Kathmandu University
Department of Computer Science and Engineering
Dhulikhel, Kavre

A Project Report
on
“Campus Navigator : Algorithm Visualization System”

[Code No : COMP 202]

(For partial fulfillment of II Year / I Semester in Computer Engineering)

Submitted by:

Ushma Sapkota (Roll No. 45)

Submitted to:

Mr. Sagar Acharya
Department of Computer Science and Engineering

Submission Date:2026/02/23

Bona fide Certificate

This project work on

“CAMPUS NAVIGATOR : ALGORITHM VISUALIZATION SYSTEM”

is the bona fide work of

“Ushma Sapkota”

who carried out the project work under my supervision.

Project Supervisor

____

Mr. Sagar Acharya,
Lecturer,
Department of Computer Science and Engineering

Date: 2026/02/23

Acknowledgement

I would like to express my sincere gratitude to everyone who contributed to the successful completion of my project, “Campus Navigator : Algorithm Visualization System.”

My deepest appreciation goes to our Data Structures and Algorithms’ lecturer, Mr. Sagar Acharya for providing me with this incredible opportunity. His invaluable support and guidance throughout the development of this project has made it a success. His expertise in data structures and algorithms laid a strong foundation for understanding and implementing the visualization techniques used in this system.

I am grateful to the Department of Computer Science and Engineering (DoCSE) for including the course Data Structures and Algorithms in our syllabus and providing a supportive environment that made this project possible.

I would also like to extend my thanks to my family and friends for their feedback and suggestions during the development process.

This project has been a crucial learning experience that has deepened my understanding of data structures and algorithms.

Sincerely,

Ushma Sapkota

Roll No. 45

Abstract

“Campus Navigator” is an interactive web based algorithm visualization system developed to help students understand and learn fundamental algorithms through visual and practical demonstrations. In the current educational environment, students often struggle to grasp abstract algorithmic concepts through textbooks alone. This project addresses the challenge by providing an engaging platform where users can visualize how pathfinding, searching and slotting algorithms work in real time on a campus map representation.

The system is developed using C++ for the backend implementation and HTML, CSS and JavaScript for frontend. The backend provides algorithm execution data through a HTTP server, while the frontend provides a visualization interface with playback controls. Three algorithms : Dijkstra’s shortest path algorithm for route finding, binary search for building location and quick sort for sorting locations by distance are demonstrated in this system. The system allows users to play, pause and navigate through each stage of the algorithm. The system also includes colour coded nodes and edges, algorithm complexity information, pseudocode and explanations of each step.

In conclusion, the campus navigator transforms abstract concepts into familiar campus navigation scenario which makes understanding of algorithms an engaging learning experience. It demonstrates how theoretical computer science concepts apply to practical real world situations like navigation and optimization. Future enhancements could include additional algorithms such as Breadth First Search, Depth First Search, Bubble Sort, Merge Sort and many more into the system.

Keywords: Algorithm Visualization, C++, Dijkstra’s algorithm, binary search, quick sort, data structures, interactive learning

Table of Contents

List of Figures

.

Acronyms/Abbreviations

HTML : HyperText Markup Language

CSS : Cascading Style Sheets

HTTP : HyperText Transfer Protocol

JSON : JavaScript Object Notation

RAM : Random Access Memory

API : Application Programming Interface

DFS : Depth First Search

BFS : Breadth First Search

UI : User Interface

Chapter 1 **Introduction**

1.1 Background

In today's digital world, understanding algorithms and data structures forms the foundation for becoming a proficient programmer. However, students often find it challenging to grasp how these abstract concepts work in practice. Traditional methods primarily rely on pseudo code, textbooks and static diagrams which fail to capture the dynamic nature of algorithm execution. This gap between the theoretical understanding and the practical application motivated the development of Campus Navigator System.

Campus Navigator is an interactive algorithm visualization system that bridges this gap by demonstrating how the algorithms operate in a familiar real world context. The system uses a campus map as the underlying data structure and transforms it into an engaging educational tool. It shows how Dijkstra's algorithm finds the shortest path between campus buildings, how binary search locates specific locations, and how quick sort organizes locations by distance.

Recent developments in educational technology have emphasized the importance of interactive visual learning tools. Studies show that students retain information better when they actively engage with content rather than passively consume it. Campus Navigator embodies this principle.

Rather than asking students to imagine how an algorithm processes data, the system shows them exactly what happens at each step. Users can see the nodes being visited, distances being updated, comparisons being made, data being rearranged all in real time with full playback control. Each step can be revisited and explanations of every step are shown in the side bar to enhance understanding. It allows users to control the pace of visualization, step through algorithm execution and explore different scenarios by changing the start and end points. The system implements three fundamental algorithms:

a. Pathfinding: Dijkstra’s algorithm demonstrates greedy approaches.
b. Searching: Binary search illustrates divide-and-conquer strategies and logarithmic efficiency.
c. Sorting: Quicksort showcases recursive thinking and in-place manipulation.

Developed using C++ for robust algorithm implementation and modern web technologies (HTML, CSS, JavaScript) for user interface, Campus Navigator represents a practical application of software engineering principles. Campus Navigator serves both as a learning aid for students studying algorithms and as a demonstration of how computer science concepts apply to real world problems we encounter daily.

1.2 Objectives

- To create an interactive visualization platform that demonstrates algorithm execution step-by-step, allowing users to understand not just what algorithms do but how they make decisions at each stage.
- To implement three fundamental algorithms (Dijkstra’s shortest path, binary search and quick sort) with complete step-by-step tracing, capturing all intermediate states, comparisons and modifications for educational transparency using C++.
- To develop a user-friendly interface using HTML, CSS and JavaScript that provides intuitive controls for algorithm playback, including play, pause, step forward, step backward, and speed adjustment.
- To provide contextual learning by implementing algorithms on a meaningful domain (campus navigation) that students can relate to rather than abstract numerical examples.

1.3 Motivation and Significance

The motivation behind developing Campus Navigator arose from my personal experience learning data structures and algorithms. Many students, including myself, could understand the algorithm pseudo code and implement basic versions but struggled to visualize how these algorithms operated on larger datasets.

Additionally, most existing algorithm visualization tools were either too simplistic (showing only basic examples with small datasets) or overwhelmingly complex (professional tools designed for researchers). There was a need for a tool somewhere in between, sophisticated enough to demonstrate real algorithmic behaviour but accessible for undergraduate students to use effectively.

The significance of Campus Navigator extends beyond being another visualization tool. It has following significances:

a. It provides contextual learning by using a campus navigation scenario to turn abstract operations into concrete algorithms.
b. It requires active engagement. Unlike passive consumption of textbooks or video tutorials, it engages users to make decisions on starting and ending points and pace of their learning. Hence, it promotes deeper cognitive processing and better retention.

Ultimately, Campus Navigator aims to make algorithm education more effective and enjoyable.

