LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

Data Structure and Algorithm Visualizer

UE18CS390B – Capstone Project Phase – 2

Submitted by:

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

Under the guidance of

Prof. Kusuma KV
Assistant Professor
PES University

August - 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

TABLE OF CONTENTS

1. Introduction

1.1. Overview

The document provides an overview of the tools, languages and methods used to implement the data structure and algorithm visualizer. It first mentions the declaration of variables, necessary functions in a high level and other starter code. Once these are defined, it is then possible to create the views to display with subsequent visualizations. This is followed by all of the object declarations that model all the auxiliary data structures involved which are required for the traversal algorithms to work. mouseup, mousedown and other events are fired at a general base class Element which allows for mouse control. Upon implementing these controls, all the elements including buttons and menus are created. Finally, functions for creating nodes via the algorithm to thereby visualize it, inserting, deleting and also rendering the error messages and results and helper functions are implemented.

1.2. Purpose

The document delves into some of the details around the design and implementation of the data structures and algorithms visualization tool.

1.3. Scope

The LLD covers, in detail, all of the functional and non-functional requirements of a modern data structures and algorithms visualization tool. Design descriptions of components, the master class diagram, the package deployment diagram have been added. Furthermore, the overall approach and code/pseudocode of key algorithms involved have been stated. The document, however, does to some extent, refrain from diving deep into actual JavaScript and CSS code and we have tried our best to explain each of them by glorifying the ones that will help us understand the overall working of the system.

2. Design Constraints, Assumptions, and Dependencies

- Addition of other algorithms (in addition to the ones already implemented) would require multiple changes across functions.
- No dependence on external JS libraries. The project is implemented in vanilla JS.

- No ES6 code.

3. Design Description

1. Variable declarations

- Global variables that define canvas size, other sizes such as margins, padding, and node size, and other boolean variables used across methods throughout the project are declared initially.
- Local variables for each component. For instance, in the case of sorting algorithms visualizer we will encapsulate the values of the concerned array in a JavaScript Array object.

2. Creation of the canvas

- The canvas is generated using HTML canvas which has its own height and width and is contained in a scroll view of a div.

3. Object declarations

- Both in-built and external implementations of data structures with their associated functions are required to hold auxiliary data for the concerned data structures and algorithms visualizers. We make use of appropriate data structures depending upon which algorithm is visualized.

4. Triggers for functions

- All of the necessary input are encapsulated into traditional HTML forms with the appropriate form fields along with a button that actually triggers the function concerned.
- This mapping is done using event listeners for mouse clicks and is triggered when this click event occurs.

5. **Creation of buttons and menus**
    - The form fields as mentioned earlier are encapsulated within divs with appropriate submit buttons which actually transmit the required function into the visualizer interface which is a unique implementation for each section of the data structures and algorithms visualizer.
    - Need to note that the algorithm desired is also passed along with the data fetched through the form fields, if a common interface is used as in the case of the sorting algorithms visualizers.

6. **Functions for all algorithms**
    - The project visualizes the most practically applicable data structures and algorithms visualizers. All of these functions are implemented that make use of data structures like typical arrays and complex graphs encapsulated within objects to hold “memory” of nodes that need to be traversed or are already traversed. At the current stage of the project, the functions could use some refactoring.
    - Helper functions have been used to split the functionality of large functions involving animation control, functions that are responsible for the canvas render and traversal algorithms into more manageable units.

