“Conceptualization of linear data structures and algorithms through visualization”

Project Report

Submitted in partial fulfillment of the
requirements for the award of the degree of
BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE & ENGINEERING

By

Name	Roll No.
Aryan Chaudhary	500105282
Ishaan Narayan	500104976
Kovid Khanna	500105251
Devansh Goyal	500106363

Under the guidance of

Dr. Virendra Kadyan

Cluster-Head-Data Science

School of Computer Science

University of Petroleum & Energy Studies

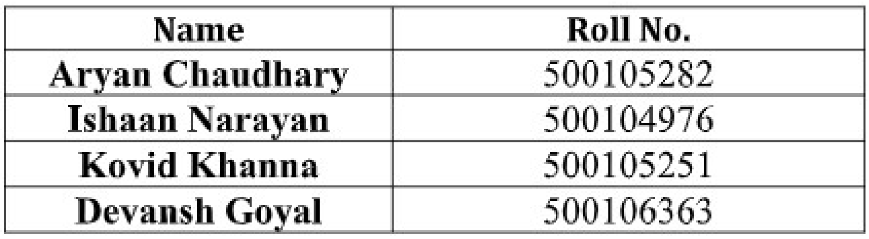
Bidholi, Via Prem Nagar, Dehradun, Uttarakhand

October – 2024

CANDIDATE'S DECLARATION

We hereby certify that the project work entitled "**Conceptualization of linear data structures and algorithms through visualization**" in partial fulfillment of the requirements for the award of the Degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING** with specialization in **AIML** and submitted to the Department of Systemics, School of Computer Science, University of Petroleum & Energy Studies, Dehradun, is an authentic record of our work carried out during a period from **August 2024** to **October 2024** under the supervision of **Dr.Virendra Kadyan, Cluster-Head-Data Science, University of Petroleum and Energy Studies.**

The matter presented in this project has not been submitted by us for the award of any other degree of this or any other University.



Name	Roll No.
Aryan Chaudhary	500105282
Ishaan Narayan	500104976
Kovid Khanna	500105251
Devansh Goyal	500106363

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: **(Dr. Virendra Kadyan)**

Project Guide

ACKNOWLEDGEMENT

We wish to express our deep gratitude to our guide **Dr.Virendra Kadyan**, for all the advice, encouragement, and constant support he has given us throughout our project work. This work would not have been possible without his support and valuable suggestions.

We sincerely thank Prof. Vijaysekra Chellaboina, Head of the Department of SOCS, for his great support in doing our project on **Conceptualization of linear data structures and algorithms through visualization**.

We are also grateful to Dean SoCS UPES for giving us the necessary facilities to carry out our project work successfully. We also thank our Course Coordinator, Dr. Sandeep Chand Kumain and our Activity Coordinator, Dr. Suneet Gupta for providing timely support and information during the completion of this project.

We would like to thank all our friends for their help and constructive criticism during our project work. Finally, we have no words to express our sincere gratitude to our parents who have shown us this world and for every support they have given us.

ABSTRACT

It is basic in computer science education that a student understands linear data structures and algorithms. These concepts are often complex and difficult to understand using the traditional teaching methods. This project is focusing on the **Conceptualization of linear data structures and algorithms through visualization** that would allow the user to have an easy framework where real-time visualizations and animations of data structures like arrays, stacks, queues, and linked lists along with sorting algorithms are done for Bubble Sort, Insertion Sort, and Selection Sort.

The project integrates OpenGL for rendering graphics and C++ for implementing algorithms. Therefore, it offers an attractive platform where users can find out the behavior of data structures and observe the step-by-step execution of algorithms with animations. The visualization tool increases understanding by displaying real-time animations and providing the final result of the algorithms.

It helps to provide an invaluable education resource to students, professionals, and educators as the gap between theoretical concept building and actual implementation can now be reduced. We're gonna include more algorithms in the near future also.

Table of Contents

1. Introduction

1.1 History

For decades, the visualization of algorithms and data structures has been a key component in computer science curricula. The early tools used diagrams and written descriptions that, in general, could not express the dynamic nature of these Algorithms. It is only recently, with graphical rendering technologies such as OpenGL and frameworks like TRAKLA2 and VisuAlgo, that real-time, interactive visualizations are within reach to significantly enhance the understanding and Learning.

Linear data structures are the starting point for understanding computer science concepts, especially those regarding algorithms and data management. However, these concepts cannot be easily understood by first-time learners in traditional and static forms. We shall design an interactive framework where linear data structure algorithms can be visualized. Our framework is designed such that complex operations are broken into step-by-step visualizations (Fig 1).

This will make the users understand the underlying process in a better and more coherent manner. Our tool does not only make learning easy but also deep insights into algorithmic efficiency with integration of key principles from the Design and Analysis of Algorithms. The tool is best suited for the student and instructor and all interested in deepening their knowledge on data structures using hands-on, visual means.

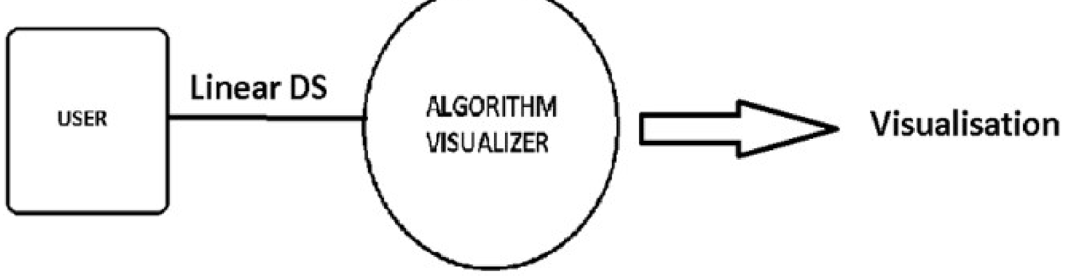


Fig 1

1.2 Requirement Analysis

The development of the Linear Data Structures Visualization tool involved in-depth knowledge of the hardware, software, and functional requirements for smooth running and a strong user experience. Below is an all-inclusive breakdown of the requirements:

1.2.1 Hardware Requirements

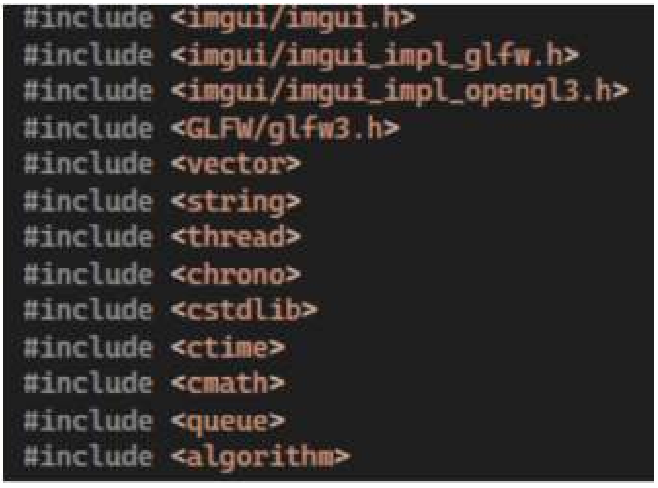
1. **Graphics Processing Unit (GPU):** It extensively uses graphical rendering for animation in real time. This makes an OpenGL-compatible GPU absolutely necessary. A minimum of 2GB of dedicated VRAM is needed for smooth rendering of animations with datasets of moderate size (e.g., 1,000 elements). NVIDIA GTX series or equivalent AMD GPUs are recommended for optimum performance.
2. **System RAM**: At least 4 GB of system RAM is considered for efficient memory allocation across the operations. For datasets of larger size or a multi-algorithm visualization, consider at least 8GB of RAM.
3. **Processor:** A multi-core processor with data input, animation renderings, and user interface updates should be allowed by it. Intel i5 or Ryzen 5 series processor and above are suitable in terms of performance. At least 1 GB of free disk space should be available for the installation of the libraries, development tools, and temporary files.
4. **Display:** At least 1366x768 screen resolution for proper visualization of elements and user interface components

