Dissertation on

"Data Structure & Algorithm Visualizer"

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology
in
Computer Science & Engineering

UE18CS390B – Capstone Project Phase - 2

Submitted by:

R. Shrenik	**PES1201800690**
Niranjan Bhaskar K	**PES1201801486**
Vidhisha Shankar	**PES1201801817**

Under the guidance of

Prof. Kusuma K.V.
Assistant Professor
PES University

June - December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

FACULTY OF ENGINEERING

CERTIFICATE

This is to certify that the dissertation entitled

‘Data Structure & Algorithm Visualizer’

is a bonafide work carried out by

R. Shrenik	**PES1201800690**
Niranjan Bhaskar K	**PES1201801486**
Vidhisha Shankar	**PES1201801817**

in partial fulfilment for the completion of seventh semester Capstone Project Phase - 2 (UE18CS390B) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period June - December 2021. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The dissertation has been approved as it satisfies the 7th semester academic requirements in respect of project work.

Signature	Signature	Signature
Prof. Kusuma K.V.	**Dr. Shylaja S S**	**Dr. B K Keshavan**
Designation	Chairperson	Dean of Faculty

External Viva

Name of the Examiners	**Signature with Date**
1. _____	**_____**
2. _____	**_____**

DECLARATION

We hereby declare that the Capstone Project Phase - 2 entitled “Data Structure and Algorithm Visualizer” has been carried out by us under the guidance of Prof. Kusuma K.V, Department of Computer Science and Engineering, and submitted in partial fulfilment of the course requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering of PES University, Bengaluru during the academic semester June – Dec 2021. The matter embodied in this report has not been submitted to any other university or institution for the award of any degree.

PES1201800690	**R. Shrenik**
PES1201801486	**Niranjan Bhaskar K**
PES1201801817	**Vidhisha Shankar**

ACKNOWLEDGEMENT

We would like to express our gratitude to Prof. Kusuma K.V (Assistant Professor), Department of Computer Science and Engineering, PES University, for her continuous guidance, assistance, and encouragement throughout the development of this UE18CS390B - Capstone Project Phase – 2.

We are grateful to the project coordinator, Prof. Silviya Nancy J, for organizing, managing and helping with the entire process.

We take this opportunity to thank Dr. Shylaja S S, Chairperson, Department of Computer Science and Engineering, PES University, for all the knowledge and support we have received from the department. We would like to thank Dr. B.K. Keshavan, Dean of Faculty, PES University for his help.

We are deeply grateful to Dr. M. R. Doreswamy, Chancellor, PES University, Prof. Jawahar Doreswamy, Pro Chancellor – PES University, Dr. Suryaprasad J, Vice-Chancellor, PES for providing various opportunities and enlightenment through every step of the way. Finally, this project could not have been completed without the continual support and encouragement we have received from our family and friends.

ABSTRACT

Algorithms and data structures form the foundation of Computer Science. An algorithm is a way to implement a solution to a computational problem while a data structure is just a way to store data. Despite numerous advances in hardware and software technologies over the decades, it is imperative that the problem solver possess the tools and mental models necessary to implement an efficient solution (in terms of time and space) . The right algorithm and data structure combo can make the difference between a solution that runs in milliseconds versus one that takes hours / days to run. Working with large amounts of data to solve complex problems fast is the need of the hour.

Gaining a good understanding of how they fundamentally work at a low level and using them to model computational problems has proven to be a challenging task for both students and tutors alike. Thus, the project aims to bridge this gap by creating a visually-pleasing, intuitive online learning experience to help problem-solvers understand how they work by allowing the users of the web application to input their data, create and see the data structures and algorithms in an animated action.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

The project allows users to create and visualize data structures and algorithms online through intuitive web animations.

This project will allow users to: Enter their own data to create, modify, visualize and therefore understand the core concepts of data structures, time complexity and space complexity in a step-by-step manner and visualize almost all pathfinding algorithms and sorting algorithms.

Further, the user will have the ability to control the animation sequences of each of the visualizations concerned. General animation controls and transition speed controls have also been provided.

The primary goal of the system is to simplify the teaching and learning process of complex topics through an intuitive, animated online learning platform used by anyone wishing to learn these topics.

CHAPTER 2

PROBLEM STATEMENT

To create an interactive online platform to visualize the creation and working of data structures and algorithms using animations on the web. The platform should allow the user to enter their own data, visualize the animated structures and algorithms and perform associated operations on those accordingly.

CHAPTER 3

LITERATURE REVIEW

3.1: A tool for data structure visualization and user-defined algorithm animation

3.1.1 Overview of the paper:

To create an exploration environment where students can learn through experimentation. This would serve as an effective supplement to traditional classroom education.

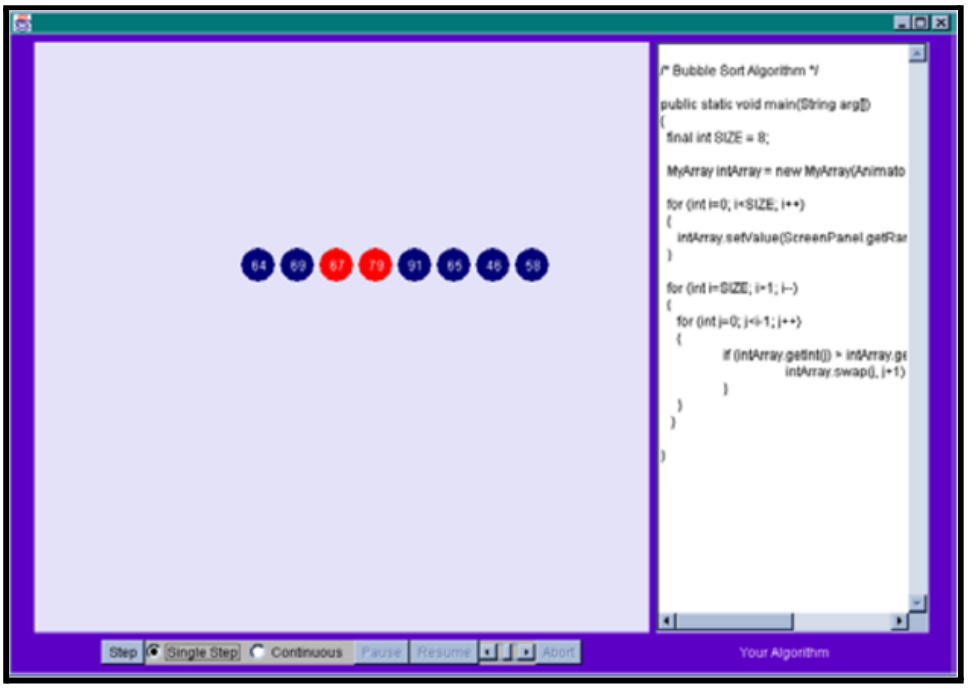
I. A software application that visualizes the most commonly used data structures and their associated operations (only insert() and delete() in this paper).
II. Additionally, the software described can also visualize user-defined algorithms.
III. The paper starts out by describing the need for an interactive visualization tool and existing algorithm animation systems (XTANGO and POLKA) from Georgia Tech which require the user to write a Java program and register events that they'd like to see visually.
IV. Following the above, an overview of the functionalities are provided for various implementations of arrays, stacks, queues, binary search trees, heaps and graphs.
V. For the user-defined algorithms, the user is supposed to code in a Java-like language called JavaMy to instantiate data structures he/she wants to observe. The JavaMy algorithm file is analyzed using a custom parser and observable data structures are added to the frame which are editable.

3.1.2 User workflow:

I. Write algorithm
II. Save File
III. Build and Compile
IV. Upon successful parse, Build and Run
V. Watch animation on the canvas

The code for the Java file, the translation, the parser and the grammar is listed out on an interface on the right.

For instance, here is an animation frame of the application attempting to visualize Bubble Sort:



3.1.3 Possible enhancements to the system:

I. Addition of more complex data structures.

II. Develop and implement a mechanism for the software package to recognize user-defined observable data structures and leave the implementation to the user to provide more flexibility.

III. Highlighting executing command of the user-defined algorithm file. This would help the user follow the execution of the algorithm.

3.1.4 Conclusions drawn:

I. A visualization tool designed to aid beginner CS students grasp concepts of basic data structures has been implemented as a desktop application.

II. The tool allows students to write algorithms in Java and observe their execution, although in a limited respect to the range of data structures available to visualize.

III. A special parser and grammar is defined to parse user-defined algorithms and visualize them on a canvas. This extends the scope of the project in the future.