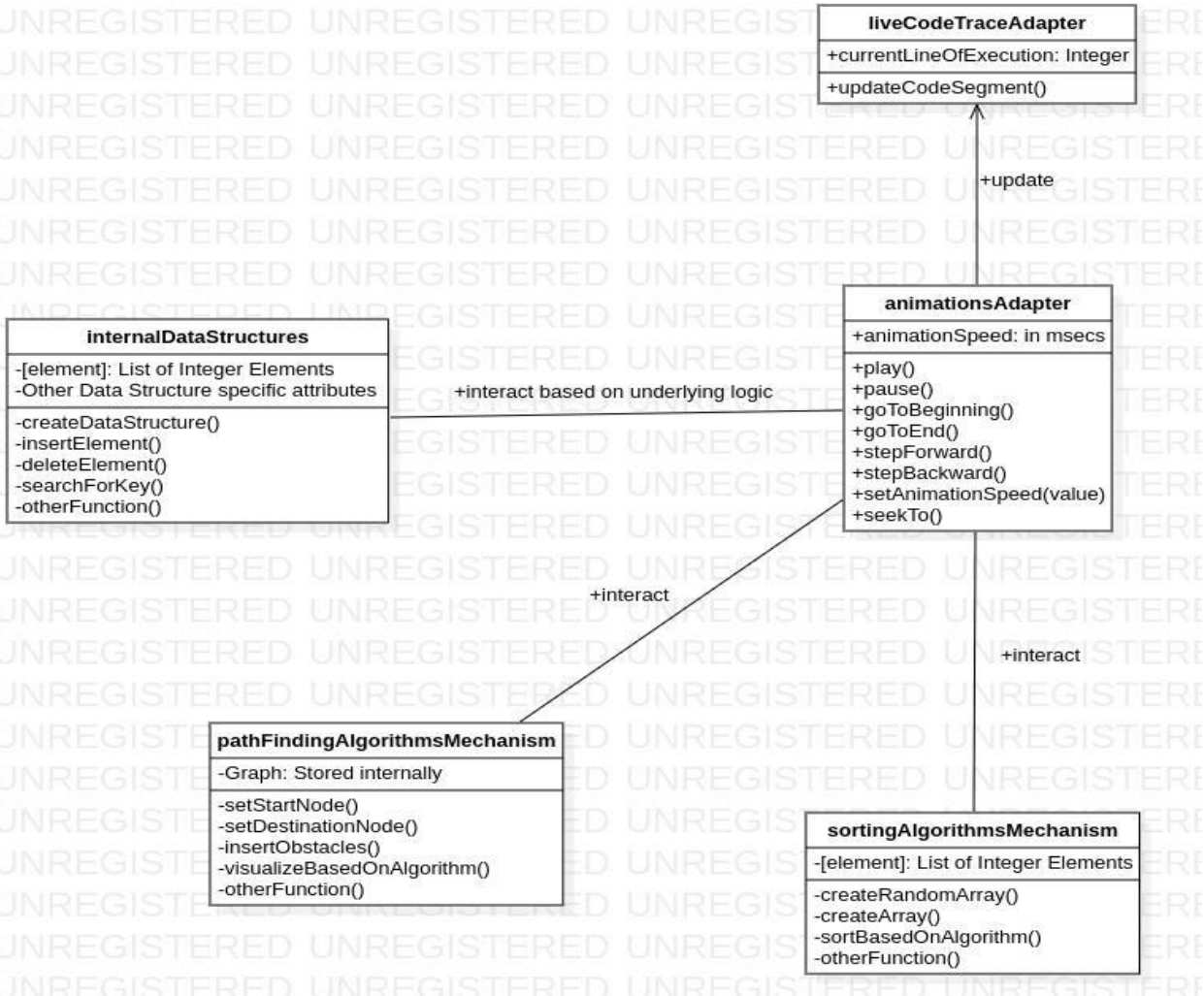
7. **Functions to render the visualizations on the canvas**
    - These are modularised and refactored according to what is necessary for each of the sections specified in the landing page.
    - So, for instance we have one interface that renders the visualizations for sorting algorithms, one for linear data structures, another for Trees and yet another for the graphs (the most complex one) and the last one for string matching which is quite similar to the one used for linear data structures but is much less complex.

8. **Live Code Trace Adapter**

    This adapter is wholly responsible for controlling the live code tracing features throughout the visualizer. However, this has been integrated only to the Sorting Algorithm visualizer. Given the time constraints we try to focus on the actual operations, animation controls, and usefulness of the system.

3.1. Master Class Diagram

This is a very abstract representation that helps understand the workflow of the entire system in a simpler and meaningful way.



Module 1: Internal Data Structures (Storage)

Essentially contains all of the data structures stored internally to efficiently compute the various outputs of be it the path-finding algorithms, sorting algorithms, or the data structure operations.

