Major Project Mid-Term Report on

“**AlgoViz**”

****

in partial fulfillment for the award of the degree of

Bachelor of Computer Engineering

**Submitted By**

Abnish Karmacharya (20070274)

Bibek Manandhar (20070282)

Rikesh Ghimire (20070305)

Sulav Shrestha ( 20070315)

**Submitted to**

**Department of Computer/IT Engineering**

**Everest Engineering College**

**Sanepa-2, Lalitpur**

June, 2024

# ACKNOWLEDGEMENT

We would like to express our sincere gratitude to all those who have contributed to the successful completion of this project on "Data Structure and Algorithm Visualization". Their support and assistance were invaluable throughout the entire process.

First and foremost, We extend our deepest appreciation to major project management commeette for their guidance, expertise, and unwavering support. Their insightful feedback and valuable suggestions played important role in shaping this project.

We would like to acknowledge our mentor Mr. Anuj Ghimire, Mr. Santosh Bist and Mr. Sailesh Pandey for introducing Data structure and algorithm and providing some previous papers regarding the topic.

Further, more we would like to thank our whole faculty members and specially our supervisor Mr. Santosh Bist who has helped us a lot for this project. Their involvement has make our work easier and complete on time.

This project would not have been possible without the collective effort and support of all those mentioned above. Thank you for everyone's effort and hardwork.

# ABSTRACT

Data Structures and Algorithms (DSA) concepts are often difficult for students to understand. The analysis of data structures and algorithms has traditionally been theoretical and mathematical form. This makes it time-consuming, challenging to study, and lacking in comprehension of how a problem is implemented in real life. The complexity of DSA has created a challenge for teachers and learners to teach and learn the concept of DSA like how it works, what is the flow of algorithms and many more. To resolve these problems our project “AlgoViz” an online learning tool that enables the teachers to teach data structures and various algorithms in computer science conveniently and the learners to learns data structures easily while promoting remote and digital learning.

Visualization of data structures, such as sorting, linked lists, and trees allows users to observe the structural changes and data flow in real-time. Visualization shows how the elements are compared, added and removed from a sorting, linked list. Algoviz shows the step-by-step process of an algorithm with bar graphical animation and compares them on basis of their space and time complexity. Algorithm visualization extends this benefit by illustrating the dynamic processes involved in algorithm execution. Sorting algorithms such as bubble sort, merge sort, quick sort, etc. can be shown as animated form how the elements are being compared and swapped. Overall, data structure and algorithm visualization act as the tool that bridge the gap between theoretical learning and practical implementation. Making these abstract ideas more tangible visualization improves problem solving skills and improves the learning technique for students and tutors in computer science.

// Algorithm and data structure are two essential courses for any computer science education. Both students and teachers face difficulty in learning and teaching the concepts of data structure and algorithm. Also, students feel difficulty in learning core concept of data structure and algorithm theoretically. In order to prevent the problem facing by the students and teachers, data structure and algorithm visualization plays a crucial role for their understanding.

Visualization tools are used for enhance learning which enables user to manipulate data and witness the outcomes of different operations. These visualization tools include the features like, playing/pausing, step by step execution, speed selection and back tracking which allows user for effective learning.

//

# Table of Contents

[ACKNOWLEDGEMENT i](#_Toc169980873)

[ABSTRACT ii](#_Toc169980874)

[Table of Contents iii](#_Toc169980875)

[LIST OF FIGURES iv](#_Toc169980876)

[Chapter 1: INTRODUCTION 1](#_Toc169980877)

[Chapter 2: LITERATURE REVIEW 2](#_Toc169980878)

[2.1 Related Works Description 1 2](#_Toc169980879)

[2.2 Related Works Description 2 2](#_Toc169980880)

[2.3 Related Works Description 3 2](#_Toc169980881)

[2.4 Related Works Description 4 3](#_Toc169980882)

[Chapter 3: METHODOLOGY 4](#_Toc169980883)

[3.1 Scope and Objectives 4](#_Toc169980884)

[Objective 4](#_Toc169980885)

[Scope 4](#_Toc169980886)

[3.2 Tools for Development 4](#_Toc169980887)

[REQUIREMENT SPECIFICATION 4](#_Toc169980888)

[3.3 User Testing 5](#_Toc169980889)

[3.4 Sequence Diagram 11](#_Toc169980890)

[3.4 ER Diagram 12](#_Toc169980891)

[3.5 Use Case Diagram 12](#_Toc169980892)

[3.6 Flow Chart 13](#_Toc169980893)

[Chapter 4: Results and Analysis 15](#_Toc169980894)

[Chapter 5: Conclusion and Future Enhancement 16](#_Toc169980895)

[REFERENCES 17](#_Toc169980896)

[APPENDIX 18](#_Toc169980897)

# LIST OF FIGURES

[Fig. Stack Implementation 6](#_Toc169981216)

[Fig. Queue 7](#_Toc169981217)

[Fig. Linked List 8](#_Toc169981218)

[Fig. Bubble Sort Implementation 10](#_Toc169981219)

[Fig. Selection Sort 11](#_Toc169981220)

[Fig. Sequence Diagram 12](#_Toc169981221)

[Fig. ER-Diagram 13](#_Toc169981222)

[Fig. Use Case Diagram 14](#_Toc169981223)

[Fig. Flow Chart 15](#_Toc169981224)

# Chapter 1: INTRODUCTION

**1.1 Background**

In the realm of computer science, data structures and algorithms are fundamental concepts that enable efficient data management and processing. Understanding these concepts is crucial for solving complex computing problems. A good programmer should be well equipped with these base concepts with the ability of critical thinking or algorithmic thinking and problem-solving skills.

Data Structures and algorithms are base, and it plays a crucial role in computer science and engineering. DSA are used in various day to day use technologies such as Web browser, Job scheduling in Operating Systems, Recursion, Database queries and much more. With the subject in student’s curriculum, students are expected to be well versed in several algorithms and data structures.

**1.2 Motivation**

Due to their abstract nature, grasping the intricacies of data structures and algorithms can be challenging. Learning and teaching DSA are both challenging for students as well as teachers. Past research has shown that the new learners find it difficult to understand the implementation of data structures and algorithms in programming. //**citation** Instructors and professors use the means of whiteboards, pen-paper, and textbook diagrams and slides to convey the logic of an algorithm. This requires a great deal of effort and intensive process as to understand the algorithm process, it needs to be broken down into steps and make use of the static tools like texts and pictures.