3.2: Testers and visualizers for teaching data structures

3.2.1 Overview of the paper:

I. The paper presents two tools to support the teaching of data structures and algorithms in Java for the CS2 course at Brown University.
   A. The visualizer which provides the interactive visualizations of user-written data structures.
   B. Testers check the functionality of the user-written data structures.

II. Unlike the previous paper, this approach does not require any modification to the user-defined code. However, students are asked to write generic, reusable implementations that conform to a specific API standard.

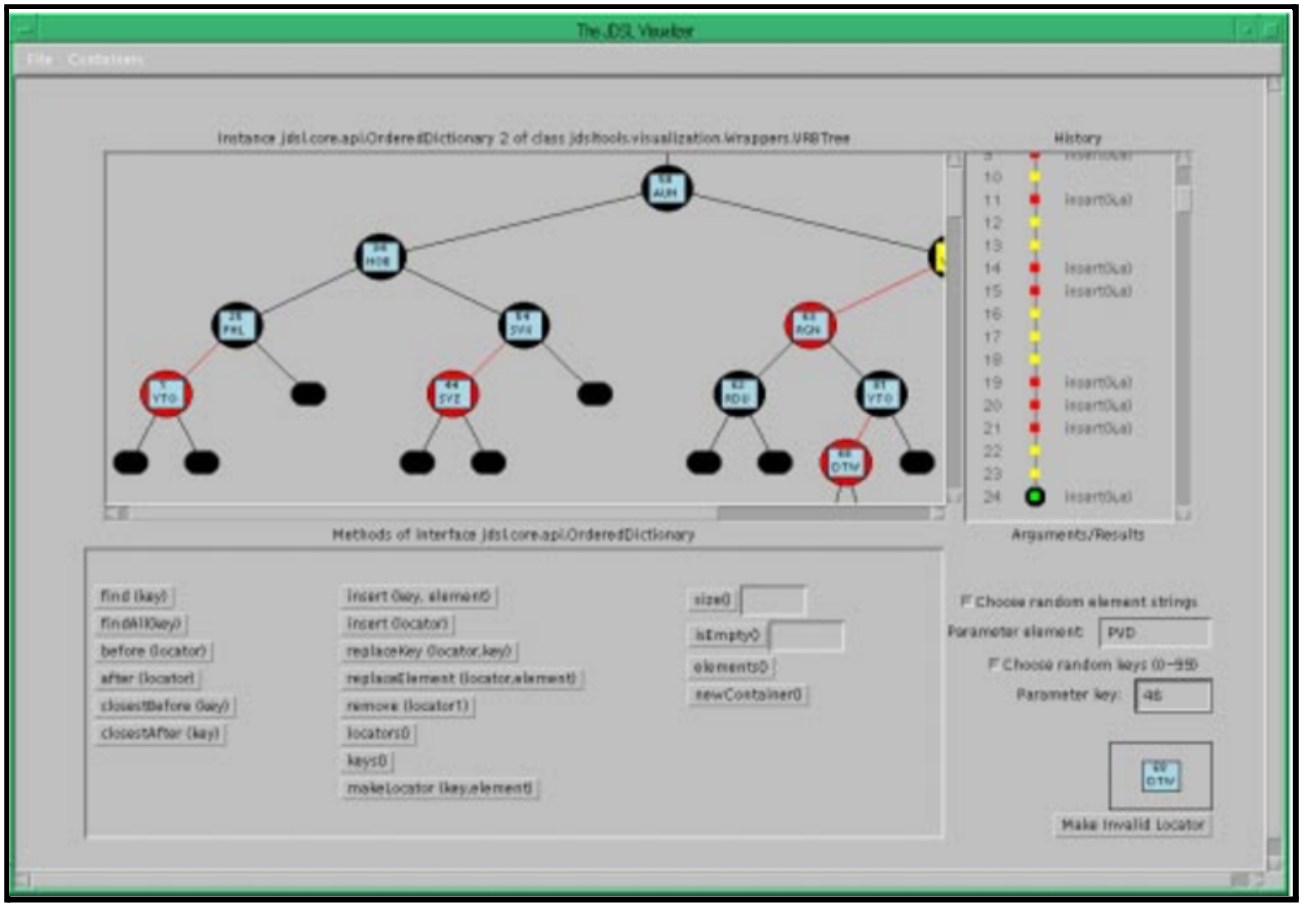
III. The goals include producing visualizations while interfering as little as possible with the design, allowing users to interact with the data structure with methods at runtime and displaying data structures at a conceptual level instead of simply visualizing the contents of the memory.

IV. The paper then describes the main visualizer: The JDSL (Library of Data Structures in Java) visualizer. This visualizer can display interactions on structures (Eg: A binary tree and the sequence of its nodes visited in preorder).

V. Additionally, it also saves the history of the actions performed, the return values, methods invoked on parameters etc. The visualizer also allows the user to have more control over the visuals of the data structure and its behavior using “hooks” which are API-specific methods allowing for more control.

Here’s a snapshot of the interface illustrating the various components of the visualizer, namely:

I. A canvas
II. A history panel
III. A method area, and
IV. A control panel



3.2.2 Implementation of the visualizer:

I. API-specific classes, objects and methods

II. Visualizer updates history, reads the data into the data structure on event occurrence and displays appropriate output on canvas (valid output/ error message).

III. The history is clickable and is iterative. At any point in the history tab, the state of the structure can be seen on the canvas.

3.2.3 Role of testers in the system:

I. JDSL testers provide students with accurate reports on the functionality of their programs. Why? To increase grading efficiency.

II. The testers test the implementation of the student's code and structure. It executes methods on data structures. Uses parsers to interpret string outputs. Compares the reference output to student output with error checking while testing.

3.2.4 Conclusions drawn:

I. The paper provides a unique implementation of visualizers and testers as a desktop application. The application uses API-specific methods to visualize data structures on a canvas. It keeps track of the history of methods invoked and provides real-time configuration of the structure.

II. The tester as a grading tool had been introduced to allow evaluators to test a student's code and structure. It compares string outputs and delivers the appropriate result report to the evaluator.

III. Future work:

A. Generic button implementations for methods (create(), insert(), delete()) across the application with a better UI.

B. Since the course CS2 was tested for quality of the implementation as well, it would serve the evaluators well to include a time and space complexity calculator as well

3.3: Teaching Support for the Visualization of Recursive Algorithms

3.3.1 Overview of the paper:

In essence, the paper lays emphasis on a software project that serves as a teaching tool for the visualizations of recursive algorithms that deals with operations on data structures and more:

I. Sierpinski Triangle Puzzle: Problem attempts to construct triangles in a Sierpinski Triangle recursively taking the level of recursion (n) as input.

II. Sierpinski Curve Puzzle: Problem attempts to construct squares in a Sierpinski Curve recursively taking the level of recursion (n) as input.

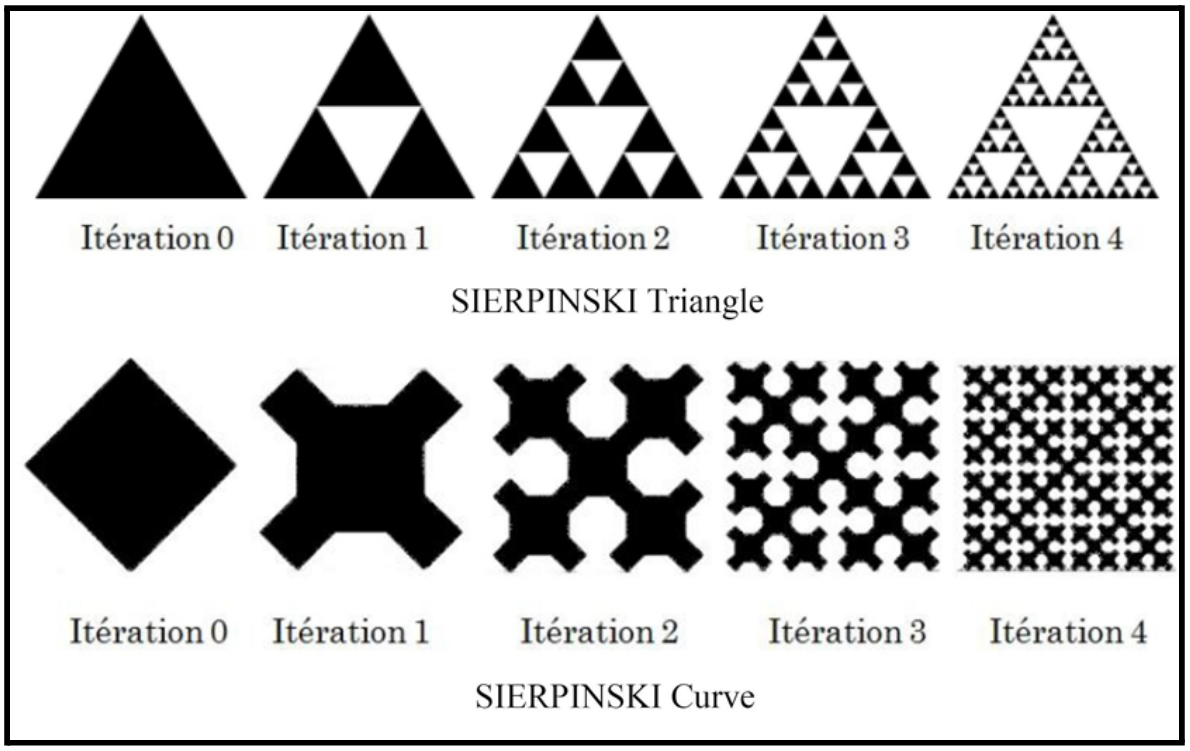
III. Knight's Tour Problem: Algorithm that attempts to compute the closed path of movement of a chess knight on an 8x8 chess board according to the rules of chess.