Module 2: Data Structure Operations Component

This component ensures seamless visualizations or simple data structure operations such as creating the data structure, inserting elements into the data structure, deleting an element, searching for a key in the data structure, etc. Internally uses a graph as it encompasses only the kind of data structures that we are implementing.

Methods:

- createDataStructure()
- insertElement()
- deleteElement()
- linearSearch()
- Other helper functions such as animation controller

Module 3: Sorting Algorithms Component

This component is entirely responsible for taking in the values (via the internal data structures) generated in an array (either used defined or randomly generated) to efficiently compute the visualizations for various sorting algorithms namely, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, and Selection Sort. The internal data structure used in this case is a linear array to make the visualization crystal clear.

Methods:

- generateRandomArray()
- generateUserDefinedArray()
- visualizeBasedOnAlgorithm()
- Other helper functions such as animation controllers.'

Module 4: Pathfinding Algorithms Component

This component essentially contains all of the necessary functionalities to execute the desired visualizations using the data passed from the internal data structures, and the animations controller into the view thereby rendering the visualization. The internal data structure used in this case is a graph.

Methods:

- setStartNode()
- setDestinationNode()
- visualizeBasedOnAlgorithm()
- Other helper functions such as animation controllers.

Module 5: Animations Adapter

This adapter is wholly responsible for controlling the animations, be it playing, pausing, manipulating the animation speed, etc. Has a neat interface between each of the components mentioned above.

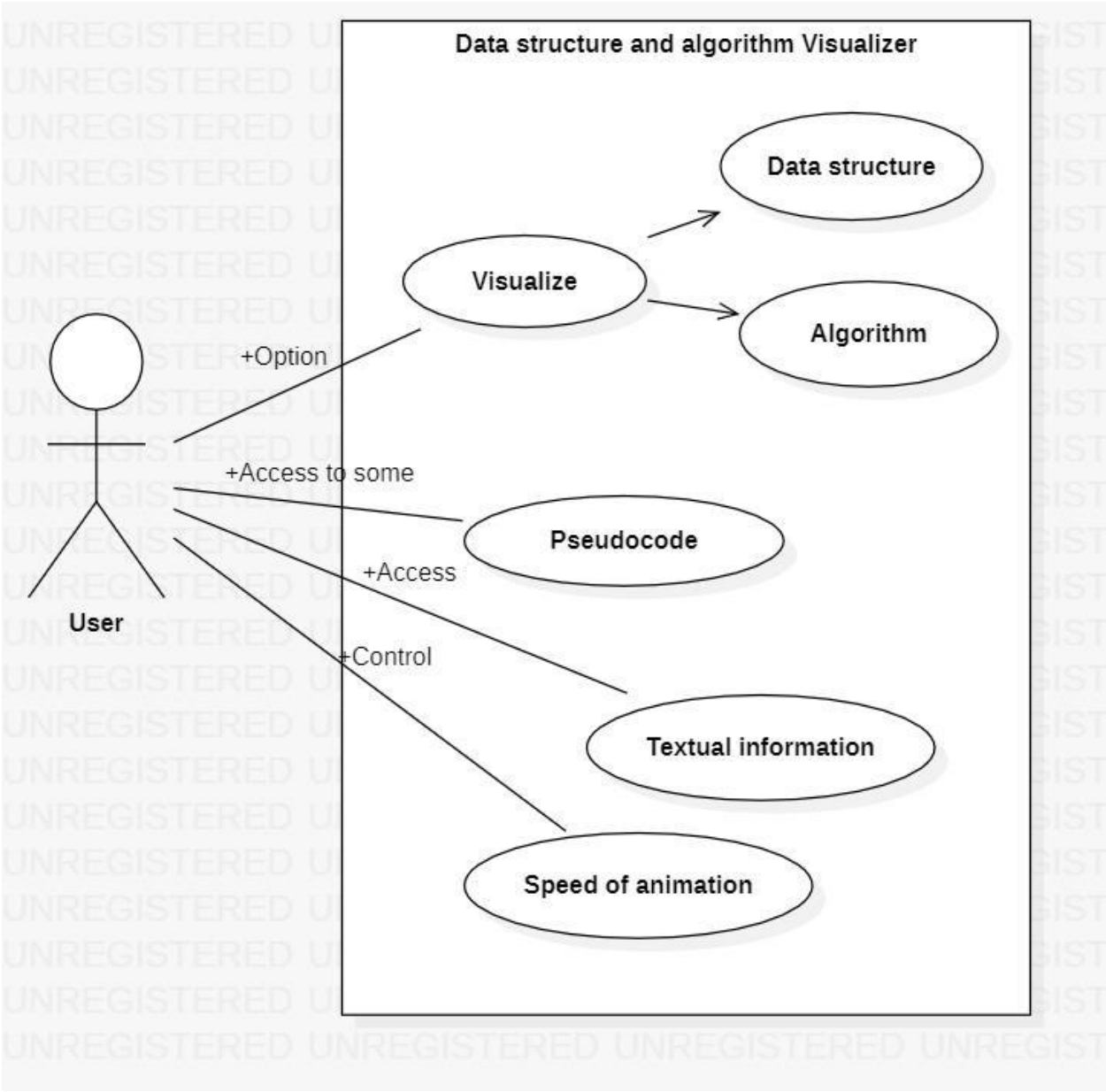
Modules:

- play()
- stop()
- setAnimationSpeed()
- Other helper functions that establish an interface between the above mentioned components and itself.

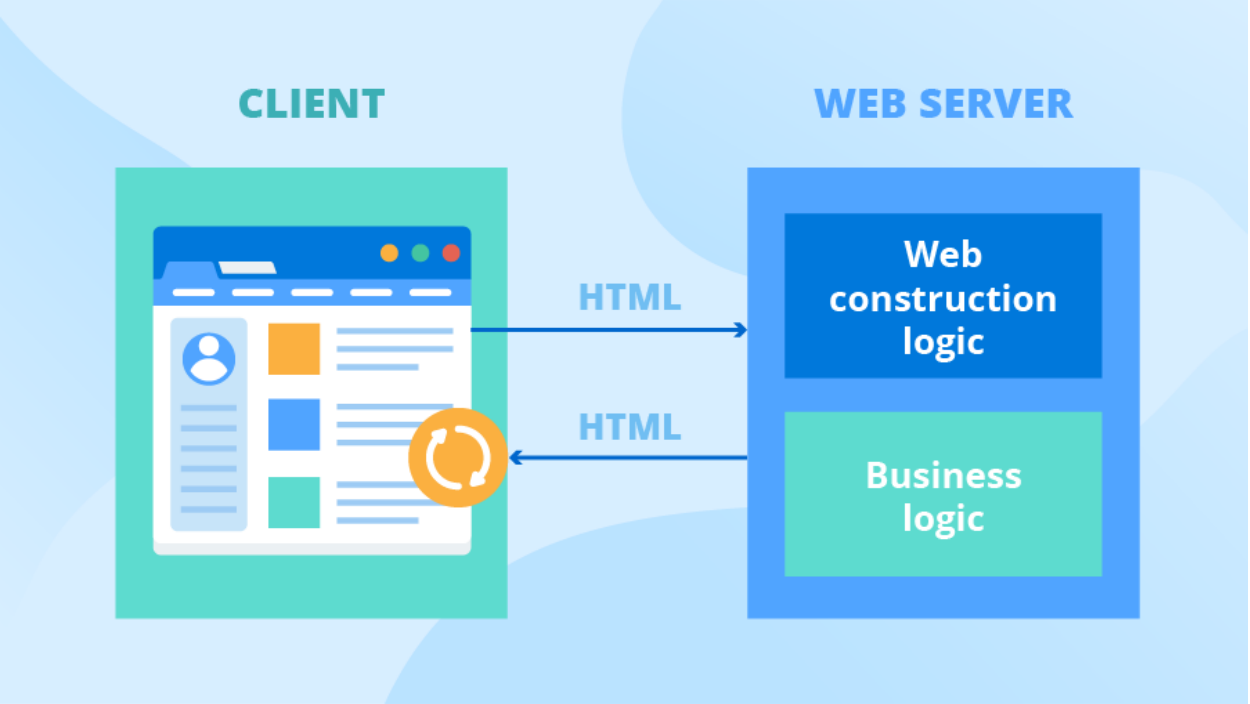
Module 6: Live Code Trace Adapter

This adapter is wholly responsible for controlling the live code tracing features throughout the visualizer. However, this has been integrated only to the Sorting Algorithm visualizer. Given the time constraints we try to focus on the actual operations, animation controls, and usefulness of the system.