Many researchers have found and proposed many solutions to overcome these barriers like graphical representation, classical animations, explanation videos, etc. These methods do enable the learner to visualize the logic but limits to the same use of the dataset i.e. do not provide the diversity to work with different dataset or reuse any existing assets for similar task with different approach. Apart from these approaches there evolves an approach that gains popularity amongst the educators in computer science education is Algorithm Simulator or Algorithm Visualizer.

**1.3 Project Objective**

* + Developing an interactive application that graphically illustrates various data structures and algorithms.
  + To enrich the learning process of computer science students by providing a visual aid as a complimentary tool to theory for teachers.

**1.4 Project Applications and Scope**

An Algorithm Visualizer is a powerful education tool that helps student visualize an algorithm by using basic animations and transition effects to understand how an algorithm is working at each step. It also helps in comprehension and retention of the understanding of these algorithms. Many research have concluded that students who are involved in the visualization process of the data structure and algorithms to understand the key working of any algorithm yield a better result in comparison to others who use classical methods because of the dynamic visual representation and interactivity of the algorithm visualizers. [3 citation]

The project we propose AlgoViz is applicable in the teaching environment as a teaching tool aid with animations and transition effects, with dynamic feature to add new data set and a quizionaire to test the student’s understanding of the topic**.**

The forthcoming parts of this paper discusses the previous research and different Algorithm Visualizer developed. Post that, the project introduces our proposed work with all of it’s design frameworks and it’s functionalities with different use cases followed by the conclusion.

**1.5 Hardware and Software Requirements**

The proposed software run effectively on almost any computing system that has the minimum requirements. Being a web application with server side rendering, it works on every device that can run a JavaScript enabled browser. Following is the requirements to run the software in the development mode.

The basic software requirements to run the program are:

1. Windows 10 or higher / MacOS
2. Visual Studio Code
3. HTML, CSS, JS, NextJS
4. Browser such as Chrome, Safari, Firefox

The basic hardware requirements to run the program are:

1. Hard disk 5 GB or higher
2. RAM 2 GB or higher
3. I3 processor or similar higher processing speed

# Chapter 2: LITERATURE REVIEW

Over the past few years with technological progress, several algorithm visualizer tools have been rapidly developed and is still increasing interest in students and teachers. Initially, teaching was based on chalkboards and static diagram to teach their concepts, which has been replaced by different visualization tools. Algorithm Visualizers have a long history in computer science education as they are shown by research as an effective means of learning tool and have received increasing interest from both students and educators. BALSA, JAWAA, HalVis, VisuAlgo.net are the various other projects on smaller scales are previously built Algorithm Visualizers that gained popularity where AlgoVisu.net is the most prominent in today’s day.

Brown University Algorithm Simulator and Animator (BALSA) BALSA [Brown et al., 1985] was one of the first algorithm simulations designed to help students understand computer algorithms. The program has served an example of the animation of many algorithms that were later developed [Wiggins, 1998]. It was developed in the early 1980s at Brown University to achieve a number of goals. Students could use it to look at the performance of algorithms and thus gain a better understanding of their performance.(M.H. Brown, 1984)

One of the popular techniques used for creating and designing the algorithm visualizers in the early days is by annotating the algorithm code by using commands for scripting programming to generate the visualization. One of the popular Algorithm Visualizers that used this technique for visualization purposes is JAWAA. //citation 6 which uses scripting language for generating animations of the data structure and algorithms by simply adding its commands in place of the output of the program. Since it is a scripting type language based on Java, a layman in the computer who has just started to learn may not be able to take the advantage it offers.

Hpermedia Algorithm Visualization System (HalVis) HalVis was developed at Auburn University in the late 1990s. The program is designed with the idea that, in order to make the algorithm visualization more academically effective, in addition to gaining visual attention (made by multiple algorithm visions), you also need to get mental attention and engage the student's mind while viewing algorithm visibility. The program displays algorithms in the multimedia area. HalVis has five modules. T**he data structure and algorithms visualization plays a critical role in development of computer science education. It not only helps in simplifying complex problems but also helps in enhanced learning.** (Sajko, 2012)

AlgoVisu.net is a web-based l conceptualized by Dr. Steven Halim in 2011 with the aims of improving teaching of data structures and algorithms through dynamic interactive visualizations. It has an extensive selection of algorithms and data structure visualizations. It also has a training components where user are tested on concepts. //Ciation visualog chainekura It was made by Professor of National University of Singapore for his students and is contributed by many throughout the world. It is widely used and is featured on platforms such as reddit, Hacker News and has received positive reviews. It uses technologies such as JavaScript, PHP and HTML5.

In this report by Naps in 2002, they have set the stage for a wide variety of future studies that will allow computer science educators to measure the relationship between a learner’s form of engagement with a visualization and the types of understanding that are affected by that engagement. Interactive visualizations specifically encourage active learning and can adapt to different learning paces, which is crucial for accommodating diverse student backgrounds. (Naps, 2002)

**Research has shown that there are various benefits associated with the such of such tools in education. Hundhausen performed a systematic evaluation regarding the effectiveness of algorithm visualizer tools, where students exhibit better problem-solving skills.** (Hundhausen, 2002)**.** This report shows the algorithms visualization effectiveness in order to uncover trends in the research that might help us better understand how and why algorithm visualization technology is effective. As they conducted experimental studies among two group containing 24 members in each group, the group 1 with having algorithm visualization technology knowledge were benefit than conceptual knowledge.

With these research, it can be expected that an Ideal algorithm visualizer produces a substantial difference in the field of study. However, there are conflicting results on the effectiveness of visualization methods over traditional methods. //cita 13. One of the reasons for not achieving desired results is the lack of simplicity and availability to wider audiences. An ideal visualizer should consists of feature such as 1. Ease of usability, 2. Platform Independence. 3. No programming bounds. 4. Interactive animations. 5. Fast and Responsive. While this list is not limited to these features but quite necessary to help our case of teaching environment.