Chapter 2 Related Works

Over the past few years, some algorithm visualization tools have emerged aiming to help users deepen their understanding of the algorithms. Some of those tools include:

2.1 VisuAlgo

VisuAlgo (visualgo.net), developed by Dr. Steven Halim at the National University of Singapore, is one of the most comprehensive algorithm visualization platforms available. It covers a wide range of algorithms including searching, sorting, graph algorithms, and data structures. VisuAlgo provides step by step animations with detailed explanations and allows users to control the playback speed. The platform has been widely adopted in universities worldwide. While it covers many topics, individual visualizations can feel somewhat generic, sorting abstract numbers and traversing abstract graphs. Campus Navigator, on the other hand, takes a different approach by focusing deeply on fewer algorithms but contextualizing them with a meaningful domain (campus navigation). This can make the learning experience more relatable and memorable for students.

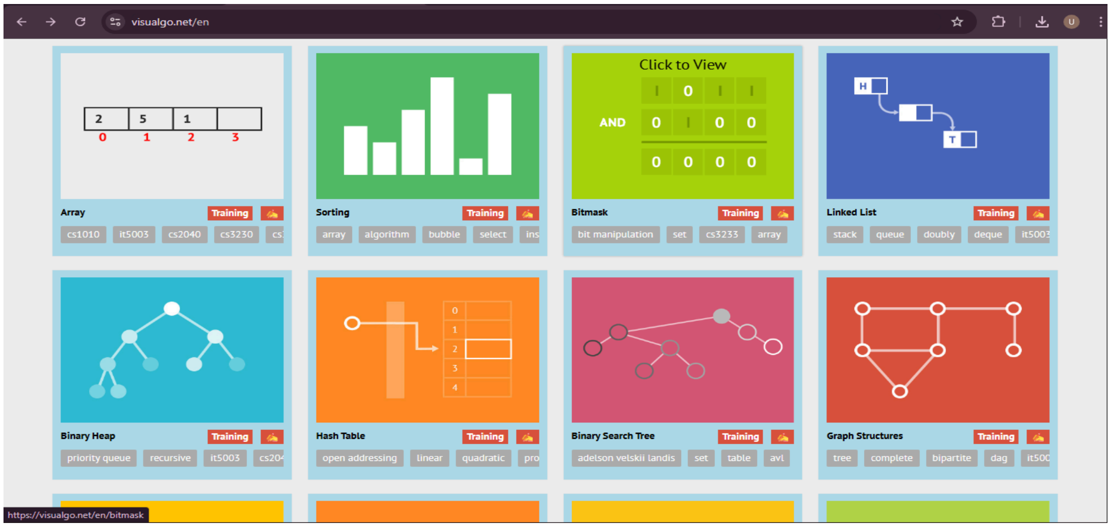
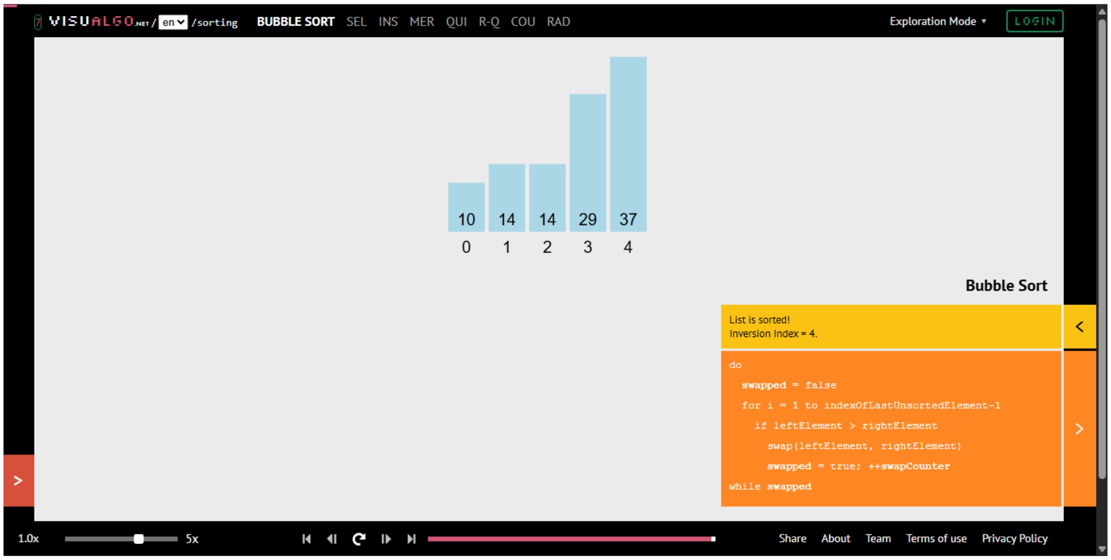

visualgo.net/en
2 5 1
0 1 2 3
Array
Training
cs1010 it5003 cs2040 cs3230 cs
Sorting
Training
array algorithm bubble select ins
Click to View
AND
Bitmask
Training
bit manipulation set cs3233 array
H
T
Linked List
Training
stack queue doubly deque it5003
Binary Heap
Training
priority queue recursive it5003 cs204
0 1 2 3 4
Hash Table
Training
open addressing linear quadratic pro
Binary Search Tree
Training
adelson velskii landis set table avl
Graph Structures
Training
tree complete bipartite dag it500
https://visualgo.net/en/bitmask

Fig 2.1: VisuAlgo's options



Bubble Sort

List is sorted!
Inversion Index = 4.

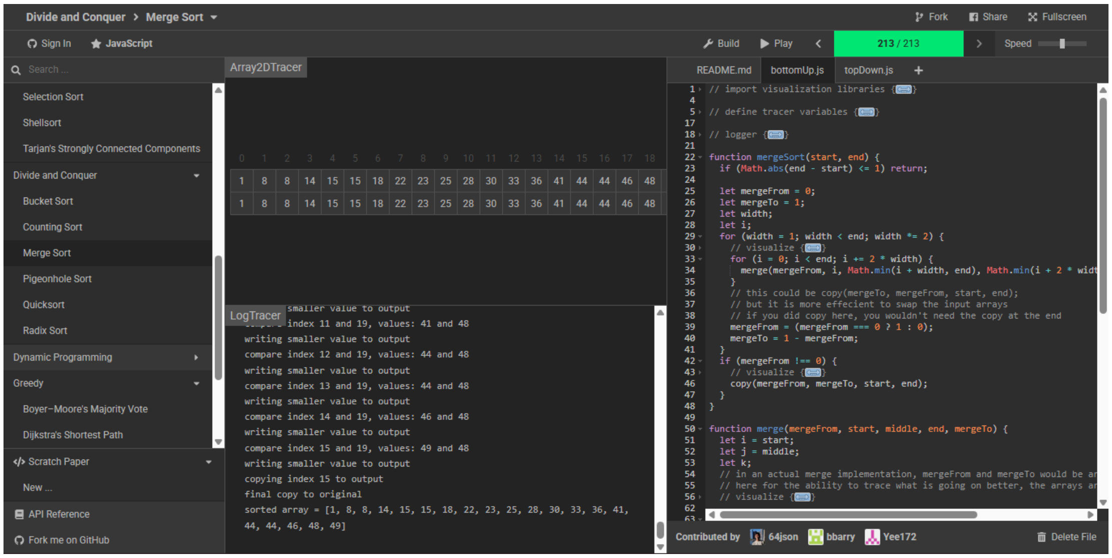
```
do
  swapped = false
  for i = 1 to indexOfLastUnsortedElement-1
    if leftElement > rightElement
      swap(leftElement, rightElement)
      swapped = true; ++swapCounter
while swapped
```

Fig 2.2: VisuAlgo’s Bubble Sort Visualization

2.2 Algorithm Visualizer

Algorithm Visualizer (algorithm-visualizer.org) is another popular open-source project that provides interactive visualizations for various algorithms. It emphasizes code level visualization, showing how code executes line by line alongside the visual representation of data structure changes. This dual presentation helps students connect abstract operations to concrete code.