3.2.3. Use Case Diagram



3.2.4. Packaging and Deployment Diagram



Since the system does not require a backend or a server handling requests, and thereby does all of it using a single client application on the front-end, the deployment simply involves this web client.

4. Proposed Methodology / Approach

1. Addition of other data structure algorithms would require multiple changes across multiple functions.
    a. Reason: The project only covers the most applicable and widely used algorithms simply because other algorithms are purely theoretical and have special, constrained use-cases. The requirement for extensibility was not prevalent and given the time constraints, a procedural approach was decided to be the simplest, cleanest and fastest approach.

2. No dependence on external JS libraries
    a. Reason: Since the graph visualization component is ultimately to be merged into a larger component, the project is implemented in plain

JS. The dependencies that come with using multiple libraries and the integration tests that follow are not feasible or required in the case of the larger integrated project.

4.1 Algorithm and Pseudocode

Starting off with the component to visualize the data structure operations such as pushing elements into a stack, popping off elements from the stack, equeuing, dequeuing, etc. To understand what exactly happens behind the scenes.

- The first functionality offered is that the system takes the values to be inserted/created or deleted from the data structure concerned using form fields. Traditional HTML input tags are used for this purpose.
- Now that we have that value, we further pass it onto the Animation Interfaces as required via a button click.
- Within this Animation Interface we have got essentially two pieces: one which triggers a block of animation at subsequent intervals and another one which analyzes every animation piece depending upon the triggered operation's click event listener.

```
this.startNewAnimation = function (commands) {
    clearTimeout(timer);
    this.animationSteps = commands;
    this.startNextBlock();
    this.currentFrame = 0;
    timer = setTimeout("timeout()", timespan);
  };
```

The above segment calls startNextBlock() which is essentially responsible for executing the concerned animation step and this happens at subsequent intervals because of the setTimeout function.

- Coming to the startNextBlock() function's implementation, this is precisely what calls appropriate functions depending upon which of the click event listeners are triggered, that is create a circle in the canvas / delete a circle from the canvas, etc.

```
if (nextCommand[0].toUpperCase() == "CREATECIRCLE") {
        this.objectManager.addCircleObject(
          parseInt(nextCommand[1]),
```

```
          nextCommand[2],
          nextCommand[5]
        );
        this.objectManager.setPosition(
          parseInt(nextCommand[1]),
          parseInt(nextCommand[3]),
          parseInt(nextCommand[4])
        );
      }
```

The above code segment is what is called when the user wants to create a circle in the canvas. The same is done for every thing that involves rendering an animated object onto the canvas.