**Evaluation of existing algorithm visualizers**

We evaluated some of the existing and popular algorithm visualizers on the condition of having features listed before. After evaluating these features strictly on the condition of our list of features, there were some major problems that existed which we tend to overcome. BALSA and JAWAA were the earliest visualizers and were written in C and Java respectively and were mainly command based for visualization. The commands were easy to understand but more sophisticated for the end-users ad we need to use exact co-ordinates to plot the data in the display frames to animate. While JAWAA were written in Java so platform independent and used Java Applets for browser-based, it is now outdated and not supported by many browsers. Therefore they lack features listed before. Halvis used multimedia and 3D animations in Hypermedia to visualize which again lacks features of platform. VisuAlgo is a great tool but it is not at all easy to use for novice students to use and may get overwhelmed with the number of algorithms and not knowing what to do. Hence we decide to create our own algorithm visualizer with attempt to include all the listed feature.

# Chapter 3: METHODOLOGY

## 3.1 Theoretical Details

The proposed system “AlgoViz” is an data structures and algorithms visualizer is a tool to aid teachers and students for better understanding the working of an algorithm by visualizing it. It consists of various components for various features and attempts to fulfill the previously listed features. With using a new and different approach by using modern visual design curated for the user of the modern era while maintaining simplicity throughout the application. We have used bars to represent the different data values present in the sorting of an arrays, and used blocks to represent the items in a list, stack jenga like rectangular block to represent the values being added to stack and circular bubbles to represent a node in a tree joined by a line. The application is made easy for the end-user to use. It has been toned down to an extent such that user who has little or no knowledge can also operate and understand the visualization process. It can also be easily controlled by play and stop buttons form the control panel and animation speed can be controlled too with next step and previous steps incorporated too. It also user color coding for different steps that is being animated.

It uses modern technologies for platform independence by using TypeScript ( Typed JavaScript ) and HTML5 in-browser environments, and NextJS is used for React reusable components and to provide server-side rendering of the application. With these capabilities, any computing machine that can run browser can run the AlgoViz web application. AlgoViz also have a quiz and leaderboards feature where user can check their understanding of specific data structure and algorithm. In order to use quiz, the user must login to the application. User doesn’t have to login to visualize the algorithms.

**3.1.1 Block Diagram of the system**

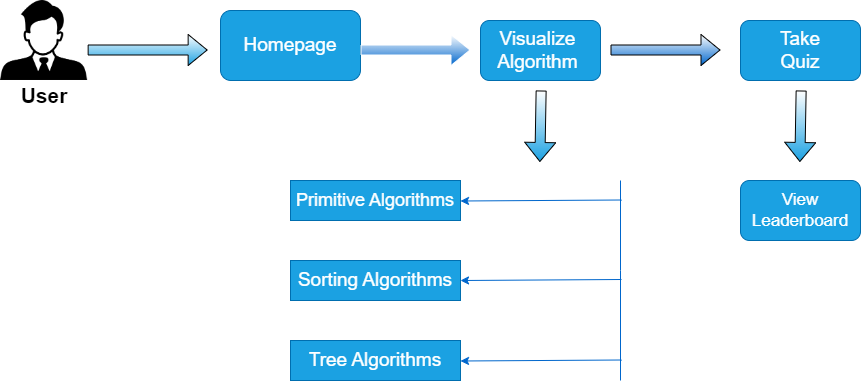


Figure 1 : Block Diagram of the system

The block diagram of the system shows that the user lands on the homepage of the application from where various algorithm visualization can be accessed. Three main algorithms that is to be visualized are Non-primitive algorithms which consists of Stack, Queue, Linked Lists. Sorting Algorthims such as Bubble sort, Selection sort, quick sort.

## 3.1.2 Use Case Diagram

A use case diagram is a type of behavioural diagram defined by and created from a Use Case Analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals and any dependencies between those use cases.

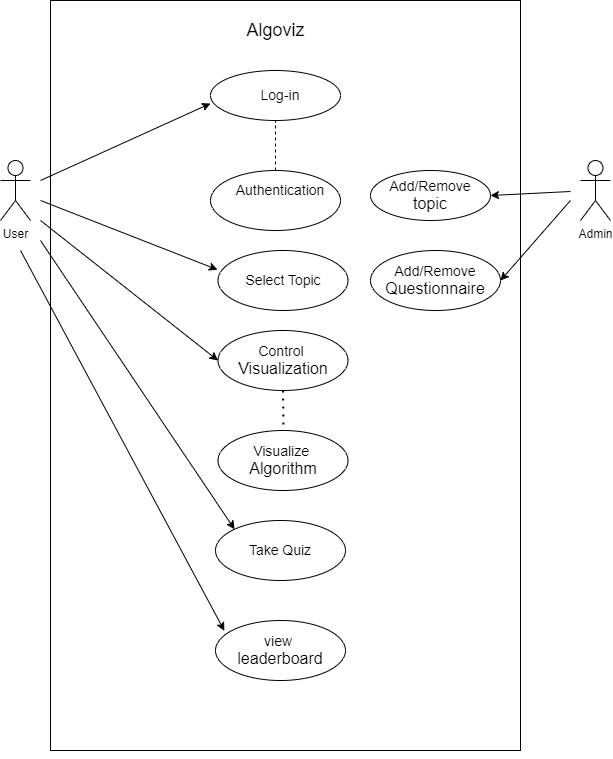


Figure 2 : Use Case Diagram of AlgoViz

Here, in the use case diagram we can see the functions that user can perform such as Login, Select a topic, Control Visualization, Take a quiz and view leaderboards. The admin in the other hand can add or remove topic and questionnaire. As our system is more focused on visualizing the algorithms to the user, the simple approach to guide through the web application is for the ease of use for the user. The user can select through any topic available in the web application and there are buttons, drop down menu, and controls to pause, play, forward, backward, and control the speed of animation, along with change the size of arrays for the sorting visualizer.

## 3.1.3 Sequence Diagram

A sequence diagram or system sequence diagram shows process interactions arranged in time sequence in the field of software engineering. It depicts the processes and objects involved and the sequence of messages exchanged between the processes and objects needed to carry out the functionality.