IV. Eight Queens Puzzle: Problem of placing chess queens on an 8x8 chessboard such that they do not endanger each other according to the rules of chess.

V. Merge Sort: Sorting algorithm that uses the divide and conquer approach to sort a given array of elements.

VI. Max-min: Algorithm that computes the maximum and minimum of the elements in an array.

Here's a demonstration of the Sierpinski Triangle and the Sierpinski Curve puzzles:



3.3.2 Implementation of the system:

I. The application was written from scratch in Java using the JavaFX library for the User Interface.

II. The software tool is considered to be a teaching/learning tool for lecturers and students. The application also provides a dual language interface - Slovak and English.

III. It is a high level visualizing tool allowing users to pause/play the algorithm at any step, stepping forward, stepping backward, changing transition speed, reset, etc.

IV. The application features several input options:

A. Manual user input

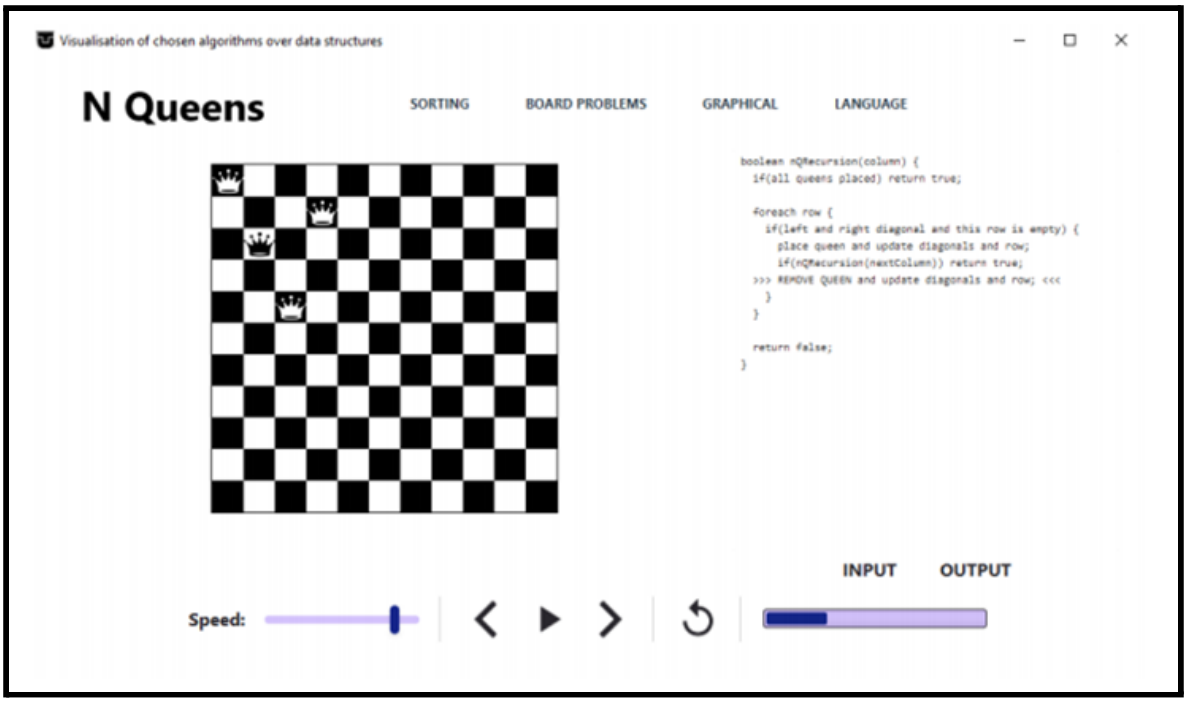
B. Generating random input by clicking a button

C. Retrieving the series of inputs from a file (for instance, an XML)

V. Moreover, the application also allows downloading the visualization images or even the pseudocode at a particular instance as a PDF document.

VI. The project also features a panel with the pseudo code of the running algorithm in it.

Here is a snapshot of the application proposed in the paper running on desktop:



3.3.3 Conclusions drawn:

Thinking recursively might be confusing in most cases although it may have fewer lines of code, having an in-depth understanding about the working of recursive algorithms through visualizations makes it all the way more easier and faster for both lecturers and students.

The application attempts to visualize the six algorithms mentioned earlier using user-defined inputs, animation controls, and finally provides an output - the visualization. It also provides a user interface in Slovak and English language.

However, this application focuses only on the six recursive algorithms specified earlier. Moreover, a web application would have been a better option than a software tool that runs locally on the OS of the device, as for such applications our primary goal is to perform tasks quickly over the internet.

3.4: Effectiveness of Combining Algorithm and Program Animation: A Case Study with Data Structure Course

3.4.1 Overview of the paper:

I. This paper primarily focuses on addressing the effectiveness of combining both Algorithm and Program Animation.

II. In accordance to this paper, there are essentially four components that concern the layout screen of an algorithm visualizer:

   A. Memory Mapping: Depicts a diagrammatic representation of the code's recursion stack.

   B. Code Display Area: Consists of the currently executing code, with pointers to the statement currently being executed.

   C. System Messages Area: This area is devoted to displaying error messages, warnings, or even hints that could potentially guide the user.

   D. User Input Area: This section includes fields to update user-defined inputs and ways to trigger other subroutines (Add(), Delete(), etc.).

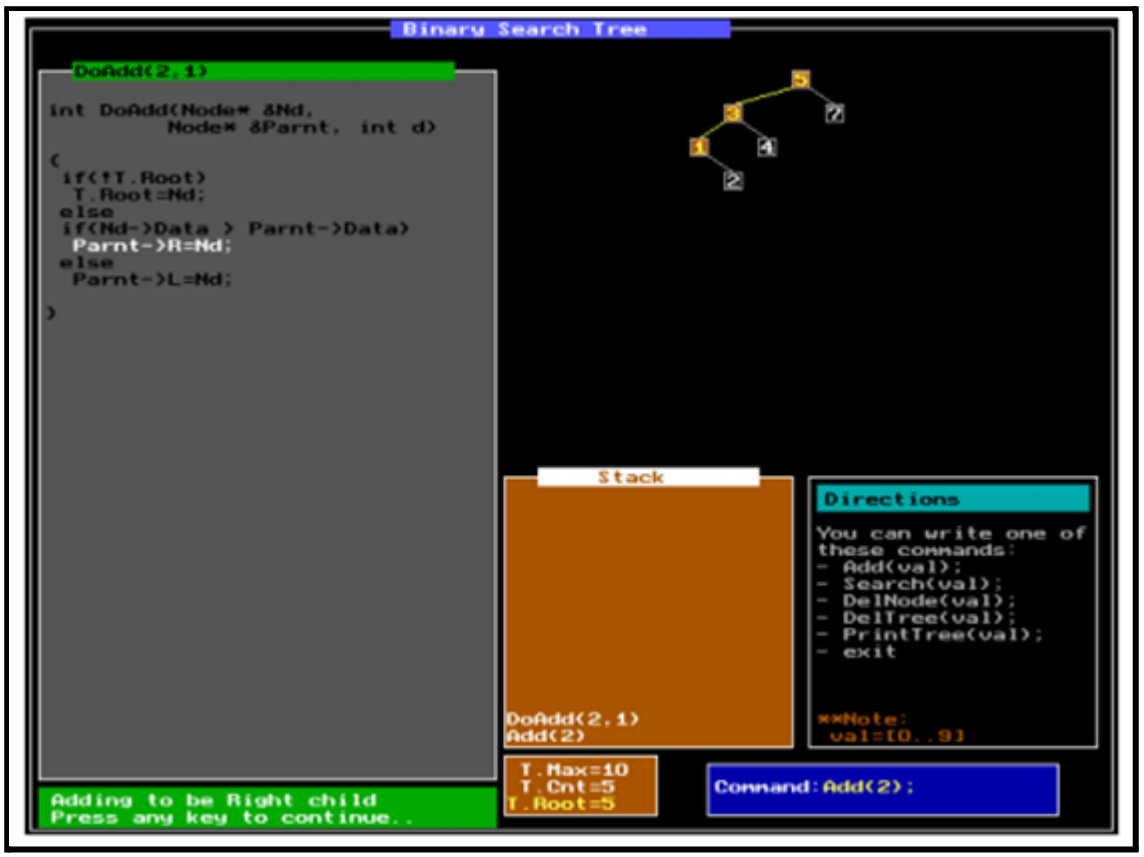
III. The system attempts to visualize operations on data structures including Arrays, Lists, Circular Lists, Stacks, Queues, and Linked Lists.