1.2.2 Software Requirements

1. Operating System Windows 10 or newer versions are preferred since they offer the maximum level of support for development tools and graphics libraries. The next versions should be able to support Linux in order to make access wider
2. Visual Studio 2022: It uses C++ and requires Visual Studio for its development and debugging. OpenGL libraries are already integrated within it.
3. **Libraries:**

- **OpenGL:** To draw all the 2D and 3D graphics, which are mandatory to illustrate data structures and operations on algorithms

- **GLFW:** Responsible for windows and context management of the OpenGL; also responsible for managing the input from the users.
- **ImGui (Immediate Mode GUI):** This is the simplest interactive graphical user interface for the selection of algorithm, setting of parameters and interaction with visualized data structures.
- **C++ Standard Libraries:** It provides the scope to the user to implement the basic data structures such as array, stack, queue and linked list. It even contains STL components like vectors, deques, lists for optimum performance.



```
#include <imgui/imgui.h>
#include <imgui/imgui_impl_glfw.h>
#include <imgui/imgui_impl_opengl3.h>
#include <GLFW/glfw3.h>
#include <vector>
#include <string>
#include <thread>
#include <chrono>
#include <cstdlib>
#include <ctime>
#include <cmath>
#include <queue>
#include <algorithm>
```

1.2.3 Functional Requirements

1. **Real-Time Visualization:** The application should produce in real time visualizations for sorting algorithms (Bubble Sort, Insertion Sort, Selection Sort) and for linear data structure operations push, pop, enqueue, dequeue, insert, and delete.
2. **User-Friendly Interface:** The interface should be user-friendly when utilizing the tool. This will include inputting data, selecting algorithms, and also changing visualization parameters such as speed or step-by-step execution.
3. **Input Validation:** The system should dynamically validate all user inputs to avoid errors. For example, Prevent stack overflow and underflow., Disallow invalid indices for linked list operations. , Ensure that sorting operations are only performed on valid data types.

4. **Scalability:** The tool must perform data of different sizes ranging from 10 to 10,000 elements without loss of efficiency.
5. **Interactive Features:** Users will be allowed to stop and start, or move forward a single step so they can observe every action done in detail. Each step must provide tooltip as well as highlighting.
6. **Complexity Analysis:** At run-time, the system should display number of The system should be designed in a modular fashion so that data structures (such as trees, graphs) and algorithms (like Quick Sort, Merge Sort) can be added in the future.

1.3 Main Objective

The Linear Data Structures Visualization project aims at being an interactive learning medium that helps one understand conceptually what is linear data structures in such a dynamic step-by-step visualization way.

1. First, make it possible to input some required data on a set of linear data structures so that the user can conveniently enter and interact with the data he or she will later utilize for learning and visualization.
2. The second goal is the processing of data gathered from the user. First, validating the input is done ensuring that it satisfies the set constraints for each data type. After validation, all data is arranged in structures and readied for future operations such as searching or sorting.
3. The third aim is to analyze the algorithms used in your project in respect to time and space complexity. This includes measuring how the running time and memory usage of each algorithm scale with the size of the input, and logging these measurements for performance insights.
4. The ultimate aim is to use OpenGL, ImGui, GLFW & GLAD to visualize data structures. This includes initializing, designing graphical representations, implementing dynamic animations for operations such as sorting and searching and offering interactive features to enrich the learning experience.

Over All Purpose, This project aims at developing an abstract framework and concrete implementation of algorithms with theoretical data structure knowledge.

1.4 Existing Scope

1.4.1 Linear Data Structures:

All the following linear data structures are visualized in real-time:

1. **Arrays:** Sorts in detail. It highlights which elements are being accessed when sorting or searching.
2. **Stacks:**
- **Push Operation:** Shows how elements are pushed into the top of the stack with animated movement.
- **Pop Operation:** Removes the top element and the stack changes are highlighted.
3. **Queues**:
- **Enqueue Operation:** Adds the elements into the rear end of the queue with the help of animated pointers.
- **Dequeue Operation:** Removes elements from the front, updating front and rear pointers on-the-fly.
4. **Linked Lists:**
- **Node Creation:** Node insertion with pointer animation
- **Node Deletion:** Removes nodes and adjusts pointers
- **Traversal:** Highlights each node while traversing step-by-step.

1.4.2. Sorting Algorithms:

This project includes both basic and advanced sorting algorithms. This implementation uses interactive step-by-step animations for all types of sorting algorithms, as shown below:

1. **Bubble Sort:**
- Real-time visual element comparisons and swaps.
- Dynamically shows the number of swaps and sorted parts.

2. **Insertion Sort:**

- Shows how the sorted section grows with one element being inserted at a time.
- Points out how elements get shifted around when inserting into the list.

3. **Selection Sort:**

- Selection of the smallest element at each steps, and places it in sorted section.

4. **Quick Sort:**

- Show partitioning steps where an array is broken down into subarrays. Animation of recursive sorting using pivot elements.

5. **Merge Sort:**

- Show how array is divided into smaller and smaller subarrays.
- It is representing the heap merge process as sorted subarrays merged to a one single sorted array.

6. **Heap Sort:**

- Animation of heaps that builds a heap at a time through swaps maintained by heap property.
- Marked extraction of max element and re-heapification from rest.

1.4.3 Complexity Analysis:

Dynamic views of given time and space complexities using implemented operations and algorithms.

Time Complexity: Number of Comparisons and swaps through Sorting algorithms are visualized.

Space Complexity: Dynamically tracks the memory usage for each algorithm and presents statistics about it.

1.4.4 User Interactions

- Import and visualize user-specified datasets
- Select algorithms and data structures to illustrate.
- Controls parameters to vary visualization speed and input sizes.
- Step-through to show exactly what happens at each step.

1.5 Meaning of Visualization in Learning and Impact of the Project

Visualization tools play a very important role in improving the understanding of algorithms and data structures. They provide learners with the dynamic behavior of these concepts, which cannot be given by static diagrams or textual descriptions. Real-time, interactive visualizations turn abstract concepts into intuitive, tangible experiences. Consistent research has proven that interactive visual aids improve problem-solving skills and help retain knowledge better in students.

This project is expected to affect a wide range of users, including:

- **Students:** It makes understanding complex algorithms and data structures easier through each step animated. This would make learning more fun and interactive. This translates into more conceptual understanding
- **Teachers:** It is very great for teaching. Once done, the teacher would have easily demonstrated operations and algorithms dynamically in class. It tries to fill the gap of theory-practical understanding and helps them fine-tune algorithms towards performance optimization within real applications.
- **Developers:** Helps to realize the better fine-tuning of an algorithm toward higher performance. It can even be treated as a debug and analyze tool for comprehending which algorithm behaves, out of these, better about performance

This project solves the student's, the educator's, and also the developer's needs altogether by blending theoretical knowledge along with an interactive visualization platform.

2. System Analysis

2.1 Current System

Current tools in usage including VisuAlgo as well as other available that can be noticed of all of the research paper read out have lots of characteristics however consist of some weakness, drawbacks
1st Many have only predefined the user customization.
The sort, only the one consisted with dependency on interactivity consisting solely Sorting algorithms
This mostly or only consist the explanation done diagrammatically in place of text along with both types

2.2 Motivations

The inspiration for this project stems from the demand for an all-encompassing tool beyond the constraints of systems. Most of the visualization tools designed nowadays have problems related to pre-specified inputs, absence of real-time metrics of complexity, and a generally low level of interactivity. The project proposed here will bridge the gap by producing a dynamic and interactive tool that the users can

- **Dynamically adaptable inputs:** The users will define data, change the size and adapt it for a selected use case to offer flexibility and personalization.
- **Interact with the data structures:** This will make users interact with the visualizations directly hence offering features like pausing, stepping through processes, or changing the speed of the visualization.