- Coming to the final piece that is responsible for calling each of the functions required for the visualizations, are the <Algorithm>.js files which essentially calls all of the animation helper functions discussed previously. Let's take an example for how the Djikstra's algorithm's visualization is implemented.

```
Graph.prototype.Dijkstra = function (startVertex) {
  // Cancel the high bright top
  for (var i = 0; i < this.vertexNum; i++) {
    this.cmd("SetForegroundColor", i, this.foregroundColor);
  }
  // Assume that it is a direction
  this.INF = 10000;
  // Source pointiShortest distance
  var dist = new Array(this.vertexNum);
  // Have you found it?i Shortest path
  var found = new Array(this.vertexNum);
  // Record path
  var path = new Array(this.vertexNum);
  // Record full path
  var fullPath = new Array(this.vertexNum);
```

```
  for (var i = 0; i < this.vertexNum; i++) {
    fullPath[i] = new Array();
  }
  // Some initialization
  for (var i = 0; i < this.vertexNum; i++) {
    if (this.matrix[startVertex][i]) {
      dist[i] = this.matrix[startVertex][i];
    } else {
      dist[i] = this.INF;
    }
    found[i] = -1; // -1Did not find
    path[i] = startVertex; // viastartVertexarrive
  }
  dist[startVertex] = 0;
  found[startVertex] = 1;
              this.cmd("SetForegroundColor",       startVertex,
this.highlightColor);
  fullPath[startVertex].push(startVertex);
  //path[startVertex] = startVertex;
  // hintShow current content
  // vertex
  this.cmd(
    "CreateRectangle",
    this.hintVertexID,
    "Vertex",
    this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
    "top",
    this.hintStartX,
```

```
    this.hintStartY - this.hintObjectHeight
  );
          this.cmd("SetForegroundColor",     this.hintVertexID,
this.foregroundColor);
          this.cmd("SetBackgroundColor",     this.hintVertexID,
this.backgroundColor);
  // Known
  this.cmd(
    "CreateRectangle",
    this.hintKnownID,
    "Known",
    this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
    "top",
    this.hintStartX + this.hintObjectWidth,
    this.hintStartY - this.hintObjectHeight
  );
           this.cmd("SetForegroundColor",     this.hintKnownID,
this.foregroundColor);
          this.cmd("SetBackgroundColor",     this.hintKnownID,
this.backgroundColor);
  // Cost
  this.cmd(
    "CreateRectangle",
    this.hintCostID,
    "Cost",
    this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
```

```
    "top",
    this.hintStartX + 2 * this.hintObjectWidth,
    this.hintStartY - this.hintObjectHeight
  );
            this.cmd("SetForegroundColor",      this.hintCostID,
this.foregroundColor);
            this.cmd("SetBackgroundColor",      this.hintCostID,
this.backgroundColor);
  // Path
  this.cmd(
    "CreateRectangle",
    this.hintPathID,
    "Path",
    2 * this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
    "top",
    this.hintStartX + 3 * this.hintObjectWidth,
    this.hintStartY - this.hintObjectHeight
  );
            this.cmd("SetForegroundColor",      this.hintPathID,
this.foregroundColor);
            this.cmd("SetBackgroundColor",      this.hintPathID,
this.backgroundColor);
  // vertex
  for (var i = 0; i < this.vertexNum; i++) {
    this.cmd(
      "CreateRectangle",
      this.hintVertexColumnID[i],
      i,
```

```
      this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX,
      this.hintStartY + i * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.hintVertexColumnID[i],
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.hintVertexColumnID[i],
      this.backgroundColor
    );
  }
  // known
  for (var i = 0; i < this.vertexNum; i++) {
    var label = found[i] == 1 ? "True" : "False";
    this.cmd(
      "CreateRectangle",
      this.hintKnownColumnID[i],
      label,
      this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
```

```
      this.hintStartX + this.hintObjectWidth,
      this.hintStartY + i * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.hintKnownColumnID[i],
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.hintKnownColumnID[i],
      this.backgroundColor
    );
  }
  // cost
  for (var i = 0; i < this.vertexNum; i++) {
    var label = dist[i] == this.INF ? "INF" : dist[i];
    this.cmd(
      "CreateRectangle",
      this.hintCostColumnID[i],
      label,
      this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX + 2 * this.hintObjectWidth,
      this.hintStartY + i * this.hintObjectHeight