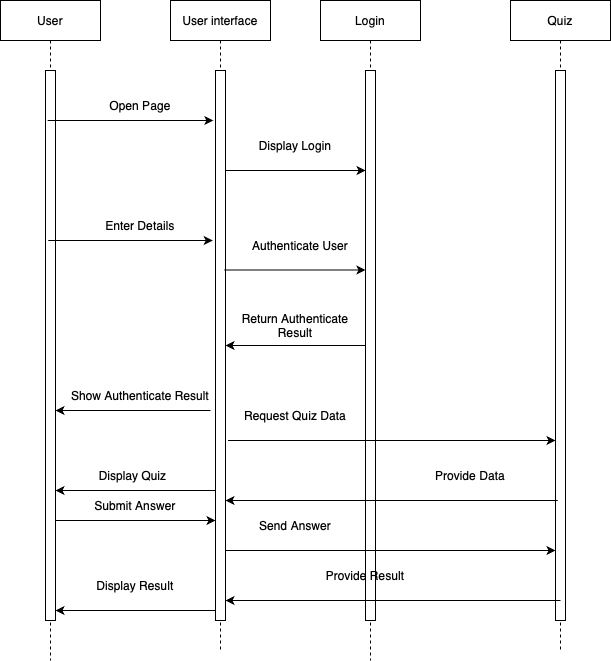


Figure 3 : Sequence Diagram

The sequence diagram visualizes the flow of control in a system. Here the actors are User, User Interface (UI), Login Authentication, and quiz. Here any scenario such as user logins into the system and takes a quiz is shown where user opens the quiz interacting with the UI, for user to take quiz he/she has to login into the system. The system validates the credentials and return authenticate result and user is shown that he/she is authenticated then only user can get quiz data and access leaderboards. This sequence continues as the user answers questions, and the UI communicates responses back to the quiz system.

## 3.1.4 ER Diagram

An entity–relationship model describes interrelated things of interest in a specific domain of knowledge. A basic ER model is composed of entity types and specifies relationships that can exist between entities. In this system there are components with their respective attributes and are connected with each other with some relation.

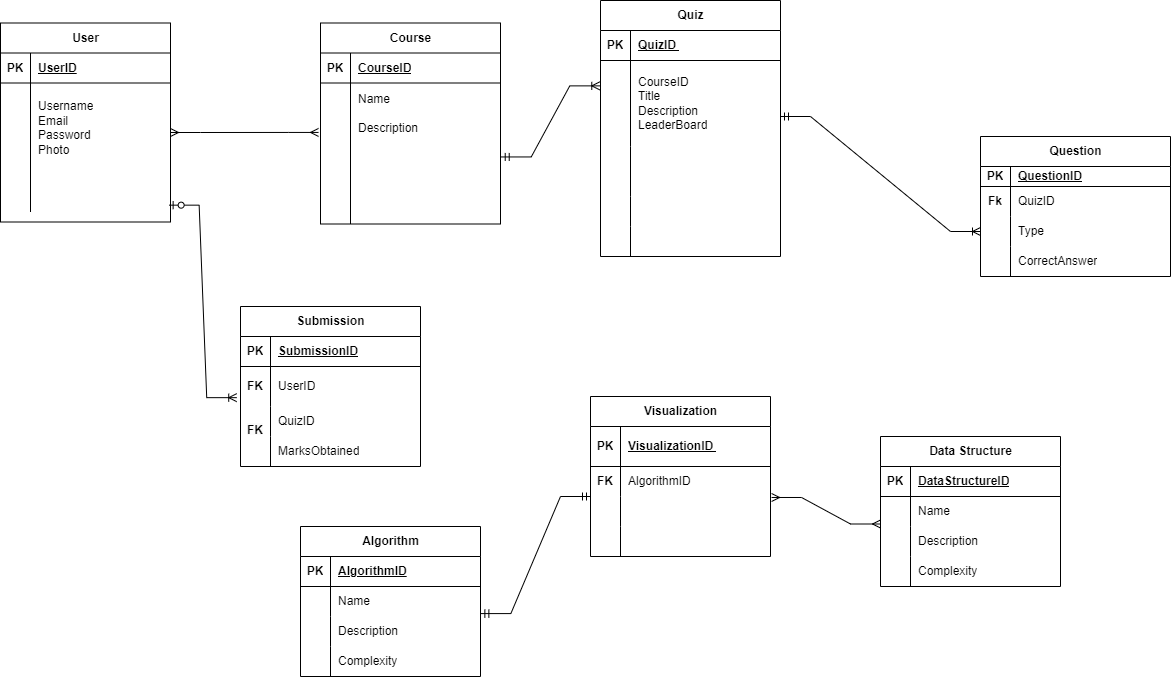


Figure 4 : Entity Relationship Diagram

## 3.1.6 Flow Chart

A flowchart is a diagrammatic representation that is used to write a process, system, or algorithm. It is created by using different symbols, such as rectangles, diamonds, and arrows that designate various steps involved in a process and arrows indicating the flow and sequence of those steps. Flowcharts find uses in many fields for the planning, documentation, and visualization of processes and workflows.

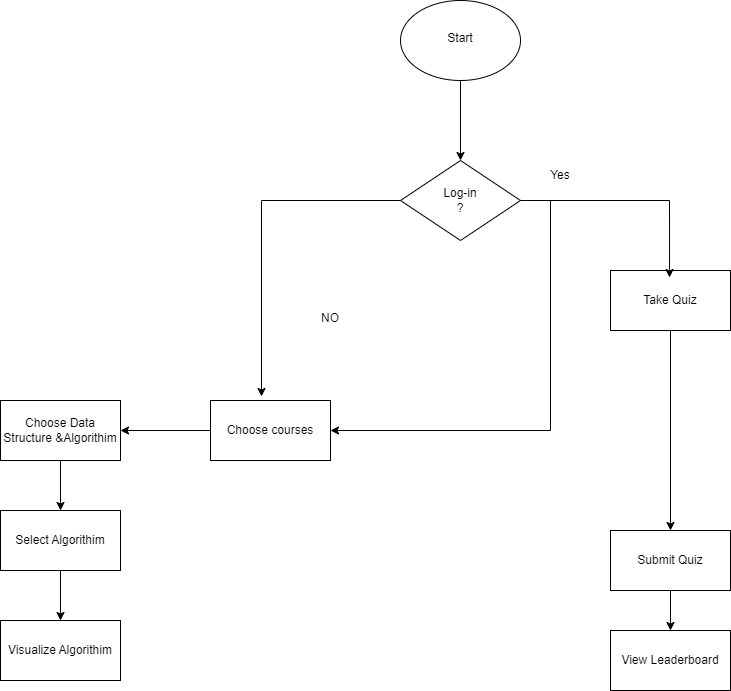


Figure 5 : Flow Chart

The flow chart starts with the decision if the user has logged into the system or not. If and only if the user is logged into the system then user can access the quiz as user data is required to be stored into the database for showing it in leaderboards. If the user is not logged it, user can still use the UI of the web application and visualize the algorithms but cannot access quiz or view leaderboards.