This project intends to bridge gaps within the existing solutions but provide hands-on learning of algorithms and data structures, making the learning process more engaging. Interactivity not only makes the tool engaging but also helps with critical thinking as users learn by experimenting with the effect of different operations on the data structures.

The complexity level in modern computing systems does make a need for checking algorithm efficiency and data manipulations. This tool delivers time and space complexities into real-time feedback to provide its users with analytical capability and prepare them to critically determine the performance trade-off; hence, it stands vital both in academic contexts as well as professional working situations.

The project is motivated by necessity to open up learning for a wider population. As intuitive as possible, so that it's equally suited to application by students and educators and developers, opening it up so first-time learners can easily grasp those fundamental concepts; modular as well, which scales in relevance for the kinds of potential future enhancements which were envisioned, like complex algorithms and additional data structures.

2.3 Proposed System

- The proposed system integrates all the new ideas of visualization with an interactive interface for the users in order to provide features:
- Running animated visualizations of sorting.
- Step-wise execution with stacks, queues, linked list.
- Modular design of application software is desirable so that it remains scalable and upgradable later as well.

2.4 Modules

The Linear Data Structures Visualization tool has a number of core modules dedicated to specific data structures and algorithms. The modules provide interactive, step-by-step visualizations to make the understanding and analysis clearer.

2.4.1 Sorting Algorithms

Step-by-step animations of the following algorithms are available in the sorting algorithms module.

Bubble Sort:

- It has animated comparisons and swaps for adjacent elements.
- The algorithm traces the sorted segment and gives a highlight of the sorted segment.
- Total swaps and comparisons are given at runtime.

Insertion Sort:

- Animates the gradual growth of the sorted segment.
- Focuses on elements that were moved to their proper placement in the sorted segment.

Selection Sort:

- Shows a selection of the smallest item in each iteration.
- Animated swapping of the smallest with the first unsorted one.

Heap Sort:

- Animate the heap construction, this includes swapping elements to re-establish the heap property
- Highlight the extraction of maximum element and re-heapification of the rest

Quick Sort:

- Shows the partitioning process in which elements are reordered according to the pivot.
- Anima recursive sort of the left and right subarrays

Merge Sort:

- It graphically shows how the division of the array takes place into smaller subarrays at the divide phase.
- Show merging process where sorted subarrays are combined stepwise.

Each animation of a sorting algorithm shows real time metrics for time complexity- comparisons and swaps and for space complexity- memory use.

2.4.2 Stack

The stack module offers animations for the following operations:

Push Operation:

- Animation of adding an element to the top of the stack.
- Updated top pointer and stack size are highlighted.

Pop Operation:

- Smooth transition in removing the top element.
- Dynamic update of the top pointer and display of stack size reduction.

2.4.3 Queue

The queue module showcases FIFO behavior with the following operations:

Enqueue Operation

- Animates the addition of an element to the rear of the queue.
- Updates the rear pointer and dynamically adjusts the queue size.

Dequeue Operation

- Animates the removal of an element from the front of the queue.
- Highlights the updated front pointer and adjusts the queue size dynamically.

2.4.4 Linked List

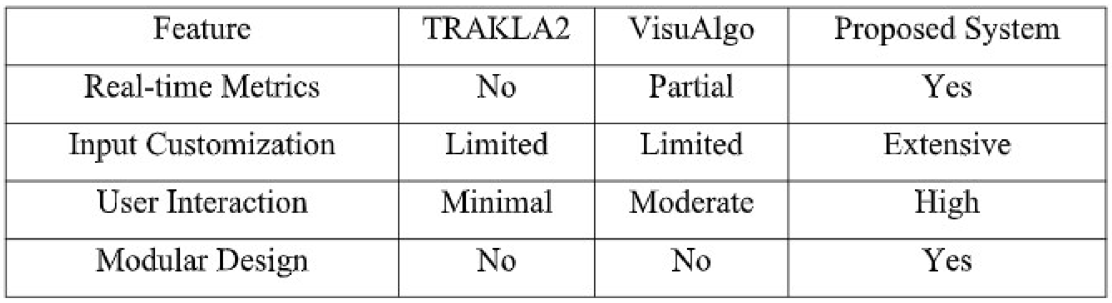
This module contains dynamic animations on the typical linked list operations:

- **Node Insertion:** The program will create and insert a new node to the start of the list or to a particular position and end. Updates pointers according to the changes made

- **Node Deletion:** The process of eliminating a node anywhere in the list. Animates pointer updates while maintaining the integrity of a list.
- **Traversal:** HighLights each node stepwise in traversal. Shows all data stored in each node and pointer updates.

2.5 Comparative Analysis

We compare the proposed system against the existing tools like TRAKLA2 and VisuAlgo:



Feature	TRAKLA2	VisuAlgo	Proposed System
Real-time Metrics	No	Partial	Yes
Input Customization	Limited	Limited	Extensive
User Interaction	Minimal	Moderate	High
Modular Design	No	No	Yes

2.6 Challenges Faced in Development

Problems we faced during the development process:
Performance Optimisation. Animations are required to run smoothly; since rendering is done on datasets.

- **Validation at inputs** - This includes bad inputs and proper input data integrity.
- **Cross-Platform Compatibility:** Highly compatible and modular implementation running across different environments, utilizing ImGui setup.

3. Design

3.1 Architecture

This Linear Data Structures Visualization tool has an architecture modular and scalable along with a certain efficiency character. This linear data structures visualization has altogether six different modules; thus, following are they;

1. Input Module

- Taking care of user's choices for algorithms, choice for data structure, as well as some visualization parameters also.
- Input Validation: Inclusion of all array's value and array size up to the level of their allowance.
- These inputs into valid form only before actually manipulating operations are converted into these forms only.

2. Processing Module

- It is that module which implements an algorithm or performs an operation on the selected data structure.
- It works upon the task of sorting, searching or data structure modifications by means of algorithms in an efficient manner
- It also tracks comparison count, swaps or even memory usage in real-time.

3. Visualization Module

- It does the animation by means of OpenGL to feed it back in real-time.
- It makes smooth and clear visualization of data structure and algorithm steps.
- Supports color-coding with dynamic resizing for better performance.

4. User Interface Module

- It is developed using ImGui.
- This module has an easily understandable, intuitive, clean, and interactive user interface by using ImGui.
- It covers dropdown menus, sliders, buttons, and live feed back for smooth user interaction.
- Users control the parameters of visualization, such as animation speed, input size, and mode for the step-by-step execution mode.

5. Performance Metrics Module

- Tracks and displays important performance metrics such as time complexity (comparisons and swaps) and space complexity (memory usage).
- Actual time updates are available during the execution of the algorithm so that users may be able to understand performance trade-offs.
- Logs metrics for further analysis and benchmarking.

3.2 Improved User Interface Design

The interface is designed to be user-friendly with:

- Dropdown selections for choosing a data structure or an algorithm
- Sliders to set the scale of the input or speed of animation.
- Color-Encoded Symbols show comparisons, swaps and end

3.3 Visualization Improvements

For greater user convenience, the following are part of the application:

- **Dynamic Scaling:** Graphical components change scale based on input size.
- **Interactive Control:** Controls pause and resume functionality as well as step process
- **Tooltips :** Explain what is going on during each step in execution.

3.4 User Experience Design Principles

Focuses on:

- User will not be confused.
- The design should be transparent and clear.
- Adaptive animations on even low power devices.
- A wide range of user, from the new comer to the platform, to the newcomer to data structures and algorithms.