Campus Navigator adopts a similar principle of but takes it further by providing natural language explanations at each step. Rather than just showing code execution, it explains why certain decisions are made and how the current state leads to the next operation. This approach aims to develop intuitive understanding rather than just procedural knowledge.



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
1	8	8	14	15	15	18	22	23	25	28	30	33	36	41	44	44	46	48
1	8	8	14	15	15	18	22	23	25	28	30	33	36	41	44	44	46	48

```
smaller value to output
        index 11 and 19, values: 41 and 48
writing smaller value to output
compare index 12 and 19, values: 44 and 48
writing smaller value to output
compare index 13 and 19, values: 44 and 48
writing smaller value to output
compare index 14 and 19, values: 46 and 48
writing smaller value to output
compare index 15 and 19, values: 49 and 48
writing smaller value to output
copying index 15 to output
final copy to original
sorted array = [1, 8, 8, 14, 15, 15, 18, 22, 23, 25, 28, 30, 33, 36, 41,
44, 44, 46, 48, 49]
```

```
// import visualization libraries {↔}

// define tracer variables {↔}

// logger {↔}

function mergeSort(start, end) {
  if (Math.abs(end - start) <= 1) return;

  let mergeFrom = 0;
  let mergeTo = 1;
  let width;
  let i;
  for (width = 1; width < end; width *= 2) {
    // visualize {↔}
    for (i = 0; i < end; i += 2 * width) {
      merge(mergeFrom, i, Math.min(i + width, end), Math.min(i + 2 * widt
    }
    // this could be copy(mergeTo, mergeFrom, start, end);
    // but it is more effecient to swap the input arrays
    // if you did copy here, you wouldn't need the copy at the end
    mergeFrom = (mergeFrom === 0 ? 1 : 0);
    mergeTo = 1 - mergeFrom;
  }
  if (mergeFrom !== 0) {
    // visualize {↔}
    copy(mergeFrom, mergeTo, start, end);
  }
}

function merge(mergeFrom, start, middle, end, mergeTo) {
  let i = start;
  let j = middle;
  let k;
  // in an actual merge implementation, mergeFrom and mergeTo would be ar
  // here for the ability to trace what is going on better, the arrays ar
  // visualize {↔}
```

Fig 2.3: Algorithm Visualizer

Chapter 3 Design and Implementation

3.1 Implementation

This chapter provides an overview of the development process for Campus Navigator, from initial planning to documentation.

3.1.1 Planning

The planning phase began with identifying the problem: students struggling to understand algorithm executions through static descriptions. Initially, various algorithm categories were considered: sorting, searching, graph algorithms, tree structures. Given the timeline constraints and goal of depth over breadth, I decided to focus on three fundamental algorithms under campus navigation theme:

-Dijkstra's algorithm (graph traversal, pathfinding)

-Binary search (divide and conquer / searching)

-Quicksort (recursion / sorting)

3.1.2 Requirement Analysis

Requirements were categorized into functional and non-functional requirements. Following functional requirements were identified:

a. **Algorithm Implementation:** Implement Dijkstra's algorithm, binary search and quick sort algorithm and generate detailed execution traces capturing comparisons, swaps and state changes.
b. **Data management:** Define campus graph structure with nodes (buildings) and edges (paths) and track algorithm execution state at each step.
c. **Backend Server:** Implement HTTP server to handle client requests and return JSON formatted step-by-step execution data.

d. **Visualization Interface:** Display interactive campus map on HTML canvas and render colour coded nodes and edges based on states. Display array elements for searching and sorting visualization. Provide real time step explanations and statistics (node visited, comparisons, distance).

e. **User Interaction:** Allow selection of different algorithms via dropdown. Enable choosing start and end locations for Dijkstra, provide search query input for binary search, offer reference location for sorting. Support playback controls and speed adjustments.

Non-functional requirements include:

a. **Usability**: Controls must be clearly labelled, visual design must be clean and professional.

b. **Reliability**: Consistent behaviour across multiple sessions, system must handle invalid inputs.

c. **Maintainability**: Code must be well organized and modular.

d. **Accessibility**: Responsive design for different screen sizes.

3.1.3 System Design

During the system design phase, the conceptual ideas were designed into a blueprint for development. The UI was designed using Figma, which was later developed using HTML, CSS and JavaScript. Similarly, the system architecture and use case diagram were designed.

The use case diagram represents how a student user interacts with the Campus Navigator System. In the diagram, the main actor is the Student User, who can select an algorithm to visualize, such as Dijkstra for pathfinding, Binary Search, or QuickSort. After selecting the algorithm, the user provides the required inputs, like choosing start and end locations, entering a search query, or selecting a reference point. Once the parameters are set, the system executes the algorithm and produces a visualization. The user can then control the visualization using options like play, pause, step forward, step backward, reset, and adjust speed. The system also allows the user to view additional information such as explanations, statistics, time complexity, and pseudocode. Overall, the diagram shows the different actions the student can perform and how the system responds to those actions.