IV. It is primarily built for high school students to get familiar with basic operations on the above mentioned data structures.

V. It provides interactivity by allowing users to enter valid commands rather than using mouse selections, and the previously discussed stop-pause fashion.

VI. This allows users to observe the effect of executing the concerned statements real-time in correlation with the statements displayed. However, there is no way of reverting back to the previous frame of the animation sequence.

Here is a snapshot of the application specified in the paper:



3.4.2 Conclusions drawn:

I. The implemented system combines both Algorithm Animation and Program Animation.

II. The visualization engine is embedded with the program script.

III. The visualization and use of the system doesn't involve rich interfaces, like a window-based modal form, and using the mouse, so it is clearly simple in order to avoid student distractions but this can also be viewed as a downside as far as the features of the system are concerned.

IV. However, there is no way to seek a specific portion of the animation sequence and the only allowed animation control is “next frame”.

3.5: Overall Literature Survey Conclusion

Visualization System Fundamentals

I. Visual learning tools are very valuable from a practical learning standpoint. The tools built in the literature covered are aimed at solving exactly this problem via an intuitive user-operable interface.

II. Visualizing user-defined algorithms (refer paper 1) are extremely hard to implement since it usually involves converting existing HLL code to a workable implementation requiring custom parsers and grammar.

III. Interactivity is important. To understand how data structures and algorithms work, it is important to understand the iterative process behind their creation and working using data structure-specific methods. Well-tested animation controls are therefore important.

IV. Well-tested bug-free code is necessary. It is important to keep in mind the functional requirements or it might lead to the code breaking on a specific test case potentially leading to undefined behaviour.

Strengths of the various visualization systems surveyed

I. All the papers we studied involved well-tested, robust, generic code (templates) implementations.

II. Since they were desktop applications utilizing the GPU for rendering animations, the structures on the canvas and overall speed of the application was fast.

III. Overall coverage of data structures was good. Basic data structures, recursive algorithms and sorting algorithms were implemented.

Pointers to keep in mind while building the application

I. Considering that web applications are way more accessible and are ideal to be used in cases such as this where performing tasks quickly over the Internet is primary, we have chosen this over building a cross-platform desktop application locally.

II. Since the literature we reviewed was fairly old, it is the case there. Further, we have also come to a conclusion that such desktop applications are more suitable for largely time-consuming tasks which is clearly not the case here.

III. Moreover, a web application can be used by anyone who has access to a web browser and is connected to the Internet.

CHAPTER 4

SYSTEM REQUIREMENTS SPECIFICATION

4.1. Introduction

The purpose of this document is to give a detailed description and the requirements for the Data Structure and Algorithm Visualizer. Illustrating the purpose and complete declaration for the development of this system, elaborating on its general constraints, interface and interactions with other external applications, this document is primarily intended to be used to keep track of the requirements of the project.

The document also formulates the various functional and non-functional requirements of the system.

I. **Functional requirements** define a system or its components. It describes the functions a software must perform. A function takes an input, does some work and generates an output. This function can be a user interaction, computational work or data manipulation among many other operations.

II. A **non functional requirement** on the other hand, defines the quality attribute of a software system. They represent a set of standards used to judge the specific operation of the system.

4.1.1. Project Scope

I. Purpose:

   A. In a very basic sense, the project allows users to visualize data structures and algorithms through web animations.

   B. Developers avoid learning DSA because they seem complicated. The project aims to solve this problem.

C. Programmers who are competent in Data Structures and Algorithms can easily model computational problems, perform tasks conforming to data processing, automated reasoning or calculations.

D. This project aims to solve that one task: “Simplifying the learning of Data Structures and Algorithms through web animations”.

II. Benefits:

A. Self-learning concepts visually has proven to be one of the best methods to retain and apply knowledge.

B. The project allows the user to enter their own data, choose from an extensive list of data structures, perform various operations on them and see the live animated creation and manipulation of the data structure on the web app.

III. Objectives:

A. Visualizing pathfinding algorithms and sorting algorithms.

B. Providing fundamental data structure-specific operations to the user

C. Providing general animation controls such as skipping back to beginning, playing, pausing, step forward to next frame, and step back to previous frame.

IV. Coverage and limitations:

A. A completely bug-free experience is slightly hard to achieve.

B. Website Performance might be an issue.

4.2. Product Perspective

4.2.1. Product Features

The product is intended to provide the enlisted features to the users:

I. Animation speed controls

II. Data-structure operations' visualizations:
   A. create(), insert() and delete() etc are some common operations.

III. Visualizing the following pathfinding algorithms:
   A. Depth First Search
   B. Breadth First Search
   C. Dijkstra's Algorithm
   D. Kruskal's Algorithm
   E. Prim's Algorithm

IV. Visualizing the following sorting algorithms:
   A. Bubble sort
   B. Insertion sort
   C. Selection sort
   D. Merge sort
   E. Quick sort
   F. Random Quick sort
   G. Counting sort
   H. Radix sort

V. Linear data Structures (some / all variations of + specific functions):
   A. Linked lists
   B. Stacks
   C. Queues

VI. Non-linear data structures (some / all variations of + specific functions):
   A. Binary Search Trees
   B. Heap Trees
   C. AVL Trees
   D. Conversion of Tree to Binary Tree and vice versa

VII. Others:

A. Pattern Matching: Simple & KMP Matching

B. Linear Search & Binary Search Algorithms

4.2.2. Operating Environment

I. Any HTML5 and JavaScript ES6 compliant browser will be able to render animations smoothly.

4.2.3. General Constraints, Assumptions and Dependencies

I. **No mobile-website or small-screen support**: Due to the need for working with complex visualizations that require lots of pixels and click ‘n drag gestures for active interaction.

II. **Interfaces to other applications:**

A. No API provision by the developers.

B. No third-party access should be allowed.

III. **Safety and security considerations:**

A. DDoS attacks: Overloading the server with too many requests can lead to extensive downtime. Request limits must be set.

B. XSS attacks: XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users.

C. Directory Listing: Hiding the website index containing important data is mandatory.

D. Basic admin/user authentication and an HTTPS certificate are a must.

IV. **Limitations of simulation programs:**

A. Devices must be connected to the internet to access the web application.

B. The web browsers used in these devices must be HTML5 and ES6 compliant.

C. No mobile-device support.

4.2.4. Risks

Developers should:

I. Hide the main directory on the website to deny unauthorized users access.
II. Keep in mind DDoS attacks. Overloading the server may lead to extended periods of downtime.
III. Disallow foreign extensions from accessing website data.
IV. Test code and check for memory leaks.

As such the application does not require any mode of user authentication on the client side/attempts to capture user information. Therefore, it doesn't entail any serious privacy constraints and associated risks.

4.3. Functional Requirements

4.3.1. Validity test on inputs

I. Visualizing operations on Data Structures
    A. Valid input with appropriate data types (whether integer, character, etc.) must be provided while inserting elements into the data structure.
    B. Further, validating the values while constructing a binary search tree such that it follows the basic definition of a binary search tree is necessary.
II. Visualizing Pathfinding Algorithms
    A. Obstacles cannot be inserted on start/destination nodes.
    B. Basic animation controls
        1. Step backward button must be disabled when it is already at the beginning of the animation sequence.
        2. Step forward button must be disabled when it is at the end of the animation sequence.