4. Implementation

4.1 Sorting Algorithms

The app provides six different sorting algorithms with animations, making it easier to understand how each algorithm works:

Bubble Sort:

- Animation of comparison and swapping of adjacent elements.
- Color changes for the bars representing the elements when comparing and changes back once sorted.
- Easily visualizes the sorted and unsorted parts.

Insertion Sort:

- Elements not yet sorted are slowly added to the already sorted part of the array.
- Highlights the insertion, indicating which elements are moved into position.

Selection Sort:

- Animation highlights the selection of the smallest element in the iterations
- Swaps the smallest with the first unsorted animation.

Heap Sort:

- Highlighting the heap buildup by animating necessary swaps for heap property preservation.
- Indicates the max removal with subsequent reheapifying the rest of elements from heap
- Highlights elements while removing from heap; when it is removed those already are sorted.

Quick Sort:

- It demonstrates the partitioning process by highlighting the pivot element.
- It demonstrates the rearrangement of elements into subarrays based on the pivot.
- It gives recursive visualization, showing the division and sorting of subarrays.

Merge Sort:

- It demonstrates the divide phase by splitting the array into smaller subarrays.
- It demonstrates the merging process, which combines sorted subarrays into a larger sorted array.
- It demonstrates comparisons during the merge process for clarity.

For all algorithms, it provides live time complexity values with comparisons made and swaps taken and also memory usage. This gives a better understanding and knowledge about the effectiveness of such algorithms.

4.2 Stacks

Push

- It pushes an item into the stack with animations; where the item will be displayed at the top
- It always highlights its updated size, top

Pop

- To insert an animation or transition - delete the element at top.
- Dynamically changes the stack pointer with a visual warning if the stack gets empty.

More Options

- Stack Overflow and underflow are handled with Visual warnings and tooltips for better clarity.

4.3 Queues

Enqueue Operation:

- Adds elements in the back of the queue.
- Dynamically change the rear pointer and make the updated queue size prominent.

Dequeue Operation

- Deletes elements from the front of the queue.
- Dynamically updates the front pointer and visually reduces the size of the queue.

4.4 Linked Lists

Insertion

- Creates a node, and inserts it at front, back or at given position with animation.
- Updating of pointers dynamically to reflect change in structure of the list

Deletion

- It demonstrates node deletion and re-pointers to maintain the integrity of the list.
- It uses new structure to make it clear.

Traversal:

- Traverses each node, and while it is doing so, highlights the node
- Data and pointer changes during traversal are shown

4.5 Error Handling and Validation

The tool has strong error handling and validation mechanisms so that the user experience is as smooth as possible:

- **Stack Operations:** prevents underflow and overflow with visible warnings, ensuring that the push and pop operations occur within valid limits
- **Queue Operations:** identifies invalid operations such as dequeuing an element from an empty queue and provides appropriate alerts
- **Linked List Operations:** the operations such as insertion and deletion will be performed within valid bounds, which point out invalid operations by issuing warnings.

4.6 Optimizations

Several optimizations have been performed to make it run smoothly and scaleable.

Double Buffering:

- It avoids flickering in animations as it renders the frames into a back buffer before they are shown on the screen.
- Ensures smooth transition, even in complex operations.

Optimized Data Structures:

- It makes use of efficient C++ STL containers such as vector, deque, and list for inner operations.
- Reduces the overhead of memory while it maximizes performance for a large dataset.

Multi-threading:

- It keeps rendering and input processing separate into different threads, such that the application remains responsive even during intensive computations.
- It prevents the possibility of hang or delay, especially when visualizing large datasets or performing operations in parallel.

4.7 Examples of Use Cases

- **Sorting Visualization**: Provide a non-sorted array to see the algorithm while the array is sorted step by step.
- **Queue Simulation:** Shows real-time enqueue and dequeue operations for actual applications such as task scheduling.
- **Linked List Operations:** Illustrates node insertion and deletion in a singly linked list.

4.6 Performance Metrics Implementation

Time Complexity: Keeps track of the number of comparisons and swaps and displays it in real-time.

Space Complexity: Illustrates memory usage, which shows how different algorithms allocate memory dynamically.

5. Applications

5.1Academic Applications

The Linear Data Structures Visualization tool is a very effective tool for students and instructors. It provides graphical step-by-step representations of operations that an algorithm performs, allowing students to learn complex concepts in an interactive and engaging manner. It bridges the gap from theoretical understanding to practical implementation, thereby encouraging hands-on learning. For instructors, it is an effective teaching aid to demonstrate processes in real time, making what would otherwise be abstract concepts accessible in the classroom setting.

5.2 Corporate Training

This tool is very useful for corporate workshops and training programs, where the professionals teach how to optimize the algorithms and the data structure. The tool, thus, helps the IT professional understand what kind of implication each operation carries on large datasets by providing them with real time visualizations. It, thus, enables engineers practical insights in building efficient systems and better ways of coding, which basically makes it a valuable product for professional development.

5.3 Applications in Research

It is a tool for researchers to test and benchmark algorithms on custom datasets. It gives in-depth analysis of time and space complexity, which enables researchers to optimize the designs of the algorithms and make them more efficient. It serves as a prototyping platform that allows researchers to experiment with new methods and techniques before actual implementation occurs. This is why it becomes an asset for algorithm analysis and innovation.

5.4 Uses in Industry

It is very useful in the industry for software development and data science. Developers can debug and optimize production systems by understanding the behavior of algorithms in real-world scenarios. Data scientists benefit by observing the real-time processing of large datasets, gaining insights into improving machine learning pipelines and data handling operations. Its ability to highlight inefficiencies ensures system performance optimization, making it a vital tool for industry professionals.

5.5 Competitive Programming Training

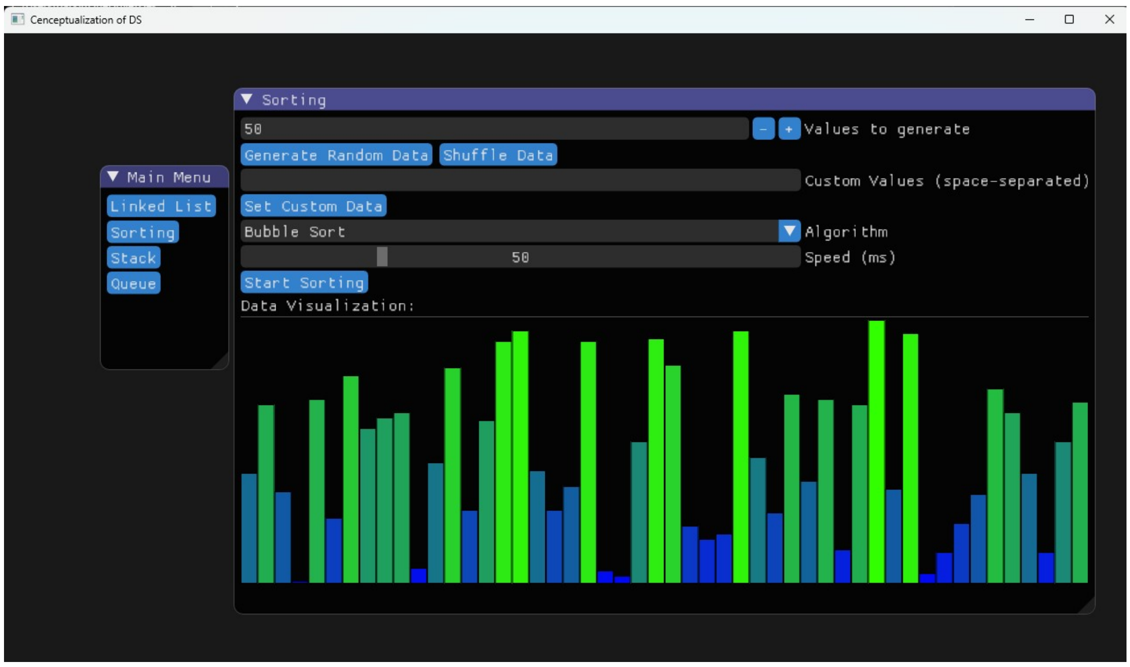
The tool acts as a dynamic and interactive resource for competitive programmers to be able to understand the constrained conditions of algorithm efficiency. Visualization of algorithm steps and associated complexity metrics allows participants to identify bottlenecks and consequently optimize their solutions. More importantly, it makes the participant visualize time and space trade-offs that can enhance better decision-making and performance.