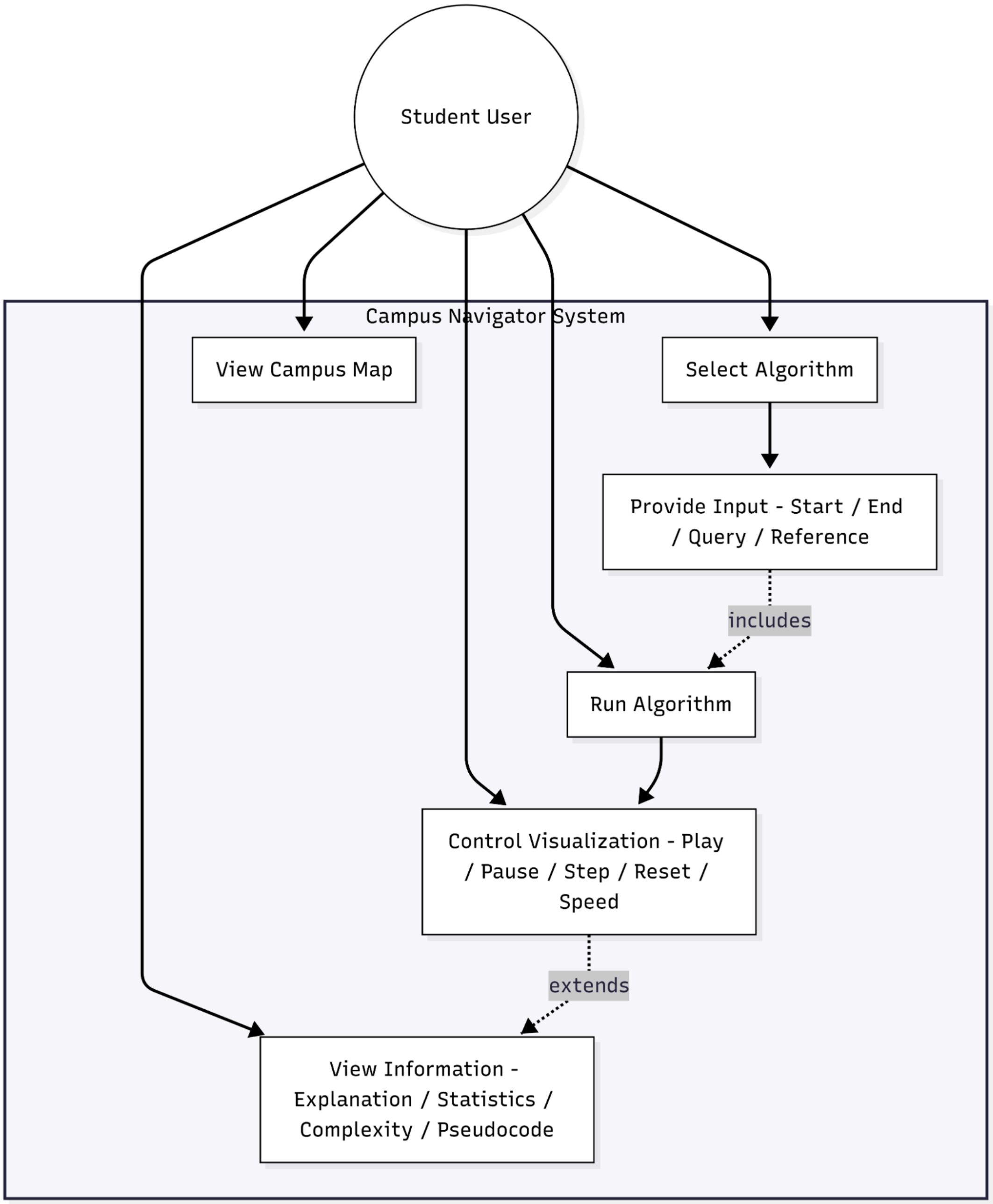

Student User
Campus Navigator System
View Campus Map
Select Algorithm
Provide Input - Start / End / Query / Reference
includes
Run Algorithm
Control Visualization - Play / Pause / Step / Reset / Speed
extends
View Information - Explanation / Statistics / Complexity / Pseudocode

Fig 3.1.1 : Use Case Diagram

The system architecture diagram shows the overall structure of the Campus Navigator System and how its different components are organized. The student user interacts with the main system interface, which handles algorithm selection, input processing, and visualization control. The system contains modules for parameter input, algorithm execution, playback controls, and information display. When the user selects an algorithm and provides inputs, the algorithm execution module processes the data and generates the output. The visualization module then displays the results in a step-by-step animated format. Additional components provide explanations, statistics, complexity analysis, and pseudocode to help users understand how the algorithm works. In simple terms, the architecture diagram explains how different parts of the system work together to take user input, process it, and present clear visual results.

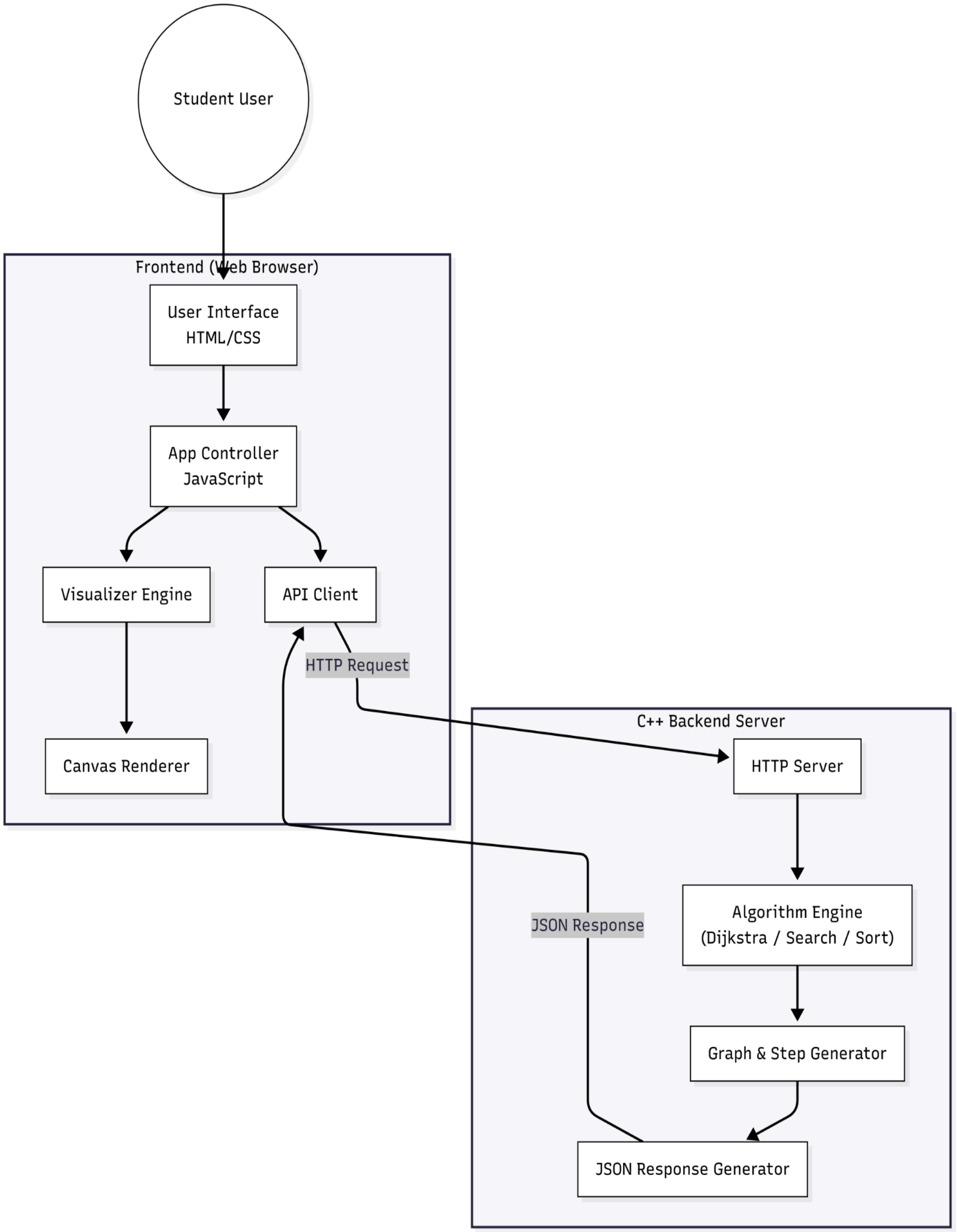


Fig 3.1.2: System Architecture

3.1.4 Development

With the design finalized, the implementation phase began. Each component was developed using C++ as the programming language for the backend logic and data processing. HTML, CSS and JavaScript were utilized for developing the frontend interface. A simple HTTP server was used to connect the frontend and backend components. Development phase included the following steps:

a. **Backend Graph Implementation**: The first step was implementing the graph data structure that represented the campus.
b. **Algorithm Implementation**: Three fundamental algorithms (Dijkstra's algorithm, binary search and quick sort) were implemented using C++.
c. **HTTP Server Development**: The HTTP server used basic socket programming to listen for connections and respond to requests.
d. **Visualization Engine:** The visualization engine was developed to render the algorithm execution on the canvas with proper scaling, colouring and animation.