    );
    this.cmd(
```

```
      "SetForegroundColor",
      this.hintCostColumnID[i],
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.hintCostColumnID[i],
      this.backgroundColor
    );
  }
  // Path
  for (var i = 0; i < this.vertexNum; i++) {
    var label = "";
    this.cmd(
      "CreateRectangle",
      this.hintPathColumnID[i],
      label,
      2 * this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX + 3 * this.hintObjectWidth,
      this.hintStartY + i * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.hintPathColumnID[i],
      this.foregroundColor
    );
```

```
    this.cmd(
      "SetBackgroundColor",
      this.hintPathColumnID[i],
      this.backgroundColor
    );
  }

  this.cmd(
    "SetState",
    "Top of the starting point " + startVertex + " Set to your
own expense0"
  );
  this.cmd("SetHighlight", startVertex, true);
  this.cmd("Step");
  this.cmd("SetHighlight", startVertex, false);
  this.cmd("Step");

  // Add a path to the starting point
  this.cmd(
    "CreateHighlightRectangle",
    this.highlightRectangleKnownID,
    this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
    "top",
    this.hintStartX + this.hintObjectWidth,
    this.hintStartY + startVertex * this.hintObjectHeight
  );
  this.cmd(
    "SetForegroundColor",
```

```
    this.highlightRectangleKnownID,
    this.foregroundColor
  );
  this.cmd(
    "SetBackgroundColor",
    this.highlightRectangleKnownID,
    this.backgroundColor
  );
  this.cmd("Step");
  this.cmd(
    "CreateHighlightRectangle",
    this.highlightRectanglePathID,
    2 * this.hintObjectWidth,
    this.hintObjectHeight,
    "left",
    "top",
    this.hintStartX + 3 * this.hintObjectWidth,
    this.hintStartY + startVertex * this.hintObjectHeight
  );
  this.cmd(
    "SetForegroundColor",
    this.highlightRectanglePathID,
    this.foregroundColor
  );
  this.cmd(
    "SetBackgroundColor",
    this.highlightRectanglePathID,
    this.backgroundColor
  );
```

```
  this.cmd("Step");
  this.cmd(
    "SetLabel",
    this.hintPathColumnID[startVertex],
    fullPath[startVertex][0]
  );
  this.cmd("Step");
  this.cmd("Delete", this.highlightRectanglePathID);
  this.cmd("Delete", this.highlightRectangleKnownID);

  for (var i = 1; i < this.vertexNum; i++) {
    var min = this.INF;
    var vertex = -1;
    for (var j = 0; j < dist.length; j++) {
      if (found[j] == -1 && dist[j] < min) {
        min = dist[j];
        vertex = j;
      }
    }
    // Can't find a valid and minimal side
    if (vertex == -1) {
      // Middle finger
      break;
    }
    // Found a suitable side
    found[vertex] = 1;
    this.cmd(
      "SetState",
       "Find to the vertices " + vertex + " Shortest path,Cost
" + min
```

```
    );
    // Depend onpathEstablish a complete path
    fullPath[vertex].push(vertex);
      for (var i = fullPath[path[vertex]].length - 1; i >= 0;
i--) {
      fullPath[vertex].unshift(fullPath[path[vertex]][i]);
    }
    // Highlight smallestcost
    this.cmd(
      "CreateHighlightRectangle",
      this.highlightRectangleCostID,
      this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX + 2 * this.hintObjectWidth,
      this.hintStartY + vertex * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.highlightRectangleCostID,
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.highlightRectangleCostID,
      this.backgroundColor
    );
    this.cmd("Step");
```

```
                    this.cmd("SetForegroundColor",      vertex,
this.highlightColor);
    // willvertexofKnownRevise
    this.cmd(
      "CreateHighlightRectangle",
      this.highlightRectangleKnownID,
      this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX + this.hintObjectWidth,
      this.hintStartY + vertex * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.highlightRectangleKnownID,
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.highlightRectangleKnownID,
      this.backgroundColor
    );
          this.cmd("SetLabel",   this.hintKnownColumnID[vertex],
"True");
                        //      this.cmd("SetForegroundColor",
this.hintKnownColumnID[vertex], this.highlightColor);
    this.cmd("Step");
    // HighlightPathAnd update
    this.cmd(
```

```
      "CreateHighlightRectangle",
      this.highlightRectanglePathID,
      2 * this.hintObjectWidth,
      this.hintObjectHeight,
      "left",
      "top",
      this.hintStartX + 3 * this.hintObjectWidth,