4.3.2. Sequence of operations

I. Visualizing Pathfinding Algorithms

A. Insert Start, Destination Nodes.

B. Insert Obstacles (if required).

C. Set animation speed.

D. Visualize!

II. Visualizing operations on Data Structures

A. Execute the required operation on the data structure concerned (say, insert(), delete(), search(), etc).

B. Use any of the previously mentioned animation controls to skim through the animation sequence with ease!

C. Disable operations when necessary.

III. Visualizing Sorting Algorithms

A. Execute the required operation (whether it is creating custom inputs to run the sorting algorithm on or, executing the selected sorting algorithm).

B. Use any of the previously mentioned animation controls to skim through the animation sequence with ease!

4.3.3. Error Handling and Recovery

I. Visualizing Pathfinding Algorithms

A. Non-existent paths from a source node to a destination node must return an appropriate error and not potentially break the application.

II. Visualizing operations on Data Structures

A. Delete/Search operations on a non-existent key under data structure specific operations must return appropriate errors.

III. Basic animation controls

A. Moving to a frame in the animation sequence that might return the above listed errors must be properly conveyed to the user.

4.4. External Interface Requirements

4.4.1. User Interfaces

I. Required screen formats with GUI standards for styles:
   A. Minimum Screen Size: 1280x720 pixels. (Ideally 1920x1080)
   B. GUI guidelines: https://material.io/design/guidelines-overview

II. Screen layout and standard functions (e.g. help).
   A. Layout: Landscape
   B. Standard functions: help, exit, animation controls.

III. Relative timing of inputs and outputs.
   A. Inputs are entered by the user. Zero-delay entry.
   B. Generated outputs are highly dependent on the operation being performed, the data structure and/or algorithm involved and the amount of data being manipulated.

IV. Availability of some form of programmable function key.
   A. The project would ideally support hot-keys for animation controls. (Not a priority)

V. Error messages
   A. Error messages and alerts clearly indicating the issue will be displayed in case of invalid inputs, illegal operations on the structures and algorithm visualizations or invalid hot-key combinations.

4.4.2. Hardware Requirements

I. Physical Interfaces
   A. A computer with a working NIC and a reliable connection to the Internet.
   B. Requires a web server to manage requests from the client.

II. Logical Interfaces
   A. A web browser that supports HTML5 and is ES6 compliant.

4.4.3. Software Requirements

I. Frontend UI components:
   A. HTML5, CSS3, VanillaJS ES6, jQuery (Event Handling).
   B. These languages and libraries are mandatory for creating the user interface of the website. HTML defines the structure, CSS defines the look of the website, JavaScript adds functionality to the UI component.

II. Operating Systems:
   A. No support for mobile operating systems including Android and iOS.
   B. Any other desktop OS with HTML5 and ES6 compliant web browser is supported.

4.4.4. Communication Interfaces

I. **HTTP:** For reliable access to the Web Interface, the application requires port 80 to facilitate communication with the client's browser.

II. **HTTPS:** For reliable and secure access to the Web Interface, the application requires port 443 to facilitate communication with the client's browser.

4.5. Non-Functional Requirements

4.5.1. Performance Requirements

I. The web application mustn't be affected largely by increased network traffic.

II. Higher request-response speeds can be achieved by increasing compression, optimizing images, CSS, and opting for better hosting solutions.

III. Response times on the website should not take >1s ideally (ignoring client side constraints) for active CRUD operations.

IV. **Software Quality Attributes:**
   A. Use browser-native JSON format to send/receive information to/from the server. Avoid XML for better application performance.
   B. Style guide to be followed : JS Style Guide

4.5.2. Security Requirements

I. Limiting request-response actions per user to prevent overloading and deny service.

II. Unlisting main website directory from public view to keep sniffers and URL Fuzzers at bay.

III. Memory leaks and other exceptions within data structure and algorithm implementations to avoid undefined behavior and crashes.

CHAPTER 5

HIGH LEVEL DESIGN

I. User Group: Students and Teachers wanting to learn DS & A

II. Application Components (planned):

   A. Visualization Library

   B. Visualization Widgets

   C. State Objects

III. Interfacing System: Web User interface built with either pure HTML, CSS and JavaScript.

IV. High-level interaction between application components:

   A. Visualization Widget Working:

      1. Viz. widget receives user input. Input states that it needs to visualize a certain function for a specific DS/A.
      2. Viz. widget creates state objects which describe the state of the DS/A from initial state to the final state.
      3. All this info is passed to <X>Widget.js which will manage how this information is shown to the user.

   B. Visualization Library Working:

      1. The library receives an array of state objects.
      2. The library begins to display the structure as the state objects. ([0 to size-1] of the state objects array)
      3. At any point of time, the user can directly change the displaying mechanism by modifying the animation speed of the visualization concerned.

   C. State Object: It is a JS object containing information on what to draw on the UI. This object is passed into the library to be drawn on the UI.

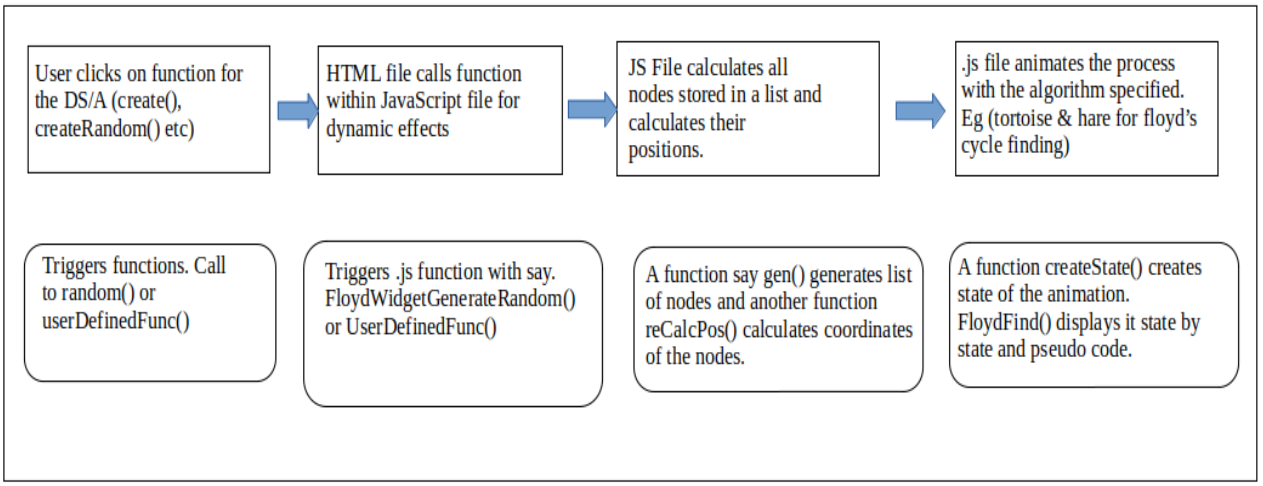
      1. The state object will contain coordinates of vertices and edges, their CSS attributes etc.

V. Code organization:

   A. Master Repository

      1. CSS Files

2. Fonts
3. Static Files (images)
4. JavaScript files
    a) visualization library
    b) widgets
5. HTML Files
6. Documentation and README
7. The application's code will be organized using files and folders, modules and



packages and objects.

8. *Note: As a general rule of thumb, for projects with < 30,000 LOC code organization is not a very important consideration.*

VI. Security: Largely dependent on the hosting platform. The system will be hosted on an HTTPS certified website if possible.

CHAPTER 6

SYSTEM DESIGN

Keeping long term development in mind, the application uses an MVC (Model View Controller) structure. The system consists of 3 parts including the visualization library, visualization widgets and state objects. These 3 modules interact by passing objects between one another. The system makes use of the **jquery** library and uses plain **HTML, CSS and JavaScript** to create and render the web-based UI functional. The project uses a visualization library, widgets and state objects.

6.1 Design Considerations

6.1.1. Design goals

I. The proposed model aims to create an interactive, user-friendly modular online platform with easy extensibility, high-performing animations with preferably reusable components for an easier and faster development/maintenance experience. The UI would involve providing the user with controllable animation controls while creating and manipulating data structures, algorithms and their associated operations.

II. Existing systems are not extensible i.e addition of new components to the website would require manually editing HTML files, writing specific DS/A specific functions that interact with the UI without an abstraction layer and therefore the proposed model makes use of widgets that abstract away the UI rendering details from the logic.