3.1.5 Testing and Debugging:

The testing phase was critical to ensure the accuracy, performance, and reliability of the system. This included:

- **Unit Testing:** Unit testing was conducted by running custom C++ test cases for each backend algorithm.
- **Integration Testing:** Integration testing involved combining backend logic with the frontend interface. It verified that all endpoints respond with valid JSON and confirmed that the frontend correctly parses JSON responses.
- **User Interface Testing:** All the playback controls were tested. It was made sure that the visualization of all the algorithms were correct.

3.1.6 Documentation

Since this is an academic project, the code was pushed to a github repository publicly. The entire process was documented for review and potential future development.

3.2 System Requirement Specifications

This section specifies the requirements of the developed system. It includes software specifications and hardware specifications.

3.2.1 Software Specifications

This system was built using both frontend and backend technologies. The frontend and backend tools used in this system are listed below:

a. **Programming Languages:**
   - **C++ 17 or later** : C++ is used for algorithm implementation. The system used the standard libraries such as vector, map, queue and string for the backend implementation.
   - **HTML5 :** HTML was used for frontend development. Canvas was used for visualization.
   - **CSS3 :** CSS was used for styling the frontend.
   - **JavaScript ES6+**: JavaScript was used for logic implementation in the frontend.

b. **Compiler:**
   - **MinGW-w64 (Windows) :** It was used as the compiler during the development of this project.

c. **Version Control:**
   - **Git and GitHub account:** It was used as a version control system for hosting the source code publicly.

d. **Development Tools:**
   - **Visual Studio Code:** VS Code was used as code editor during the development of Campus Navigator.

3.2.2 Hardware Specifications

It gives information about the required minimum configuration of the system to run the project smoothly which includes:

- **CPU:** Dual-core processor, 2.0 GHz or faster
- **RAM:** 4 GB
- **Storage:** 2 GB available space
- **Display:** 1280 x 720 resolution
- **Network:** Ethernet or WiFi for GitHub access

Chapter 4 **Discussion on the achievements**

The development of Campus Navigator presented both technical and conceptual challenges that required problem solving and refinement. This chapter discusses the achievements, obstacles encountered and how they were solved.

The successful completion of Campus Navigator fulfilled all primary objectives set before. The system demonstrates three fundamental algorithms (Dijkstra's algorithm, binary search and quick sort) with complete step by step visualization and provides an intuitive user interface with full playback controls.

While developing the system, initially I encountered problems with the canvas scaling. The graph structure appeared broken because nodes would be cut off the canvas. This was fixed by implementing automatic viewport calculation which calculated the scale factor to fit the graph in canvas with padding.

Another challenge encountered was synchronizing the frontend with the backend. Type mismatches, array indexing differences caused inconsistencies which were solved carefully. Infinity was converted to -1 in JSON and displayed as the infinity sign in the frontend. '1' was added to step numbers for user display.

This project encouraged problem solving, enhanced my technical skills and overall captured my growth in software development.

4.1 Features:

The completed Campus Navigator includes following features:

a. **Algorithm Visualization:** Dijkstra's shortest path, binary search and quick sort were implemented with step by step execution showing every decision and state change. Colour coded animation is provided for every algorithm.

b. **Interactive Controls:** Play/pause animation is available with visual feedback. Step forward/backward navigation is allowed through algorithm execution. Playback speed of the animation can be controlled by the user.

c. **Campus graph representation:** Visual campus map with nodes (buildings) and edges (paths) is designed and represented as a graph in canvas.

d. **Educational features:** Clear explanation and pseudocode of each step of the represented algorithms is available. Algorithm complexity information is displayed. Real time statistics (nodes visited, comparisons made and current distance) are displayed in the sidebar.

e. **User Interface:** Clean and professional design with informative panels and responsive canvas with auto scaling is present.

f. **Error Handling:** Invalid inputs are handled correctly. Clear error messages are provided for the clarity of users.

These features collectively provide a comprehensive educational tool that makes algorithm learning interactive, visual and effective.

Chapter 5 Conclusion and Recommendation

Campus Navigator represents a successful integration of computer science education and interactive technology. The project aimed to create an educational tool that makes algorithm learning more engaging and interactive through visualization and this goal has been achieved.

From a technical perspective, the project showcases full-stack development including backend algorithm implementation in C++, HTTP server development, JSON-based API design, frontend interface with canvas based visualization programming.

The system demonstrates that complex algorithmic concepts can be made accessible through thoughtful interface design. By contextualizing algorithms within a familiar campus navigation system, Campus Navigator transforms abstract ideas into practical problem solving. Most importantly, this system validates the principle that education is enhanced when students actively engage with content rather than passively consume it.

5.1 Limitations

As with any software project, Campus Navigator has limitations that should be acknowledged.

a. **Limited Algorithm Coverage:** Only three algorithms are visualized (Dijkstra, binary search, quicksort) due to time constraints. Other fundamental algorithms like BFS, DFS, merge sort, heap sort could be covered.

b. **Static data:** All data is hardcoded. No ability to import graphs from files, save or load scenarios.

c. **No User Authentication:** The system is single-user. No authentication, user profiles or progress tracking.

d. **Quiz Integration:** Initially, a quiz system was planned to enhance interaction with the users but was not implemented.

5.2 Future Enhancement

Several enhancements would significantly improve the Campus Navigator.

a. **Algorithm Expansion:** Other fundamental algorithms such as BFS, DFS, merge sort, heap sort can be further implemented in the system.
b. **User accounts and tracking:** Authentication system can be implemented. The system can be modified to track the learning progress over time.
c. **Interactive quizzes:** In order to further enhance the learning process, prediction and decision questions ("Which node will be visited next?") during the playback can be implemented.

With future development following this roadmap, Campus Navigator could evolve into a top tier algorithm education platform benefiting thousands of students worldwide.