6. Output Screens

6.1 Sorting Operations

Bubble Sort:

- Compares and swaps adjacent elements, animations clearly indicating the changes.
- Highlights each pair of elements being compared to show the decision-making process during sorting.
- Marks the largest unsorted element after each iteration, moving it to its correct position.
- Displays the shrinking unsorted region, making it clear how the algorithm progresses toward completion.
- Real-time metrics show the number of swaps and comparisons performed during execution, helping users understand the algorithm's efficiency.



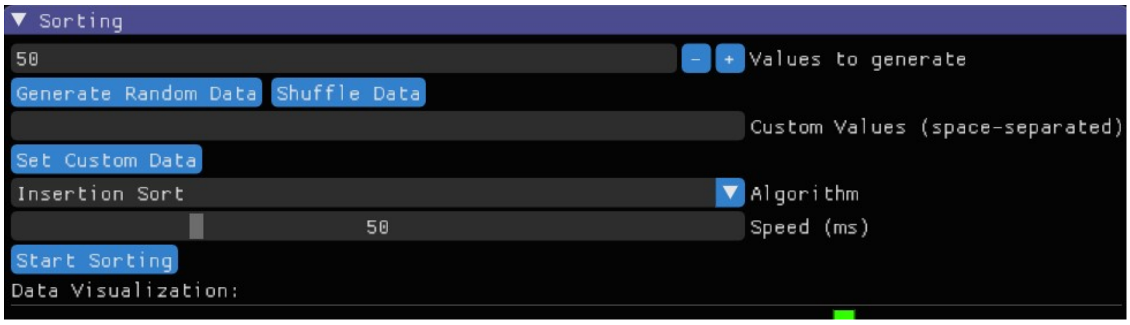
50 Values to generate

Bubble Sort Algorithm

50 Speed (ms)

Insertion Sort:

- It demonstrates the inserting of unsorted elements into the already sorted portion step-by-step.
- It marks out how the current element that it is inserting and shifting space for it.
- This now shows the growth in a sorted portion with it strictly different from the unsorted array.
- It gives animated and step-by-step movements on its elements to show their change in position clearly while giving the number of shift or comparisons made in sorting.



50 Values to generate

Custom Values (space-separated)

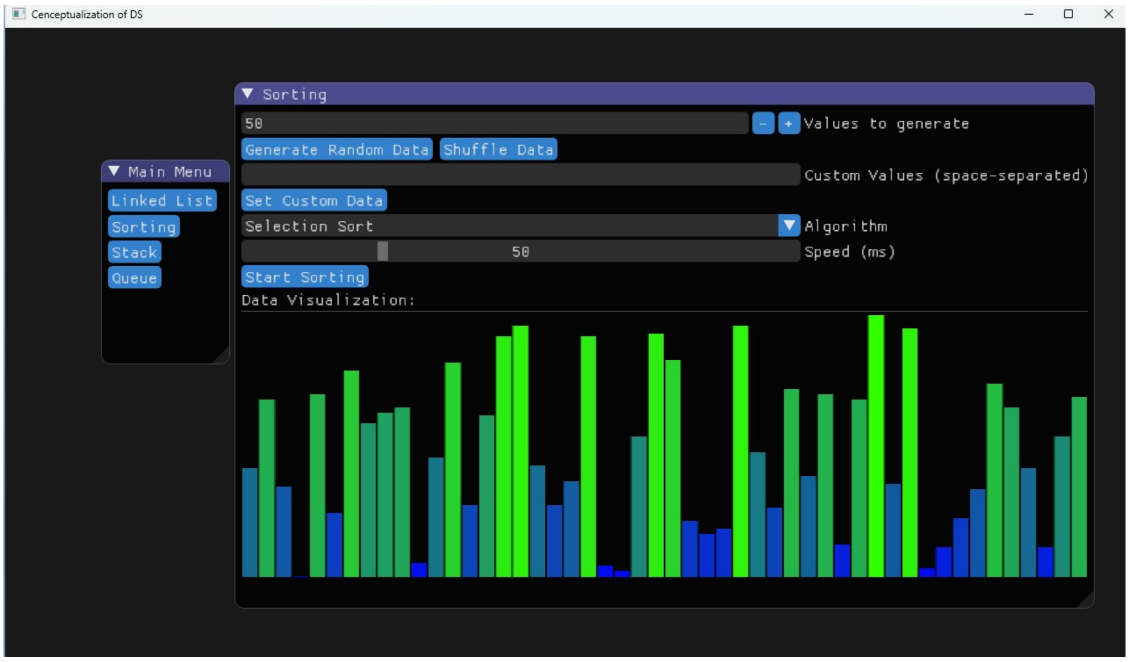
Insertion Sort Algorithm

50 Speed (ms)

Data Visualization:

Selection Sort:

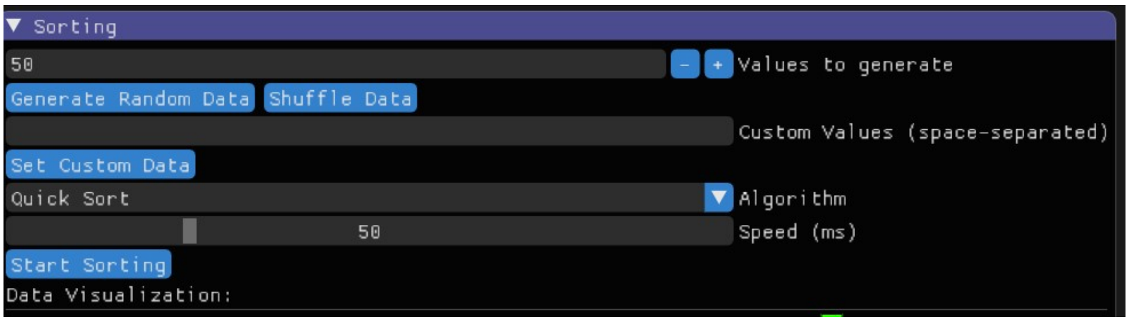
- It shows the selection of the smallest element and it gets shifted to its place at every iteration.
- It points the attention towards scanning for smallest unsorted element.
- It represents the exchange of smallest element with the first element of the unsorted elements through which the advancement of sorted part is very visible.
- It carries the count of comparison in every iteration that may be helpful to the users for better understanding.
- It also shows that how the algorithm narrows down its attention by focusing on the decreasing unsorted portion.