III. **Speed**: The animations are smoothly rendered on the UI with no lag. A good internet connection is required.

IV. **Security**: The website doesn't inherently deal with any sensitive content and there is no data collection. The hosted website will take care of vulnerabilities like directory listing. The project code is written keeping security in mind.

V. **Availability**: Subject to the hosting plan we end up choosing.

VI. **Privacy**: No risks. No data collection taking place.

6.1.2. Architecture choices

I. **The visualization library** consists of existing JS files (some existing, some added, some edited) that define basic drawing elements used for complex animations involving different kinds of structures (nodes and edges). For each DS/A we make a widget relating to an HTML file. This controls the animation of each of the operations as well as the displaying of the code by manipulating the DOM.

II. **Visualization Widgets:** These widgets contain the logic of the data structures and algorithms visualized. It passes the objects to the graph library which will be reflected on the UI. We make use of the Canvas API and the library to do so.

III. **State Object:** It is a JS object containing information on what to draw on the UI. This object is passed into the library to be drawn on the UI. The state object will contain coordinates of vertices and edges, their CSS attributes etc.

IV. **Benefits**:

A. Reusable visualization library: The library can function independently from the visualization widgets. As long as an array of state objects is supplied, it can display the DS/A on the UI => Other future tools can use this library to achieve the same look and feel.

B. Modularization: The library only contains the rendering information, separating the logic and the UI. Also, the library we modify will directly manipulate the DOM objects hiding the rendering details from the viz. logic => easier to manage code.

V. Drawbacks:

Development time.

Direct DOM manipulation can slow down rendering.

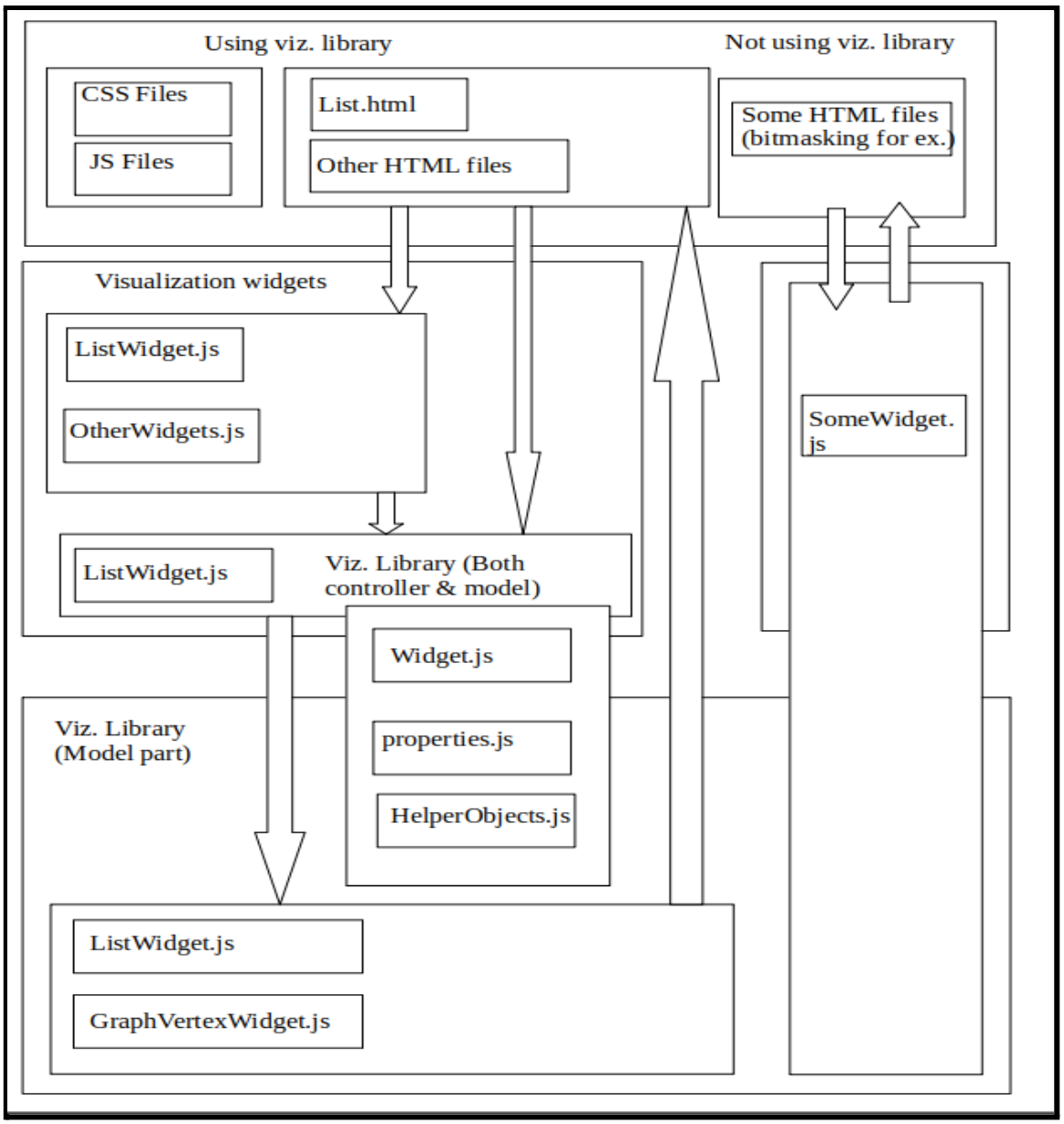


Fig: Overall Architecture Diagram denoting the visualization library and the widgets

6.1.3 Constraints, Assumptions and Dependencies

I. Extensibility and effort: Functions (start,end,algo) were doing the rendering, not the widgets. This means that new tools can't use existing code and we'd have to reimplement the UI every time we want to add something. Abstraction using widgets solves this problem.

II. Animation efficiency: In our initial implementation, when a user decided to return to the previous frame, the library replayed the entire animation from the first frame to the requested frame => Noticeably slow. Now, the visualizations can jump b/w frames in O(1) time since all that needs to be done is to load the state object for that frame.

III. Availability and security are dependent on the hosting platform and the payment-tier we choose.

IV. No mobile access: Screens with a minimum width of 768px is required for smooth animations and interactivity on the platform.