## 3.2 Tools for Development

, shows the process with color coding and shows the time complexities for each algorithm.

## REQUIREMENT SPECIFICATION

## 3.3 User Testing

Alpha Testing: Internal testing is done by the team member to make sure that all the functionalities work as expected.

Beta Testing: External testing is done with the group of people to gather feedback.

Data collection: collect qualitative data and quantitative data through interviews and surveys.

**SYSTEM FUNCTIONALITIES**

The landing page is the page the user first interaction with with web application. It allows user to choose any topic and start their learning process. The topics that are available for the user to choose are : Non-primitive Data Structures such as Stack, Queue, Linked lists. Sorting algorithms such as bubble sort, insertion sort, quick sort. Tree algorithms such as Binary Tree and AVL trees.

**1. Non - Primitive Data Structures**

A. Stack

A stack is a linear data structure that follows the Last In, First Out (LIFO) principle. This means that the last element added to the stack is the first one to be removed. Stacks are used in various applications due to their straightforward and efficient data management capabilities. Stack is considered as fundamentals of the data structure as it provides a simple yet powerful way to manage data. Its unique behaviour is essential for various programming, algorithmic, and system-level operations that would be cumbersome or inefficient with other data structures.

Stack in programming in used in many areas such as memory management, backtracking algorithms such as depth-first search (DFS) in graphs to store the path travelled by pushing into the stack and backtracked by popping, used in software applications like text editiors where actions are pushed onto a stack and undo operations pop from the stack, for balancing symbols and syntax parsing, OS make use of stack to manage task scheduling and interrupt handling and simplifies problem solving in competitive programming.

Here basic operations such as Push, Pop, Peek (Top of Stack) is shown by these methods:

* **Add Item**: Add an element to the top of the stack.
* **Take Out Item**: Remove the top element from the stack (LIFO)
* **Peek**: Retrieve the top element without removing it (highlight it)

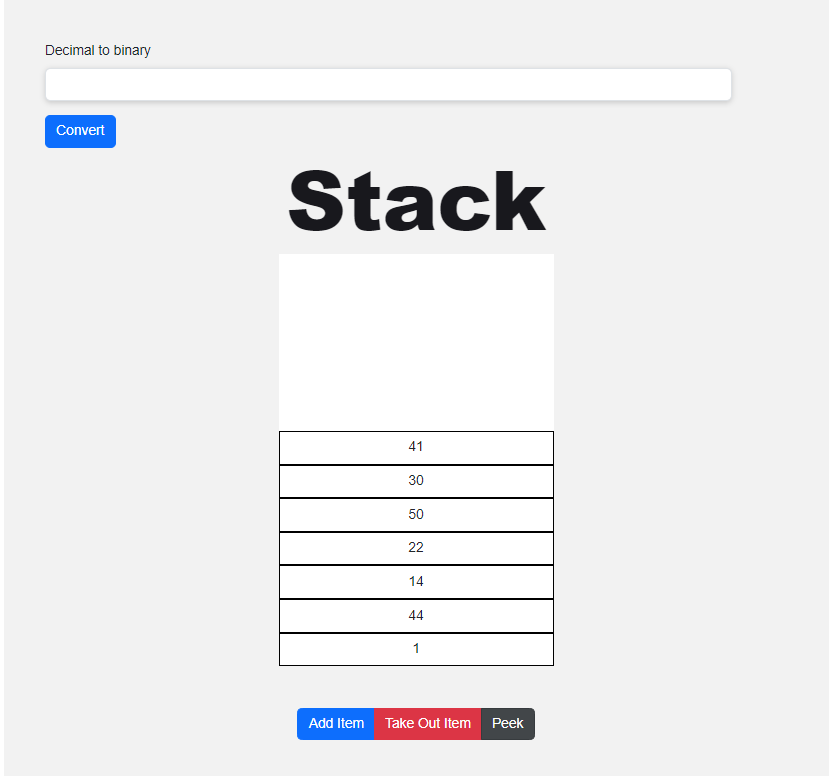


Figure 6 : Stack Visualization

In our web application Stack is implemented by creating a class name Stack. Since Typescript requires type scripting of the datatype the variable going to hold, an items: number [] = []; initializes an empty array to hold the stack elements.

class Stack {

      items: number[] = [];

// Push: Add an element to the top of the stack

      add(item: number) {

        this.items.push(item);

      }

// Check if the stack is empty

      isEmpty() {

        return this.items.length === 0;

      }

// Clear the stack

      clear() {

        this.items.length = 0;

      }

// Pop: Remove and return the top element of the stack

// Return undefined if the stack is empty

      takeOut() {

        if (this.isEmpty()) {

          return undefined;

        }

        return this.items.pop();

      }

// Return the number of elements in the stack

      size() {

        return this.items.length;

      }

//Return the top of the stack

      peek() {

        return this.items[this.items.length - 1];

      }

    }

With use of this class, we can initialize any new instance of class for example  const myStack = new Stack(); which provides all of the methods of the class to the new class instance myStack. It is within the useEffect hook so it renders when the component is changed. Using this we can make use of the Stack datastructure for various use cases for example we have used it to change the decimal into binary and then put the values in the stack which can be then Remove item (popped), Peek (view top of the stack), or add new item (Push). Different event handlers were used to trigger the event of the button click  const addItemBtn = document.querySelector('#addItemBtn'); add added an event listener of ‘click’ to add the random generated item into the stack. Behind the scenes in the CSS box container, box size and its flex direction is defined such that visualization makes sense as a real life example of stacking books on top of the other.

B. Queue

A queue is a linear data structure that follows the First In, First Out (FIFO) principle. This means that the first element added to the queue will be the first one to be removed. Queues are widely used in various applications where order matters, particularly in scenarios that involve waiting and is highly effective for managing tasks, resources and data flow.

Common use cases of Queue data structures are Breadth-First search (BFS) in graphs and trees where queue is used to explore nodes level by level. Nodes are enqueued when discovered and dequeued when they are processed. Task scheduling and process management in OS, queue manages tasks in CPU scheduling where processes are added to the queue and executed in order. In Producer-Consumer problem, queue manages communication between producer threads (add data) and consumer threads ( remove and process data) aiding in multi threading. It is used in messaging systems, for example AWS SQS uses message queues to manage passing between different parts of an application and ensuring messages are processed in correct order.