50 Values to generate
Custom Values (space-separated)
Selection Sort Algorithm
50 Speed (ms)
Data Visualization:

Quick Sort:

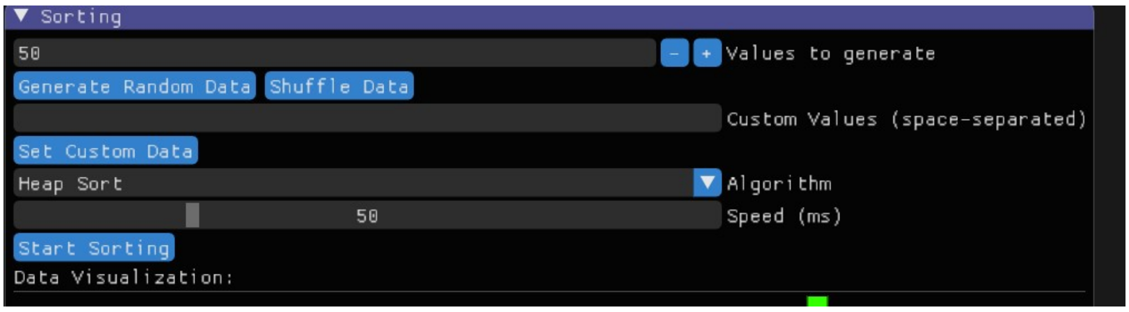
- It emphasizes the step of choosing a pivot and splitting the array into two subarrays.
- It animates the process of shuffling elements in such a way that all elements smaller are on the left side of the pivot, and the larger ones on the right side.
- Recursive calls are animated for sorting the left and the right subarrays, which updates dynamically.
- It vividly illustrates how the array gets completely sorted by merging the outputs of the sorted subarrays.
- It also gives real-time feedback on the recursion depth and partitioning steps to get an understanding of the divide-and-conquer strategy.



50 Values to generate
Custom Values (space-separated)
Quick Sort Algorithm
50 Speed (ms)
Data Visualization:

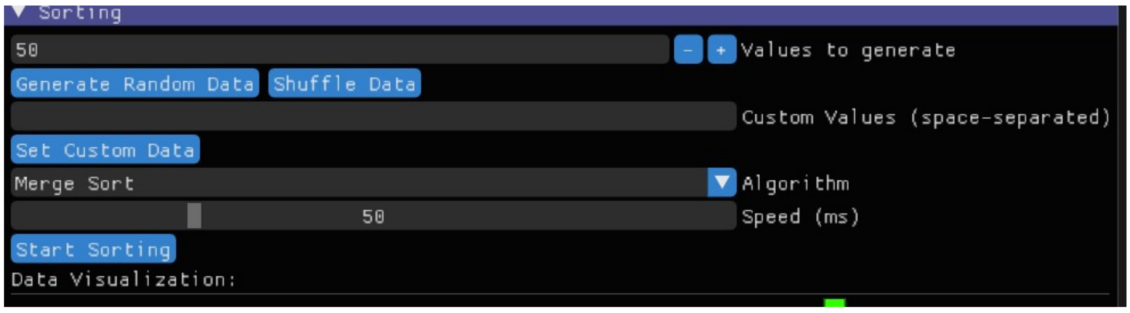
Heap Sort

- This shows how the heap gets constructed and how the maximum element gets extracted from it.
- This also clearly brings out the comparison and swapping that is involved during the heap construction.
- Animate the extraction of the largest element from the root and its insertion in the sorted portion.
- Re-heapify after every extraction: This is made evident to the user in terms of how the heap property gets restored.
- Counts the number of swaps and comparisons that are performed; this would enable users to assess the performance of the algorithm



Merge Sort

- It describes the recursive divide step as splitting the array into smaller subarrays.
- It animates the process of merging the sorted subarrays, highlighting the comparison step of elements of two subarrays.
- It shows pretty clearly how sorted subarrays merge into large, sorted segments.
- It counts and displays the comparisons carried out during the merging process, thus allowing insight into the algorithm's complexity.
- This gives a clear idea about how the algorithm works on smaller portions and subsequently merges them to sort the entire array.



6.2 Linked List Operations:

Linked List Visualizer This is an intuitive graphical representation of operations for linked lists and hence allows the user to better comprehend dynamic interactions of pointers and nodes. Some main features are as follows:

Node Representation:

- The nodes are represented in the graphical format as circular structures holding within them the value each node.
- The arrows depicted within the graph indicate a link or a traversal of order in the nodes.

Insertion Operations:

- Insert at Head: A new node is inserted at the head of the list. The animation depicts the updation of the head pointer to the new node, and the previous head now points to this new node.
- Insert at Tail: A new node is added to the tail end of the list. The traversal goes all the way to the last node, and its pointer is updated with the new node.
- Insert at Position: Inserts a new node at the position. A traversal is animated all the way to the position. Pointers are updated

Deletion Operations:

- Delete at Position: Deletion of the given position node is shown and in the animation, how this traversal reaches the node that gets unlinked and thereafter the pointers get connected to each other

Traversal:

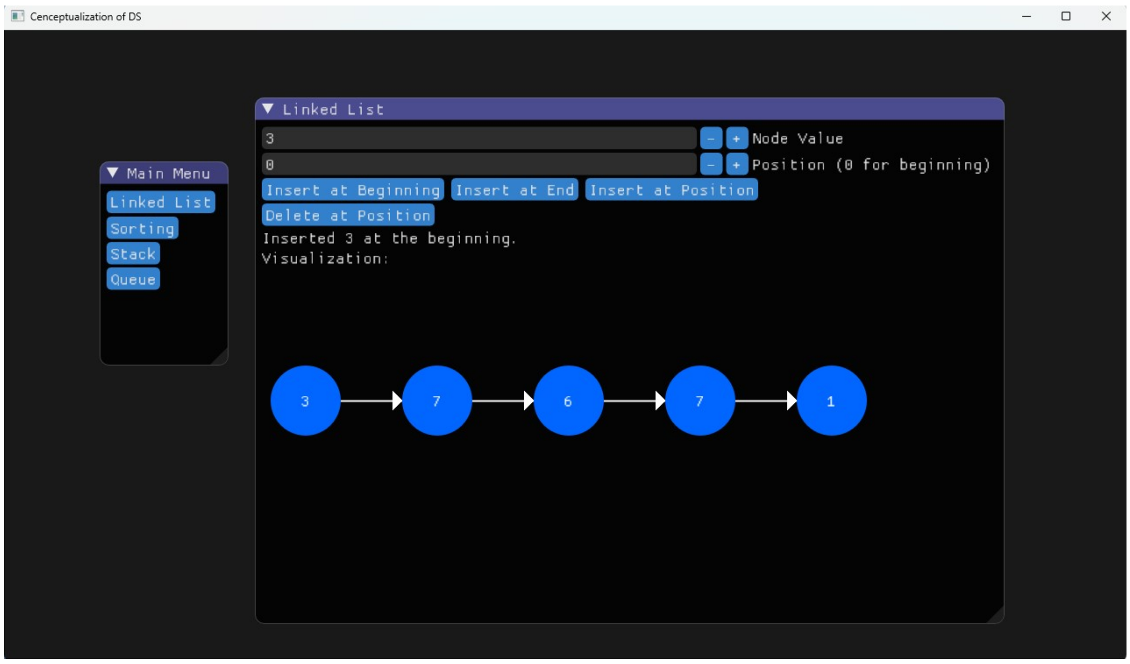
- The tool marks nodes that are traversed through; therefore, explicitness of the traversal path together with pointer updates.
- pointers are moving as indicating where the traversal is currently getting progressed.

Pointer Updates

- The pointer changes are animated by showing connectivity between nodes due to inserts and deletes.
- Clear animations of pointer change make understanding this process more intuitive.

Error Handling

- Visual cue is given whenever operations that tried to insert or delete at a wrong position have been encountered.



```
3                                    - + Node Value
0                                    - + Position (0 for beginning)
Inserted 3 at the beginning.
Visualization:
```

6.3 Stack Operations

Push Operation:

- Whenever an element is pushed onto the stack, the visualizer animates its entry at the top of the stack.
- The new stack size will appear dynamically, as indicating the real-time stack's state.
- A sliding in transition effect will make it feel like it's the new thing in the stack - just popped.

Pop Operation:

- Animation of popping-off an element at the top
- Automatic decrement of stack-size with update of pointer-top reflecting the new stack-size.
- Graphical Underflow Alarm in case pop-off is done when stack is already empty.