6.2 Master Class Diagram

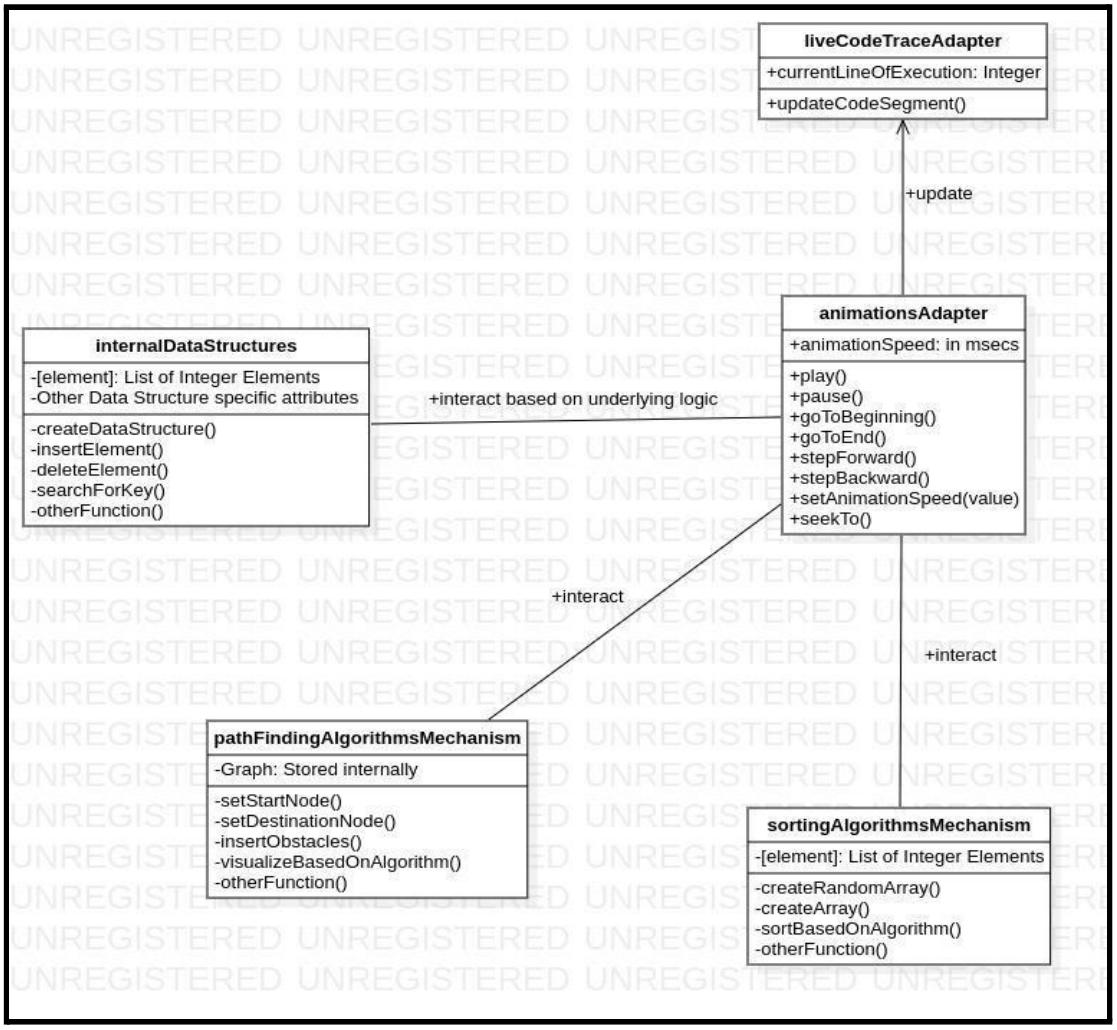


Fig: Master Class Diagram

The figure describes the classes involved in the application along with their class methods. The interactions, updates have also been indicated.

6.3 User Interface Diagrams

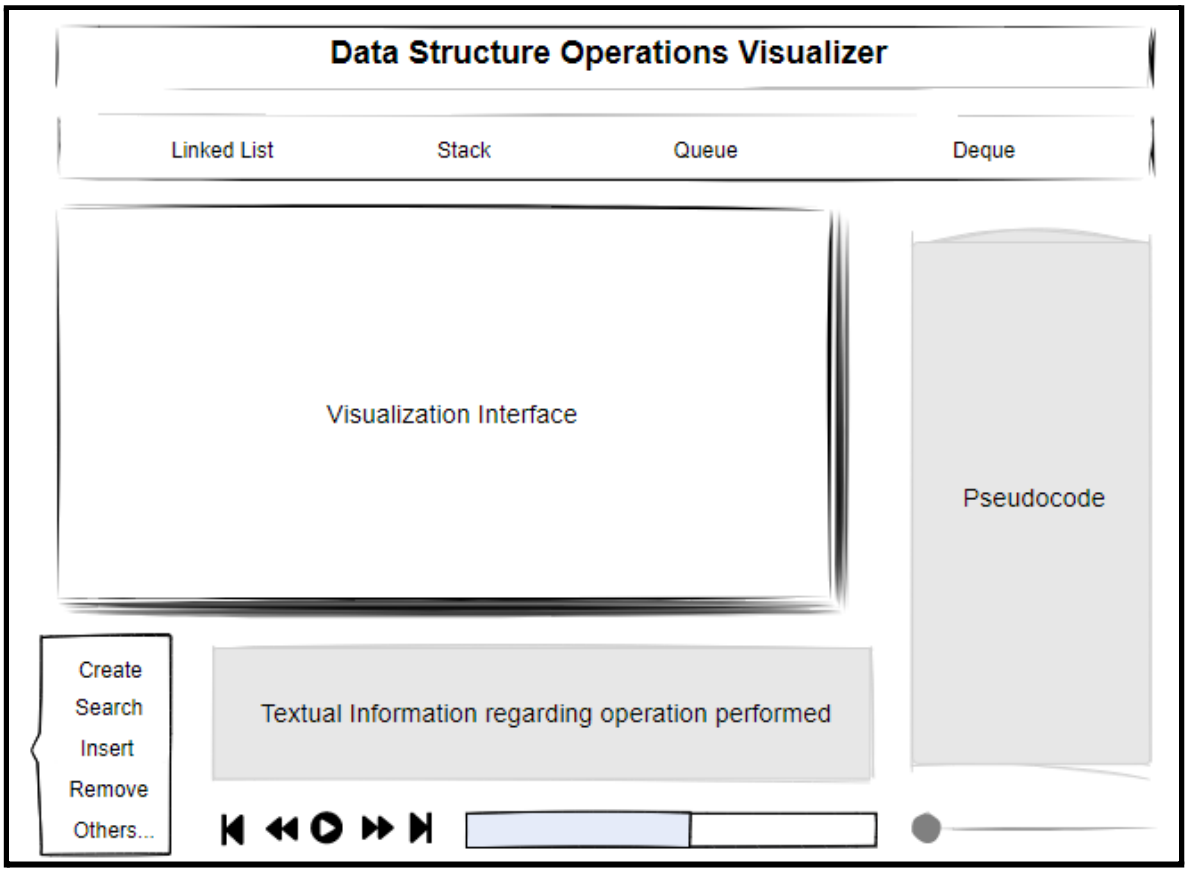


Fig: Current Landing page

The figure is an interpretation of the UI of the current landing page. Each section of the image is self-explanatory. The option to select the DS/A, the operations, the code tracer panel and media controls are present.

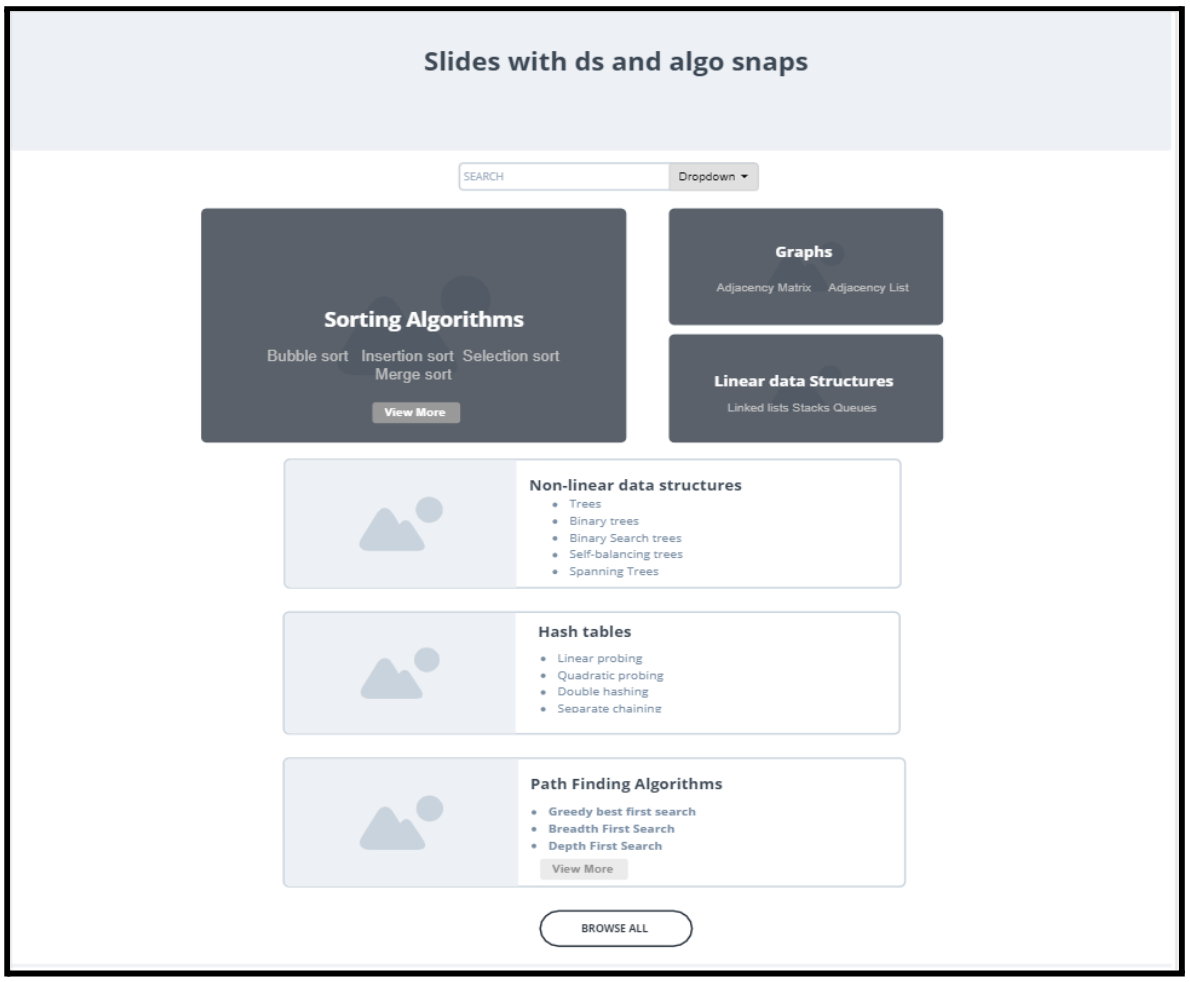


Fig: Sections of DS & A available for visualization (temporary)

This is a temporary mock image of the sections the website might have indicating the DS/A that have been implemented.