References

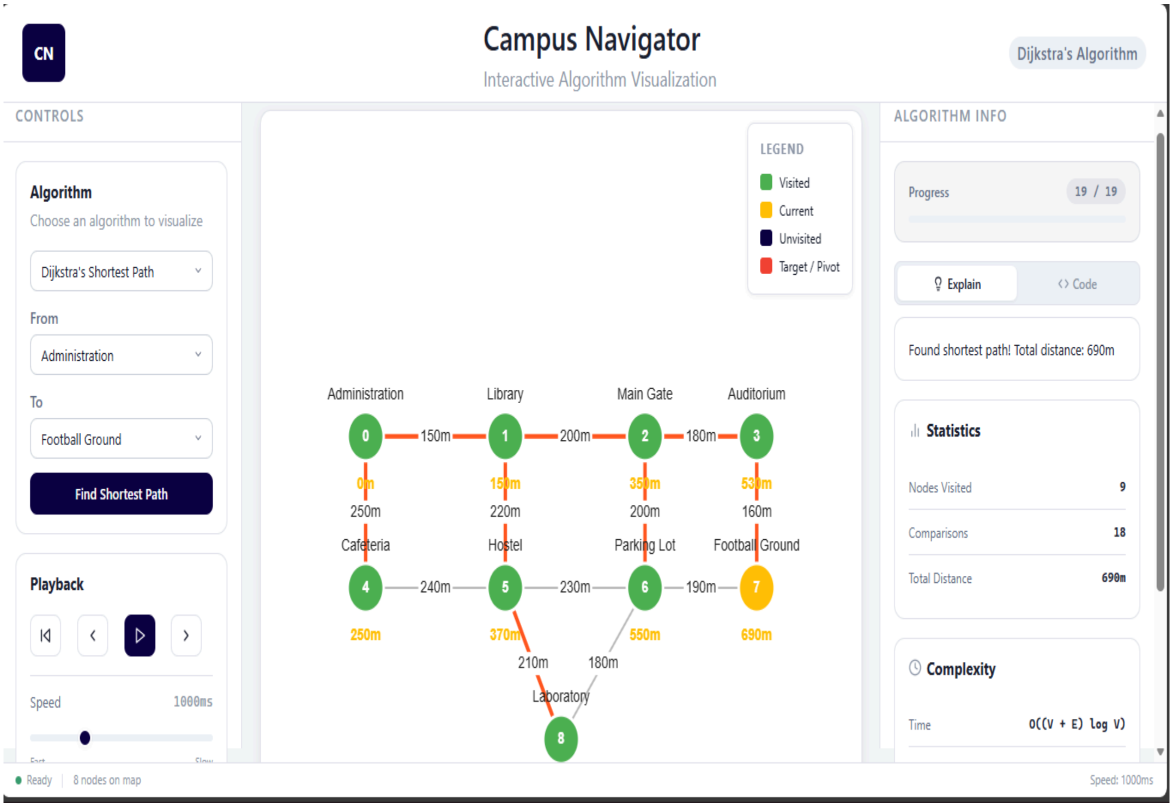
.1. Algorithm Visualizer. (2024). *Interactive algorithm visualizations*. https://algorithm-visualizer.org

2. Halim, S., & Halim, F. (2024). *VisuAlgo: Visualising data structures and algorithms through animation*. National University of Singapore. https://visualgo.net

3. Naps, T. L., Rößling, G., Almstrum, V., Dann, W., Fleischer, R., Hundhausen, C., Korhonen, A., Malmi, L., McNally, M., Rodger, S., & Velázquez-Iturbide, J. Á. (2003). Exploring the role of visualization and engagement in computer science education. *ACM SIGCSE Bulletin, 35*(2), 131–152. https://doi.org/10.1145/782941.782998

4. Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education, 13*(4), Article 15. https://dl.acm.org/doi/10.1145/2490822

Appendix



CN

Campus Navigator

Interactive Algorithm Visualization

Dijkstra's Algorithm

CONTROLS

Algorithm

Choose an algorithm to visualize

Dijkstra's Shortest Path

From: Administration

To: Football Ground

Find Shortest Path

Playback

Speed 1000ms

ALGORITHM INFO

Progress 19 / 19

Explain | Code

Found shortest path! Total distance: 690m

Statistics

Nodes Visited	9
Comparisons	18
Total Distance	690m

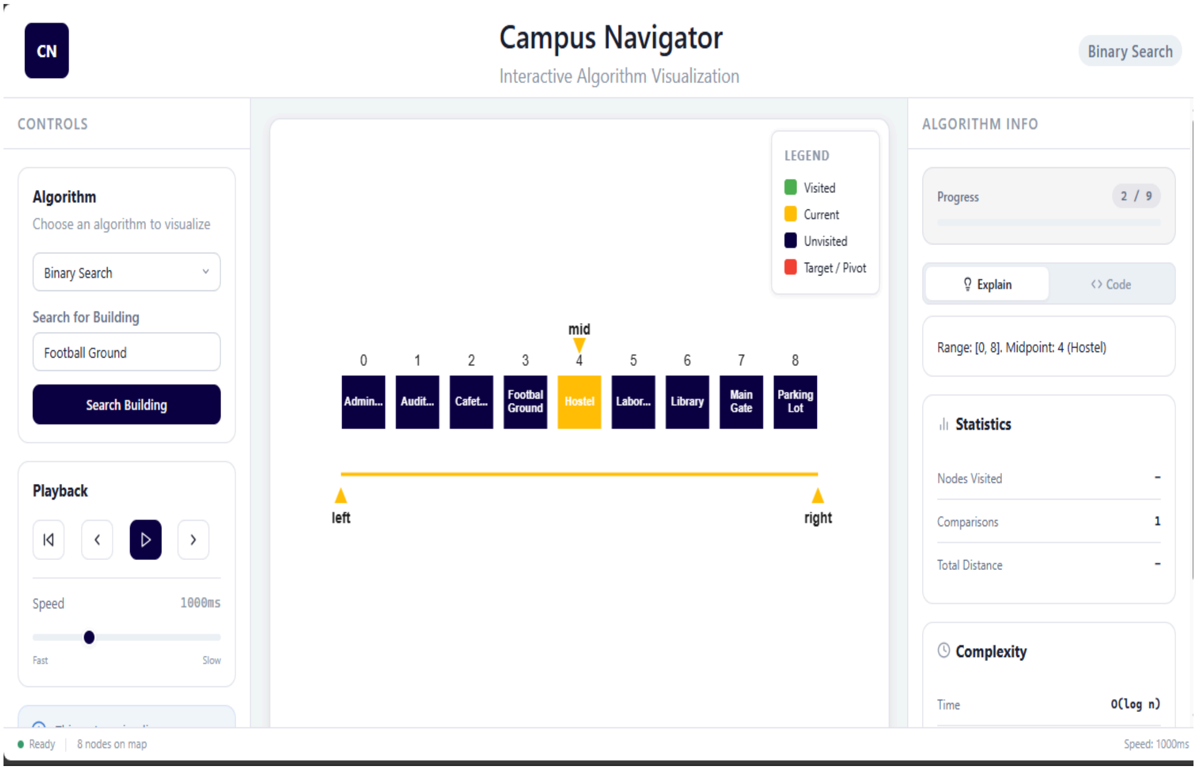
Complexity

Time O((V + E) log V)

● Ready | 8 nodes on map

Speed: 1000ms

Fig 1:Visualization of Dijkstra's Algorithm



Binary Search

Search for Building: Football Ground

Speed: 1000ms

Progress: 2 / 9

Range: [0, 8]. Midpoint: 4 (Hostel)