Current Stack View

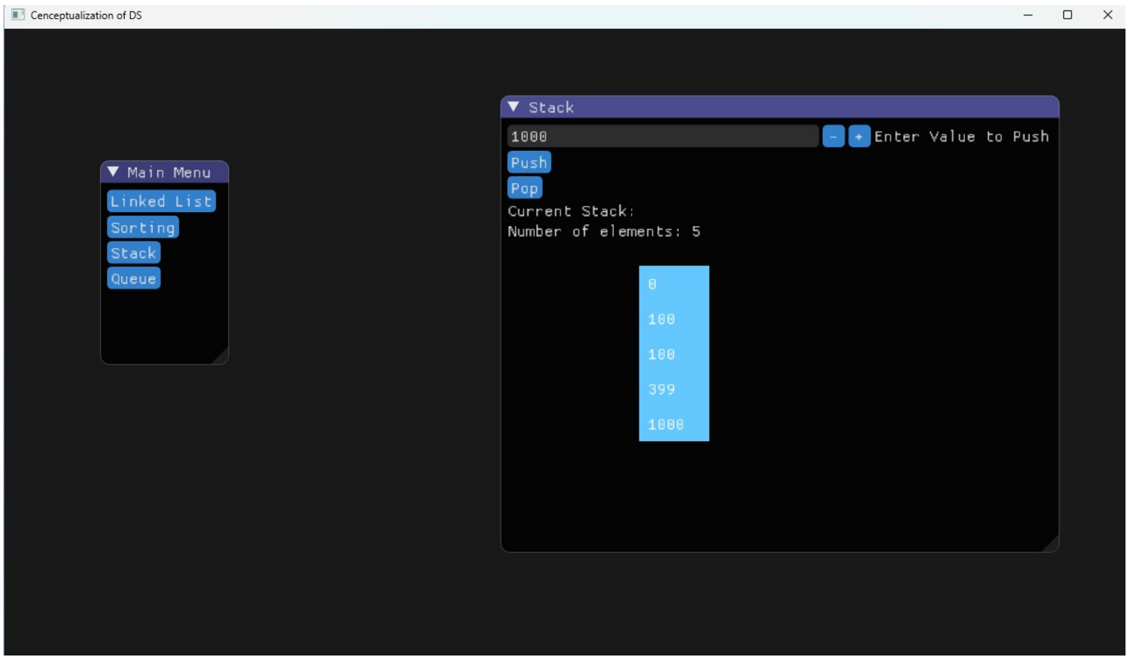
- Graphical stack refreshes every time elements are pushed or popped.
- Each element will be shown as a square block, along with their value.
- The stack resizes dynamically based on the number of elements it has at the time.

Error Handling

- Stack wills not overflow or underflow. It provides a visual alert and tooltip in those cases.
- It ensures valid operations and guides the user with prompts when invalid operations are attempted.

Interactive User Interface:

This has a user-friendly interface in which users enter values to push onto the stack and easily pop. It will provide immediate visual response to user input, making it intuitive and engaging.



Current Stack:
Number of elements: 5

6.4 Queue Operations Visualization

Queue Visualizer is a thrilling and engaging graphical representation of queue operations that shows the behavior of a queue as if it were a First-In-First-Out (FIFO). This way, users get to see how these elements are added to the queue and left the queue and how these front and rear pointers will work in a dynamic way.

Features of Queue Visualization:

Enqueue Operation:.

- The new location for the enqueue operation updates the back pointer instantaneously.
- It is smooth on-screen movement as this new value enters the queue, and thus this new state of the queue is easily seen.

Dequeue Operation

- This depicts when an element gets removed from the front of the queue
- It also depicts a "departure" element along with the front pointer dynamically moving to its successor in the sequence.
- Visual queues size is reduced, along with immediate feedback on queue for new state.

Dynamic front and rear pointers:

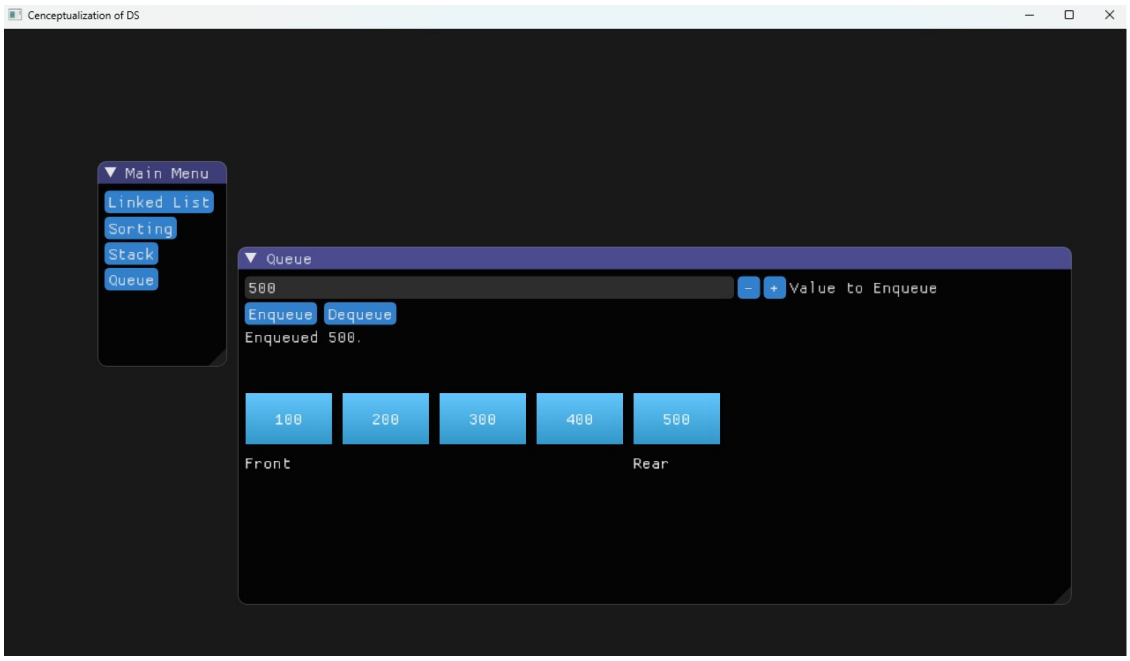
- A clear label is given both to the pointers so users understand where the entry goes in (rear) as well as where it has to exit from (front).
- Each time enqueue as well as dequeue happens it dynamically updates the positions.

Current Queue Display:

- This shows rectangular blocks in the order of the queue elements displayed, along with readable values.
- Whenever elements are enqueued or dequeued, the display reflects the current state of the queue in real-time.

Error Handling:

- Prevents the queue underflow by generating alerts to the user if they attempt to dequeue an element from an empty queue.
- Deals with all edge cases in a non-technical manner, which includes invalid operations and the maximum capacity of the queue.



```
Enqueued 500.
```

7. Limitations and Future Improvements

7.1 Current Limitations

The Linear Data Structures Visualization tool is new and efficient but has limitations of usability and scalability by the following limitations.

It is broadly categorized in that it only supports a few data structures and algorithms like arrays, stacks, queues, and linked lists at present. It supports simple as well as some complex sorting algorithms like Bubble Sort, Insertion Sort, Selection Sort, Quick Sort, Merge Sort, and Heap Sort but does not support any kind of non-linear data structure like trees, graphs, and hash tables. Other than this, it omits advanced algorithms like more advanced graph algorithms or dynamic programming methods which restrict its usage in a wide-range of computational problems.

Performance on Large Inputs

Its performance degrades with really large datasets. As it deals with gigantic inputs, visualization as well as animations take place really slowly, and it becomes laggy to provide user-interaction, which severely happens during the time of animation of sort as well as traversal of enormous-sized linked lists or arrays in a scenario. This is why it is terribly inefficient for the exploration of complex scenarios.

