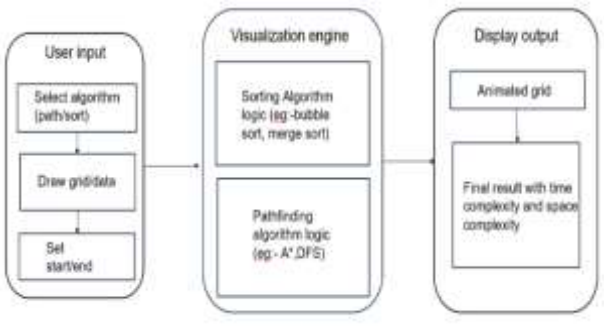

Figure 3.5 System Architecture

The user input section allows learners to choose the type of algorithm they want to run, such as sorting or path-finding. They can also set basic options like grid size, array size, and the start and end points for path-finding. The system checks the inputs to make sure they are valid so the results shown are accurate.

The core of the tool is the Visualization Engine. It runs each algorithm step by step and sends the updates to the display module. This includes actions like swaps, comparisons, and node exploration. The engine also allows users to pause, resume, or change the speed of the animation at any time.

The display module shows these steps through simple animations and colour changes. For sorting algorithms, bars change height and colour based on the operations being performed. For path-finding, the system highlights visited nodes, active edges, and the final path. The modular structure makes it easy to add new algorithms in the future without changing the whole system.

XI. Future Scope

The visualization system in this project can be improved in many ways in the future. One important improvement is adding more algorithms to the tool. Including methods such as Floyd–Warshall, Bellman–Ford, Radix Sort, Counting Sort, Shell Sort, and Bucket Sort would give users more options to test and compare. With more algorithms, learners can study how different techniques work and how their speed and memory use change with different inputs.

Another useful addition would be an automated analytics feature. This could show charts and live performance data while an algorithm runs. Such information would help users understand how input size affects execution time and system usage. It would also make the tool more helpful for research and detailed analysis.

Machine learning could also be added to make the tool adapt to the user. By observing how a user interacts with the system, the tool could give personalized suggestions, adjust animation speed automatically, or create custom learning paths. This would help learners understand difficult concepts at their own pace. Making the tool available on more platforms is another strong area for improvement. Converting the web version into a Progressive Web App (PWA) or building desktop and mobile versions would allow more people to use the tool easily. Adding offline access would help learners who do not always have internet connectivity.

Collaboration features could also be added. A shared workspace or classroom mode would allow multiple users to view and control the same visualization together. Connecting the tool with Learning Management Systems (LMS) such as Google Classroom or Moodle would help teachers assign tasks, track progress, and manage learning activities. Adding gamification could make learning more enjoyable. Small challenges, quizzes, badges, or points could motivate users to try more algorithms and explore the tool more deeply.

Future versions may also include 3D views or Augmented Reality (AR) to make the learning experience more interactive. These technologies can help students see algorithm behaviour in a clearer and more engaging way. Adding language options and accessibility settings would also make the tool more friendly to a global audience. The tool can also grow beyond algorithms by including visualizations of data structures like stacks, queues, linked lists, trees, and graphs. This would help learners study more computer science concepts in one place. An AI-based explanation feature could also provide automatic hints, summaries, or simplified pseudocode.

Finally, making the tool open-source would allow students, educators, and developers from different places to contribute. This would support continuous improvement and help the tool evolve into a widely used platform for algorithm education.

XII. CONCLUSION

The Path-Finding and Sorting Algorithms Visualizer helps users understand algorithms by showing their steps through simple animations and clear colour changes. This makes it easier for learners to see how decisions are made and how data changes during execution, improving their understanding of both sorting and path-finding processes.

During development, challenges such as keeping the animations in sync with the algorithm steps, supporting large inputs, and maintaining smooth performance in the browser were addressed. Solving these issues made the tool more stable and effective.

This project shows the importance of visual learning in computer science. When learners can watch algorithms run step by step, they gain a clearer understanding of ideas like node traversal, comparisons, and data movement. Because the tool works in any modern browser, it is accessible to many users and supports goals related to better digital learning and technological progress.

The system is designed so that new algorithms and features can be added later. This makes the visualizer a useful and flexible tool for students, teachers, and anyone who wants to learn algorithms in a simple and interactive way.

References

[1] International Journal of Creative Research Thoughts (IJCRT), vol. 11, no. 7, Jul. 2023, ISSN2320-2882.

[2] A. Kulkarni, S. Padave, S. Shrivastava, and V. Kawtikwar, "Algorithm Visualizer," International Journal for Research in Applied Science and Engineering Technology (IJRASET), vol. 11, no. 7, pp. 1818–1823, Jul. 2023.[Online].Available:https://doi.org/10.22214/ijraset.2023.54837.

[3] International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE), vol.10, no.12, Dec 2021.

[4] M. Mukasheva, Z. Kalkabayeva, and N. Pussyrmanov, "Visualization of sorting algorithms in the virtual reality environment,one space "Frontiers in Education, vol.8, 2023.Available:https://doi.org/10.3389/feduc.2023.1195200

[5] "A Comprehensive Guide with the ISort Visualizer," International Journal of Innovative Science and Applied Engineering (IJISAE), 2024. [Online]. Available: https://ijisae.org/index.php/IJISAE/article/download/6645/5504/11824

[6] "Review Paper on Algorithm Visualizer," International Research Journal of Modernization in Engineering, Technology and Science (IRJMETS), vol. 4, no. 11, Nov.2022.[Online]Available:https://www.irjmets.com/uploadedfiles/paper/issue_11_november_2022/31719/final/fin_irjmets1669731512.pdf

[7] M. Bargir, K. Yedre, P. Gajane, G. Sawant, and M. Gore, "Sorting Algorithm Visualizer," International Journal of Creative Research Thoughts (IJCRT), vol. 12, no. 4, pp. b266-b273, Apr. 2024. [Online]. Available: https://www.ijcrt.org/papers/IJCRT2404143.pdf

[8] N. Rupavatiya, J. Darji, H. Moradiya, D. Chanchad, and J. Saturwar, "Algorithm Visualizer," Journal of Emerging Technologies and Innovative Research (JETIR), [Online]. Available:https://www.jetir.org/papers/JETIR2304522.pdf