Nodes Visited: -

Comparisons: 1

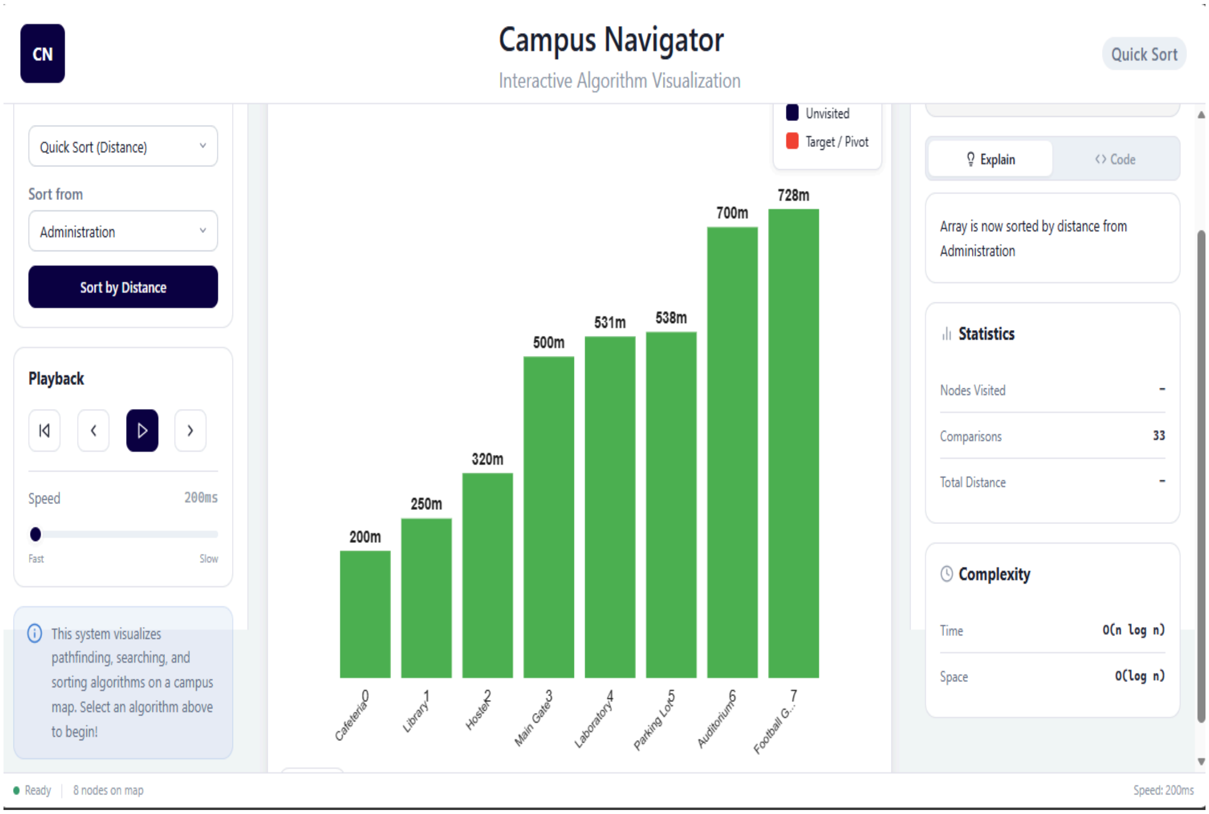
Total Distance: -

Time: O(log n)

● Ready | 8 nodes on map

Speed: 1000ms

Fig 2: Visualization of Binary Search



Campus Navigator

Interactive Algorithm Visualization

Quick Sort

Quick Sort (Distance)

Sort from: Administration

Speed: 200ms

This system visualizes pathfinding, searching, and sorting algorithms on a campus map. Select an algorithm above to begin!

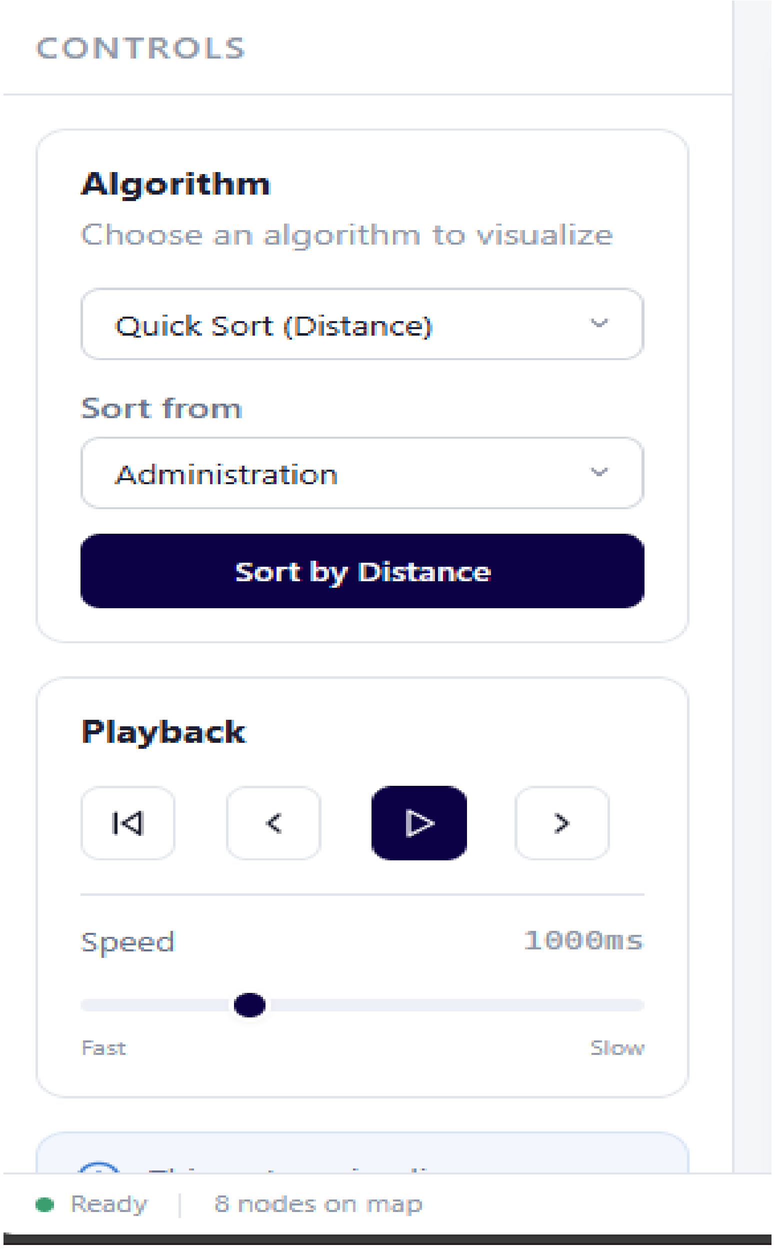
Array is now sorted by distance from Administration

Nodes Visited: -
Comparisons: 33
Total Distance: -

Time: O(n log n)
Space: O(log n)