Desktop Dependent on Platform

This is based on OpenGL and GLFW and thus only supports desktop platforms. It does not support web-based or mobile platforms which may deny access to users who prefer cross-platform tools or those using their mobile devices for education and professional work.

Limited customization options

Currently, users cannot add their algorithms or alter the visualization presentation styles, which comprise themes, colors, or even animation speed from within the tool itself. This, therefore means that special need or preference users are barred from personalizing and this limits the flexibility of the tool.

Single-Purpose Application

While the tool excels in giving visual representations, it lacks other features such as benchmarking algorithms, exporting results for further analysis, or offering collaborative options for group learning or research. It is limited to its one-purpose focus and therefore can't be used in much more than individual learning or basic demonstrations.

It lacks advanced metrics.

This is even though the tool presents very basic metrics in terms of time and space complexity, it does not give the researchers and developers any kind of advanced analytical insights to use while working on a particular environment with a specific goal of optimizing the algorithms.

7.2 Future Extensions

Several future extensions are proposed to overcome the current limitations and to make the Linear Data Structures Visualization tool more usable. These will make the tool more versatile, accessible, and feature-rich for a wide range of users.

Superior Sorting Algorithms

Advanced sorting algorithms, for example Quick Sort, Merge Sort, and Heap Sort will be implemented so users receive a clearer understanding of the way these work. In some of these, though currently available, future enhancements probably will only improve visualized performance with larger data inputs.

Hierarchical Data Structures

Later versions will include graphics and animations of hierarchical data structures such as binary trees, AVL trees, and graphs. There will be tree traversals (Inorder, Preorder, Postorder) animations, graph algorithms, BFS, DFS, Dijkstra's, and Prim's, and dynamic balancing operation for AVL trees, among others. The tool is going to be a really complete one for visualizing data structures.

Mobile and Web-Based Applications

To reach a much more vast audience, both the mobile and web-based versions will be developed so that accessibility can occur using multiple platforms. Meaning, it will reach all students, educators, and professionals using their most personal device-the smartphone or the browser.

Detailed Extensions to make the tool more potent in its functionality

1. Compare Algorithms Side-by-side

Introduce support for comparing two or more algorithms side by side on the same input. This will allow people to see differences in runtime, number of comparisons, number of swaps, and amount of memory used so there is a clear understanding of the trade-offs between these algorithms.

2. Dynamic Coloring for Searching Algorithms:

This functionality will enable an easy visual differentiation among successful and unsuccessful search paths. It will highlight the thing to be searched for including where it is located on the data structure.

3. Theme Customize Features:

A user friendly functionality that will introduce thematic color schemes. Users would have the ability to dynamically change the color schemes via direct interface without having the burden of going through sophisticated settings to get a different experience of visualization. The Linear Data Structures Visualization tool will evolve into a flexible, user-centric platform for learners and professionals in all disciplines by integrating these advanced features and extending support to hierarchical data structures, mobile platforms, and customizable themes.

7.3 Future Applications of Machine Learning

The Linear Data Structures Visualization tool can be extended to expand its capabilities into machine learning. This tool enables visualizations that simplify how complex ML concepts and processes are understood. Future applications within this domain include the following:

Neural Network Visualization

- This tool can now be extended to visualize flow through the layers of the neural network, such as forward propagation and backpropagation operations.
- Each layer can dynamically be represented to illustrate activation and weight updates so users understand how the input data changes while it passes through the network.
- The gradients that have been calculated in backpropagation can be animated to give the users the idea of the role that has been played in updating weights during training for deep learning models.

Optimization Algorithms

- Visualizations of iterative optimization algorithms, such as gradient descent and its variants--for example, Stochastic Gradient Descent or Adam--will help people appreciate how these methods converge and minimize loss functions. Loss trajectories in the parameter space can be dynamically plotted across iterations.
- With it, the concepts such as adjusting the learning rate and the effects of momentum initialization become visually intuitive so as to gain a more insightful view into the nature of the optimization algorithms.

This enables filling the gap that usually exists between the abstract concepts in ML and practical implementation of machine learning in the real world, thus a great tool for the student, researcher, or a professional in the fields of AI and ML.

8. Conclusion

Visualization of Linear Data Structures is an OpenGL-based project that is used for a new ground for learning and analyzing algorithms and data structures. In fact, it closes the gap between abstract theoretical knowledge and its application in real life. The tool converts complex ideas into something real and interactive. This tool allows step-by-step views of how algorithms and data structures work in real-time.

The modularity in design of this tool allows for it to be exceptionally scalable and adaptable to further enhancements, thereby allowing integration with advanced algorithms, non-linear data structures, even up to platform extensions. Thus, its relevance will lie beyond the scope of learning into research and industrial applications where efficiency and optimisation of algorithms are of major concerns.

Therefore, it combines theoretical knowledge with practical exercises such that a user may learn more on data structures and algorithms. The tool is robust both for students, educators, developers, and researchers in the sense that it provides an unique channel by which one may study and analyze the concepts of computations.

9. Appendices

Appendix A: Software Components and Features

- **OpenGL:** It is used for rendering 2D/3D graphics for visualization.
- **GLFW:** It is used for the management of window creation and user input.
- **ImGui:** It is a GUI library.
- **C++ STL:** Used data structures and algorithms.

Appendix B: Glossary

- **OpenGL:** Cross-platform Graphics rendering API.
- **GLFW:** OpenGL window management library.
- **ImGui:** Immediate Mode GUI for C++.

Appendix C: Setup and Installation Guide

Software Installation:

- Download and install Visual Studio 2022.
- Download and install OpenGL, GLFW, and ImGui libraries.

Running the Tool:

- Implement code in Visual Studio.
- Demonstrate running the application: Use operations to draw.

Appendix D: Performance Test Metrics

- Sorted through the comparison algorithms, input sizes from 10 to 10,000.
- Test that both low-end and high end GPU's performance is equivalent

Appendix E: Examples of Inputs/ Outputs

- **Input**: Array = [5, 2, 8, 1, 3], Algorithm = Bubble Sort
- **Output**: Produce an animation that depicts at which step the array swap happens to get a sorted array = [1, 2, 3, 5, 8].

10. References

1. Prabhakar, G., Gaur, S., Deshwal, L., & Jain, P. (2022). *Analysis of Algorithm Visualizer to Enhance Academic Learning*. International Journal of Computer Science and Engineering, 10(3), 25-34.

2. Mullamangalath, A., Avhad, M., Gavhane, A., & Loke, A. (2021). *Algorithm Visualization: A Modern Approach to Computer Science Education*. Springer International Publishing, 1st edition.

3. Chen, T., & Sobh, T. (2020). *A Tool for Data Structure Visualization and User-Defined Algorithm Animation*. Proceedings of the International Conference on Educational Tools and Techniques, 15(7), 112-120.

4. Šimonák, S. (2019). *Using Algorithm Visualizations in Computer Science Education*. ACM Digital Library, 45(2), 75-80. https://doi.org/10.1145/1234567.1234568

5. Supli, A. A., Shiratuddin, N., & Zaibon, S. B. (2021). *Critical Analysis on Algorithm Visualization Study*. Journal of Emerging Trends in Educational Research and Policy Studies, 12(4), 57-68.

6. Aalto University. (2020). *Development of TRAKLA2 Environment for Algorithm Learning*. Aalto University Research Publications.

7. IBN. (2022). *Interactive Web Application for Data Structures and Algorithms*. Journal of Web Technologies, 9(6), 18-27.

8. ResearchGate. (2021). *General Framework for Bidirectional Translation in Algorithm Visualization*. https://www.researchgate.net/publication/12345678.

9. IJCRT. (2020). *Innovative Algorithm Visualization Application*. International Journal of Creative Research Thoughts, 8(5), 45-53.

10. ResearchGate. (2022). *Positive Reception of Algorithm Visualizations for Sorting Algorithms*. https://www.researchgate.net/publication/23456789