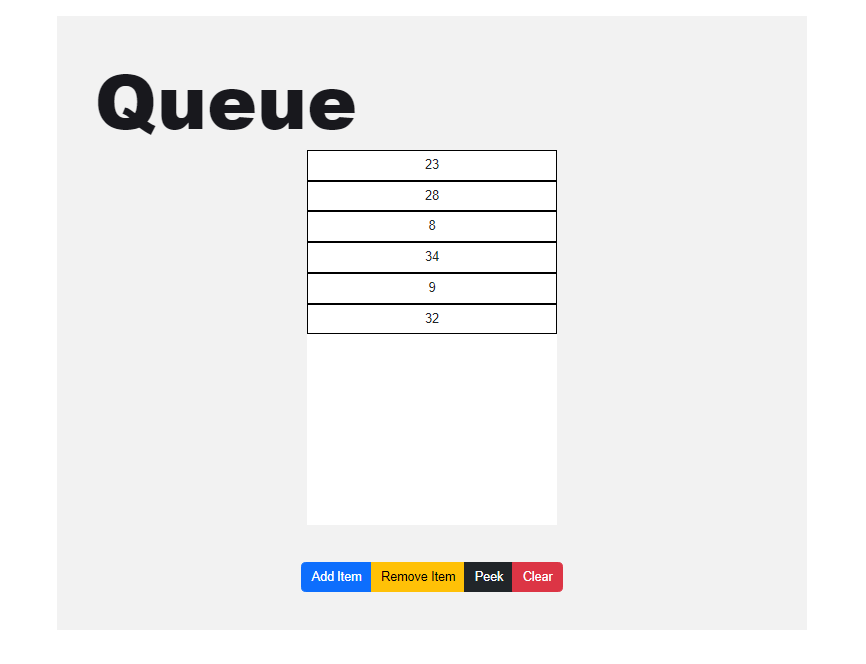


Figure 7 : Queue Visualization

While looking similar to Stack, it works on different principle i.e. FIFO. Basic operations of Queue such as Enqueue, Dequeue, Front/ Peek. Clear is carried out by these methods.

* **Add Item**: Add an element to the end of the queue.(Enqueue)
* **Take Out Item**: Remove the front element of the queue.(Dequeue)
* **Peek**: Retrieve the front element without removing it.
* **Clear**: Remove all the elements from the queue.

Similar to the stack class, we begin with creating a Queue class which will have the methods for the working of the queue. Here items is defined as key value pair which stores the queue elements with numeric keys.

class Queue {

  items: { [key: number]: number } = {}; //Stores queue elements with numeric keys.

  itemToAddKey = 0; //Tracks the next key for adding an item

  itemToRemoveKey = 0; //Tracks the next key for removing an item.

// Enqueue: Add an item to the back of the queue

  add(item: number) {

    this.items[this.itemToAddKey] = item;

    this.itemToAddKey++;

  }

// Check if the queue is empty

  isEmpty() {

    return this.size() === 0;

  }

// Clear the queue and reset the keys

  clear() {

    this.items = {};

    this.itemToAddKey = 0;

    this.itemToRemoveKey = 0;

  }

// Dequeue: Remove and return the front item from the queue

// Return undefined if the queue is empty

  remove() {

    if (this.isEmpty()) {

      return undefined;

    }

    const item = this.items[this.itemToRemoveKey];

    delete this.items[this.itemToRemoveKey];

    this.itemToRemoveKey++;

    return item;

  }

// Return the number of items in the queue

  size() {

    return this.itemToAddKey - this.itemToRemoveKey;

  }

// Peek: Return the front item without removing it

// Return undefined if the queue is empty

peek() {

    return this.items[this.itemToRemoveKey];

  }

}

With the class defined, we can make use of it by making a new instance of the class know as myQueue. const myQueue = new Queue(); inside the useEffect hook and wrapped in a react component named as QueueComponent. Different event handlers and even listeners are added to ensure each buttons calls the said method in the queue class and visualization take place by adding/removing and clearing the queue and rendering the component each time any button is clicked.

takeOutItemBtn?.addEventListener('click', () => {

        myQueue.remove();

        renderQueue();

        peekQueueItem();

}

Here this event listener on click of a button calls the remove method from the class and calls renderQueue to render the queue such that changes can be seen and peekQueueItem peeks the item next.

C. Linked List

A linked list is a linear data structure where each element (called a node) contains a data part and a reference (or link) to the next node in the sequence. Unlike arrays, linked lists do not require contiguous memory allocation, making them dynamic and efficient for insertions and deletions. Linked lists are versatile and fundamental data structure used in various applications where dynamic data handling and flexible memory management are important.

Comman uses cases of linked lists include Dynamic memory allocations : Linked lists are used when the size of the data structure is not known beforehand and needs to be dynamically modified. They are suitable when frequent insertions and deletions occur.

Implementation of stack and queues can be done using linked list efficiently providing constant time complexity for insertion and deletion operations compared to arrays. It is used in memory management to keep track of free and used memory blocks. It is used to implement hash tables (chaining method) and also for polynomial arithmetic with each node representing a term of the polynomial and much more.

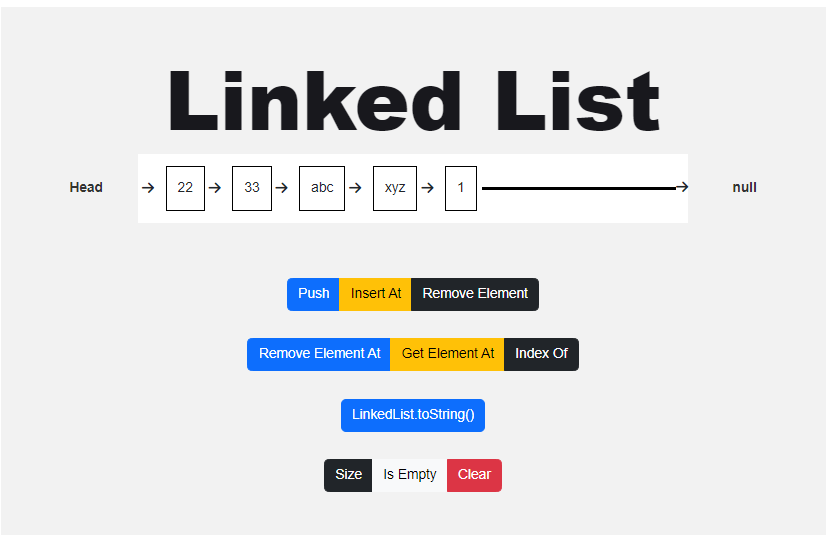


Figure 8 : Linked List Visualization

Basic operations such as Push, Insertion, Deletion, and Search are done by various methods.