Ready | 8 nodes on map | Speed: 200ms

Fig 3: Visualization of Quick Sort



CONTROLS

Algorithm

Choose an algorithm to visualize

Quick Sort (Distance)

Sort from

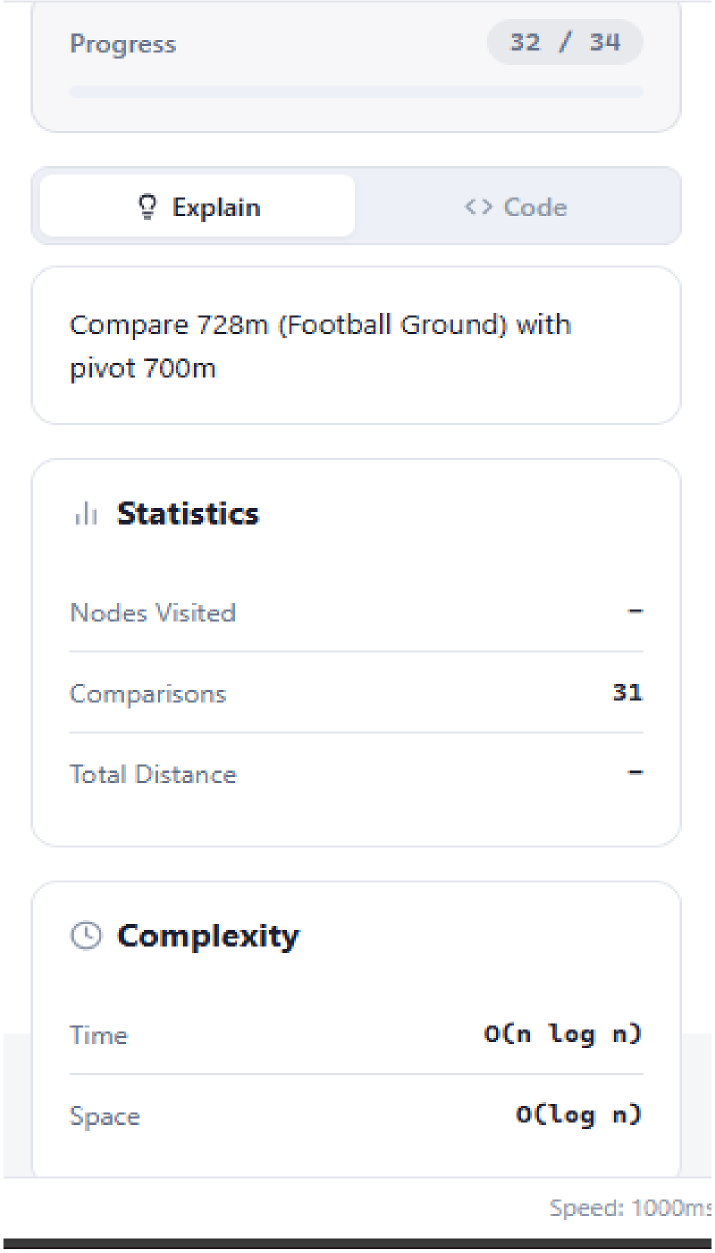
Administration

Playback

Speed 1000ms

Ready | 8 nodes on map

Fig 4: Algorithm Controls



Progress 32 / 34

Compare 728m (Football Ground) with pivot 700m

Statistics

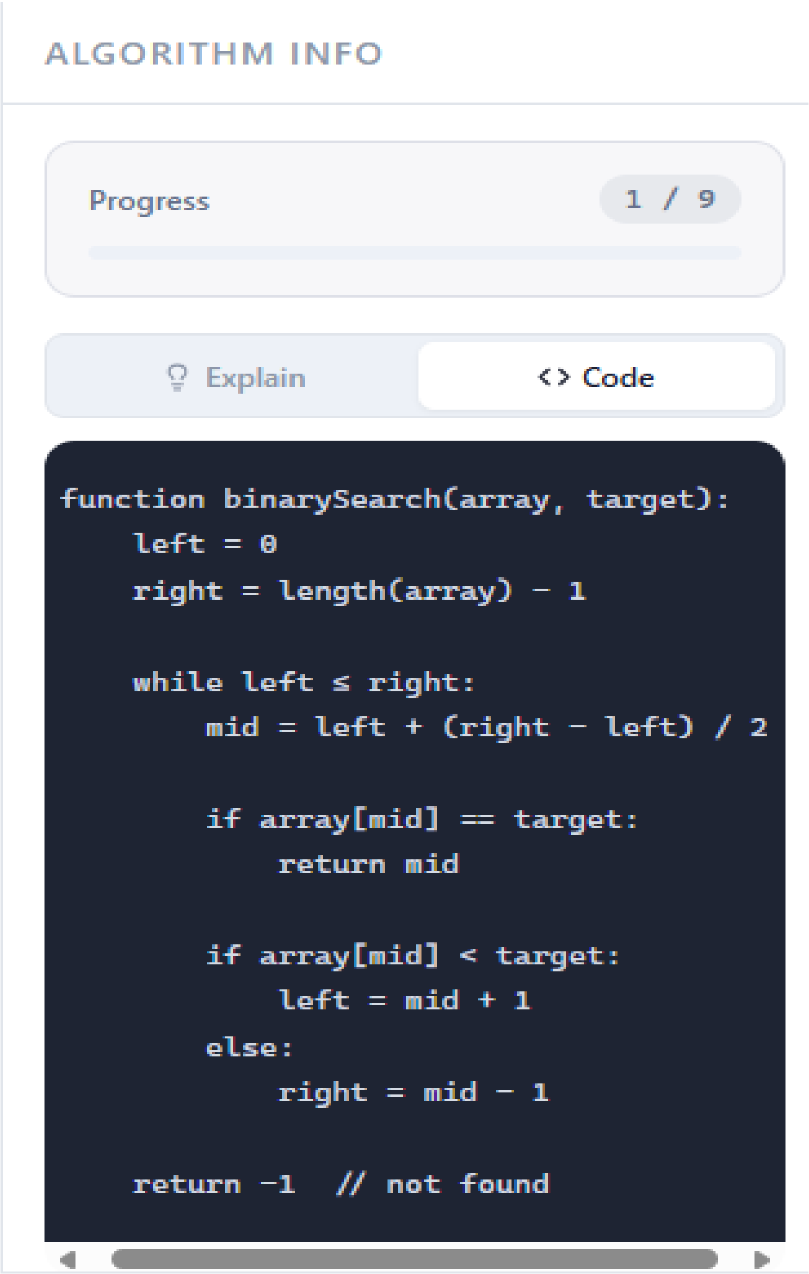
Nodes Visited	–
Comparisons	31
Total Distance	–

Complexity

Time	O(n log n)
Space	O(log n)

Speed: 1000ms

Fig 5: Algorithm Information



```
function binarySearch(array, target):
    left = 0
    right = length(array) - 1

    while left ≤ right:
        mid = left + (right - left) / 2

        if array[mid] == target:
            return mid

        if array[mid] < target:
            left = mid + 1
        else:
            right = mid - 1

    return -1  // not found
```

Fig 6 : Pseudocode Section