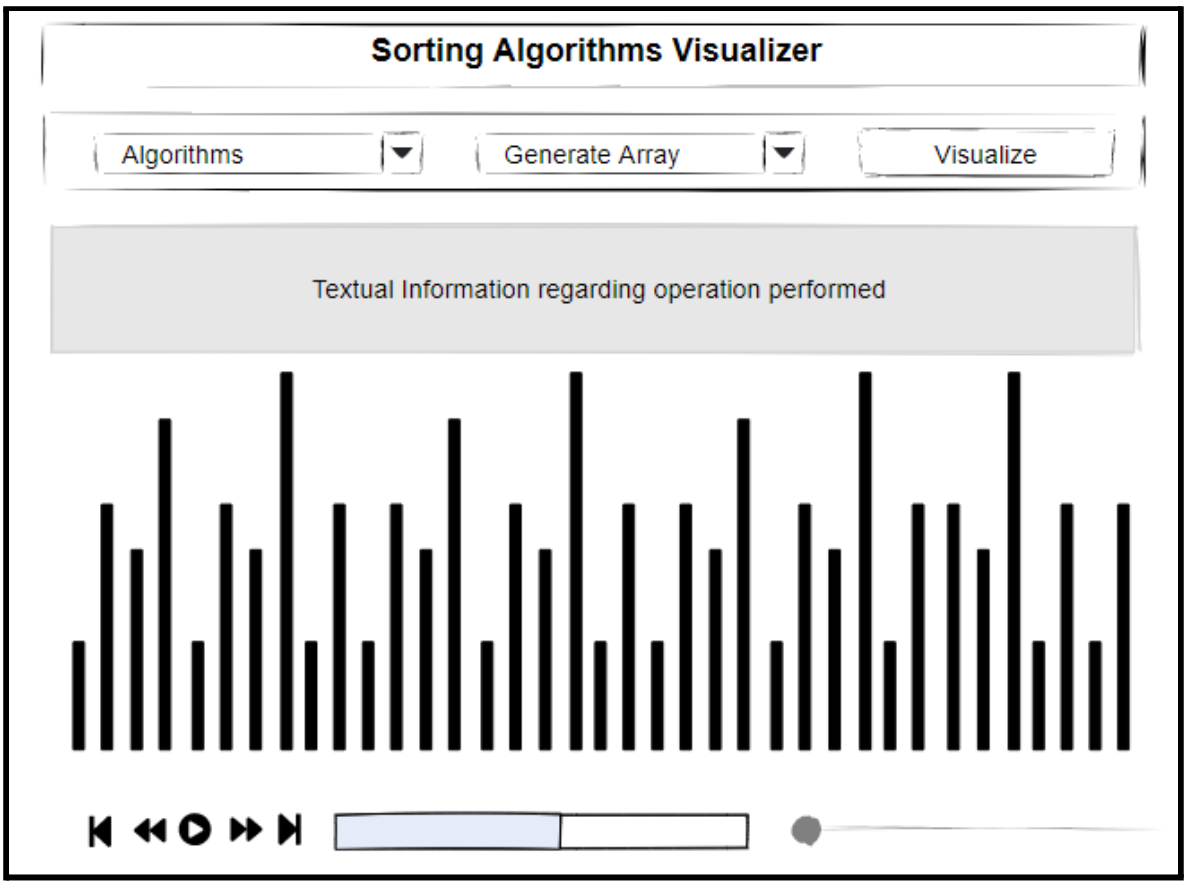


Fig: Sorting algorithm visualizer

The above diagram describes the sorting visualizer with bars indicating size of integers and the options on top indicating the actions to manipulate the data and perform the self-describing operations.

6.4 Reusability Considerations

1. The project takes into consideration extensibility and reusability of components both from a frontend perspective as well as an animation generation perspective.

2. Reusable visualization library: The library can function independently from the visualization widgets. As long as an array of state objects is supplied, it can display the DS/A on the UI => Other future tools can use this library to achieve the same look and feel.

CHAPTER 7

CONCLUSIONS OF CAPSTONE PROJECT PHASE-II

7.1 Achieved deliverables:

I. Linear data structures (linked lists, stacks and queues) implemented
II. Binary Search Trees, Heaps , AVL Tree, Binary tree <-> tree conversion implemented
III. DFS, BFS, Dijkstra, Prim and Kruskal algorithms implemented for graphs
IV. Insertion, Bubble, Selection, Quick, Counting Sort, Merge and Radix Sort implemented
V. Basic string operation algorithms: Simple match and KMP implemented

7.2 Conclusion:

I. The idea of a platform to visualize data structures and algorithms, see them in action and the ability for the user to perform operations on them is vital to learning how they work. It is important to note however that the platform only acts as a supplement to traditional methods of teaching and not as a replacement. Offline doubt-clarification and additional problem solving are important.

II. Working on building well-tested, error-free implementations is more important than covering a large set of data structures since a buggy implementation will confuse users and drive them away. A lot of code for the application is reusable and a modular approach is to be followed.

III. The platform must be fast and visually pleasing and should have low downtime to appeal to teachers and students.

IV. As an essential feature of the visualization system, the several animation controls feature benefit the user groups largely allowing them to seek through any portion of the animation sequence.

V. In addition to that, implementing the live code tracing feature is quite time consuming in terms of development time, and it is not a necessity for the remaining course of the project.

VI. Moreover, such an application that helps its users to visualize the core concepts in Data Structures and Algorithms, be it basic data structure operations or sorting algorithms or even the diverse pathfinding algorithms will largely help the user group concerned whether they are teachers, or students.

APPENDIX A: ACRONYMS, ABBREVIATIONS AND DEFINITIONS

1. **JSON**: JavaScript Object Notation. It is a popular format of storing and moving data.
2. **XML**: eXtensible Markup Language. The language defines rules and encoding formats which are both human and machine readable
3. **CRUD**: Create, Read, Update and Delete. These operations performed on the database when interacting with the web application.
4. **DDoS**: Distributed Denial of Service is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet.
5. **NIC:** Network Interface Controller is a chip on the motherboard of the user's PC allowing the computer to connect to the internet.
6. **XSS:** Cross(X)-site scripting is a type of security vulnerability typically found in web applications. XSS attacks enable attackers to inject client-side scripts into web pages viewed by other users.
7. **MVC:** Model-View-Controller Architecture, essentially separating out the data structures involved, user interfaces, and the core business logic of the application.
8. **DS & A:** Data Structures and Algorithms
9. **DOM**: Document Object Model. It represents an object with a tree.

APPENDIX B: REFERENCES

1. JavaScript Style Guide:
    - https://google.github.io/styleguide/jsguide.html
    - Designed and maintained by Google.
2. Material Design Style Guide:
    - https://material.io/design/guidelines-overview
    - Designed and maintained by Google.
3. Cyber Security Risks For Modern Web Applications: Case Study Paper For Developers And Security Testers by Devanshu Bhatt.
4. Tao Chen and T. Sobh, "A tool for data structure visualization and user-defined algorithm animation," 31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No.01CH37193), Reno, NV, USA, 2001, pp. TID-2, doi: 10.1109/FIE.2001.963845.
5. Osman, Waleed & Elmusharaf, Mudawi. (2014). Effectiveness of Combining Algorithm and Program Animation: A Case Study with Data Structure Course. Issues in Informing Science and Information Technology. 11. 155-168. 10.28945/1986.
6. Teaching Support for the Visualization of Selected Recursive Algorithms Baraník, Róbert and Steingartner, William. ipsitransactions.org/journals/papers/tar/2021jan/p3.pdf
7. Ryan S. Baker, Michael Boilen, Michael T. Goodrich, Roberto Tamassia, and B. Aaron Stibel. 1999. Testers and visualizers for teaching data structures. SIGCSE Bull. 31, 1 (March 1999), 261–265. DOI:https://doi.org/10.1145/384266.299779