The pseudo code of main method that is required for linked list operation is provided here.

export class Node<T> {

  element: T;

  next: Node<T> | null;

  constructor(element: T, next: Node<T> | null = null) {

    this.element = element;

    this.next = next;

  }

}

A class Node with generic class <T> is made that allows the node to hold any type of element. element properties stores the value of the node which is of type T and next is the property that points to the next node in the list. It can be a Node <T> or null if it’s the last node. Then a constructor is initializes a new note with an element type T and next pointer whose default value is null if not provided. Then another class Linked List is made to define the methods of the Linked List where user can add anywhere in the linked list, remove from anywhere. Stack and queue limited to remove the element from either back or front but here the adding new element and removing element is more efficient,

getElementAt(index: number): Node<T> | undefined {

    if (index < 0 || index >= this.size()) {

      return undefined;

    }

    let current = this.head;

    for (let i = 0; i < index && current !== null; i++) {

      current = current.next;

    }

    return current!;

  }

insertAt(element: T, index: number): Node<T> | undefined {

    if (index < 0 || index > this.size()) {

      return undefined;

    }

    const node = new Node(element);

    if (index === 0) {

      const current = this.head;

      node.next = current;

      this.head = node;

    } else {

      const previous = this.getElementAt(index - 1);

      const current = previous!.next;

      node.next = current;

      previous!.next = node;

    }

    this.count++;

    return node;

  }

removeAt(index: number): Node<T> | undefined {

if (index < 0 || index >= this.size()) {

return undefined;

}

let current = this.head;

if (index === 0) {

this.head = current!.next;

} else {

const previous = this.getElementAt(index - 1);

current = previous!.next;

previous!.next = current!.next;

}

this.count--;

return current!;

}

indexOf(element: T): number {

let current = this.head;

for (let i = 0; i < this.size() && current !== null; i++) {

if (this.equalsFn(element, current.element)) {

return i;

}

current = current.next;

}

return -1;

}

With the method indexOf we can easily get the index of any element by traversing through the list. And with the index we can proceed to insertAt a specific index or Removing element at specific index.While this adds new feature and efficient datastructure than stack or queue, it adds the complexity to understand. While these actions are visualized with ease for the user with click of buttons and giving a value to the prompt.

2. SORTING ALGORITMS

1. Bubble Sort

An in-place sorting algorithm that finds max. Element in each cycle and puts it in appropriate position in list by performing swapping adjacent elements. In bubble sort, we continue swapping adjacent elements until they are in correct order.

As we need to iterate the whole array for every element, the complexity of this algorithm is O(n^2).

PSEUDO CODE FOR BUBBLE SORT

Initialize n = Length of Array  
  
BubbleSort(Array, n)  
{  
 for i = 0 to n-2  
 {  
 for j = 0 to n-2  
 {  
 if Array[j] > Array[j+1]  
 {  
 swap(Array[j], Array[j+1])  
 }  
 }  
 }  
}

Time and Space Complexity:

Best Time Complexity: O(n^2)

Average Time Complexity: O(n^2)

Worst Time Complexity: O(n^2)

Best Space Complexity: O(1)

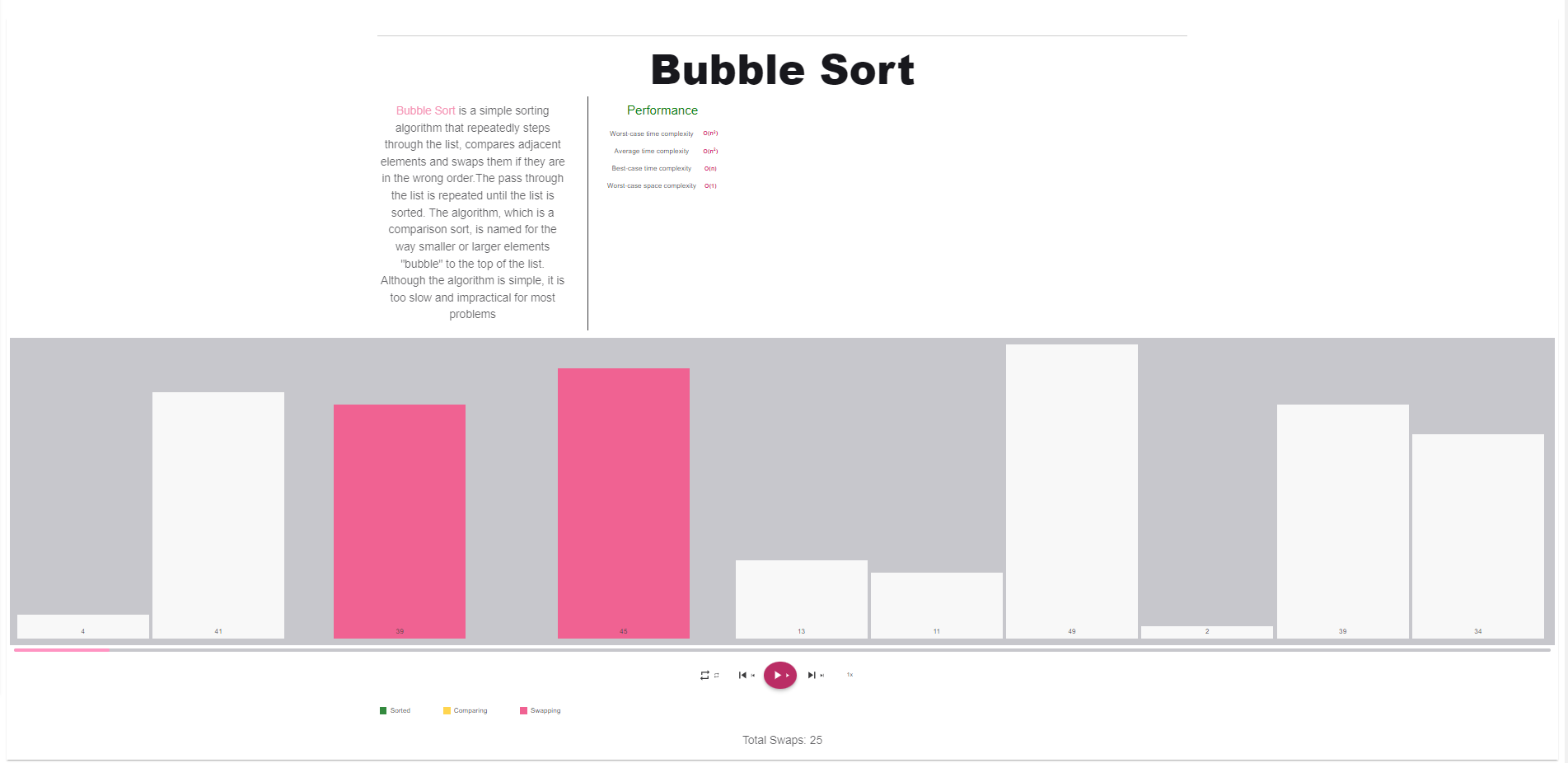


Fig. Bubble Sort Implementation

1. Selection Sort

An in-place sorting algorithm that finds minimum element in each cycle and puts it in appropriate position in list.

PSEUDO CODE FOR SELECTION SORT

Initialize n = Length of Array  
  
SelectionSort (Array, n)  
{  
 for i = 0 to n-2  
 {  
 i\_min = i  
 for j = i+1 to n-1  
 {  
 if Array[j] < Array[i\_min]  
 i\_min = j  
 }  
 Swap(Array[j], Array[i\_min])  
 }  
}

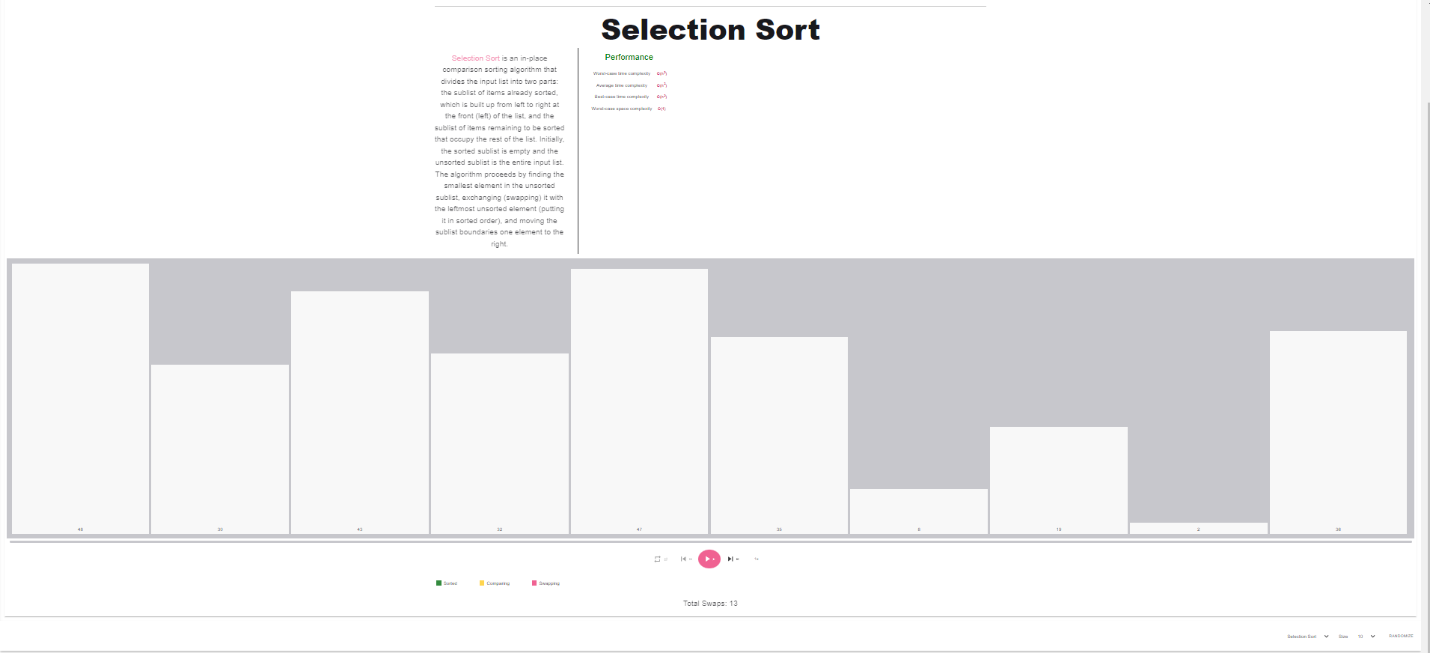


Fig. Selection Sort

Here in all sorting algorithms, its brief description along with the time complexities and the toolbar to change the algorithm speed, change the size of the array, randomize array size and fast forward and backward the steps. The color coding of the steps make the visualization much more understandable form the user view.

# Chapter 4: Results and Analysis

The tool read like a human as it had integrated informative graphics that allowed a user to easily see what was happening in an array, which is similar to a list, a group of connected linked elements, etc. A person could see bubble sort, heap sort, radix sort when trying to sort any information while someone wanted to understand sorts through graphs were useful. Users found step by step execution useful plus being able to change speed rates from slow to fast in addition to providing input their own datasets.

Quizzes are integrated into the tools which allow user to take the test of their understanding and able to see the score in the leaderboard which allows to track student performance. Also, UI is designed user friendly ensuring easy navigation and interactio

# Chapter 5: Conclusion and Future Enhancement

This system is implemented for visualizing some of the sorting, searching algorithms. This is a helpful tool for all kinds of students and tutors to easily understand the execution of algorithms. It focuses on “Algorithm Visualization”, which allows better understanding of its operations.

As Algorithm Visualizer is a combined platform that is a comprehensive result for educator and students to learn online efficiently. For future enhancement we will include more sorting, searching and path-finding algorithm.

# References

Hundhausen, C. D. (2002). *A Meta-Study of Algorithm Visualization Effectiveness. Journal of Visual Languages & Computing,.* Hundhausen.

M.H. Brown, R. S. (1984). *A system for algorithm animation, Proceedings of the 11th annual conference on Computer graphics and interactive techniques, SIGGRAPH.* ACM New York, NY, USA.

Naps, T. R.-I. (2002). *Exploring the Role of Visualization and Engagement in Computer Science Education.*

Sajko. (2012). *Algorithm Visualization.* DCI FEEI TU of Košice. Košic,Slovak: Sajko.

# APPENDIX