      this.hintStartY + vertex * this.hintObjectHeight
    );
    this.cmd(
      "SetForegroundColor",
      this.highlightRectanglePathID,
      this.foregroundColor
    );
    this.cmd(
      "SetBackgroundColor",
      this.highlightRectanglePathID,
      this.backgroundColor
    );
    this.cmd("Step");
    var labelPath = "";
    for (var i = 0; i < fullPath[vertex].length; i++) {
      labelPath = labelPath + fullPath[vertex][i] + " ";
    }
          this.cmd("SetLabel",   this.hintPathColumnID[vertex],
labelPath);
    this.cmd("Step");
    // Delete two high debut
    // this.cmd("Delete", this.highlightRectangleKnownID);
    this.cmd("Delete", this.highlightRectangleCostID);
```

```
    this.cmd("Delete", this.highlightRectanglePathID);
    this.cmd("Step");

    this.cmd("SetHighlight", vertex, true);
    this.cmd("Step");
    this.cmd("SetHighlight", vertex, false);
    this.cmd("Step");
    for (
      var edge = this.firstEdge(vertex);
      edge != null;
      edge = this.nextEdge(edge)
    ) {
      this.cmd("SetHighlight", edge.startVertex, true);
      this.cmd("Step");
      this.cmd("SetHighlight", edge.startVertex, false);
      this.cmd("Step");
      var lineSt = edge.startVertex;
      var lineEn = edge.endVertex;
      if (!this.directed && lineSt > lineEn) {
        var tmp = lineEn;
        lineEn = lineSt;
        lineSt = tmp;
      }
      this.cmd("SetLineHighlight", lineSt, lineEn, true);
      this.cmd("Step");
      this.cmd("SetLineHighlight", lineSt, lineEn, false);
      this.cmd("Step");
      if (
        found[edge.endVertex] == -1 &&
```

```
        dist[vertex] + edge.weight < dist[edge.endVertex]
      ) {
        this.cmd(
          "SetState",
          "Update from the vertex" +
            startVertex +
            " arrive " +
            edge.endVertex +
            " The shortest path is" +
            startVertex +
            ".->" +
            vertex +
            "->" +
            edge.endVertex
        );
        this.cmd("Step");
        dist[edge.endVertex] = dist[vertex] + edge.weight;
        // Highlightdistrenew
                        //   this.cmd("CreateHighlightRectangle",
this.highlightRectangleCostID,
          //      this.hintObjectWidth,  this.hintObjectHeight,
'left', 'top',
           //       this.hintStartX  +  2*  this.hintObjectWidth,
this.hintStartY + edge.endVertex * this.hintObjectHeight);
                            //    this.cmd("SetForegroundColor",
this.highlightRectangleCostID, this.foregroundColor);
                            //    this.cmd("SetBackgroundColor",
this.highlightRectangleCostID, this.backgroundColor);
        // this.cmd("Step");
```

```
                                    //     this.cmd("SetLabel",
this.hintCostColumnID[edge.endVertex], dist[edge.endVertex]);
        // this.cmd("Step");
        // this.cmd("Delete", this.highlightRectangleCostID);

        path[edge.endVertex] = vertex;
      } else {
        this.cmd(
          "SetState",
          "vertex" +
            startVertex +
            " arrive " +
            edge.endVertex +
            " The path is not updated"
        );
        this.cmd("Step");
      }
          // After finding the vertices of a shortest path,
updatecost
      this.cmd(
        "CreateHighlightRectangle",
        this.highlightRectangleCostID,
        this.hintObjectWidth,
        this.hintObjectHeight,
        "left",
        "top",
        this.hintStartX + 2 * this.hintObjectWidth,
                        this.hintStartY   +   edge.endVertex   *
this.hintObjectHeight
      );
```

```
      this.cmd(
        "SetForegroundColor",
        this.highlightRectangleCostID,
        this.foregroundColor
      );
      this.cmd(
        "SetBackgroundColor",
        this.highlightRectangleCostID,
        this.backgroundColor
      );
      this.cmd("Step");
      this.cmd(
        "SetLabel",
        this.hintCostColumnID[edge.endVertex],
        dist[edge.endVertex]
      );
      this.cmd("Step");
      this.cmd("Delete", this.highlightRectangleCostID);
    }
    // Update completed, deleteknown high light circle ID
    this.cmd("Delete", this.highlightRectangleKnownID);
    this.cmd("Step");
  }


   // Processing the remaining vertices of the path, the path
is set toNO PATH
  for (var i = 0; i < this.vertexNum; i++) {
    if (found[i] == -1 && dist[i] == this.INF) {
       // hintRegionally highlight this vertex, then modify the
pathlabel
```

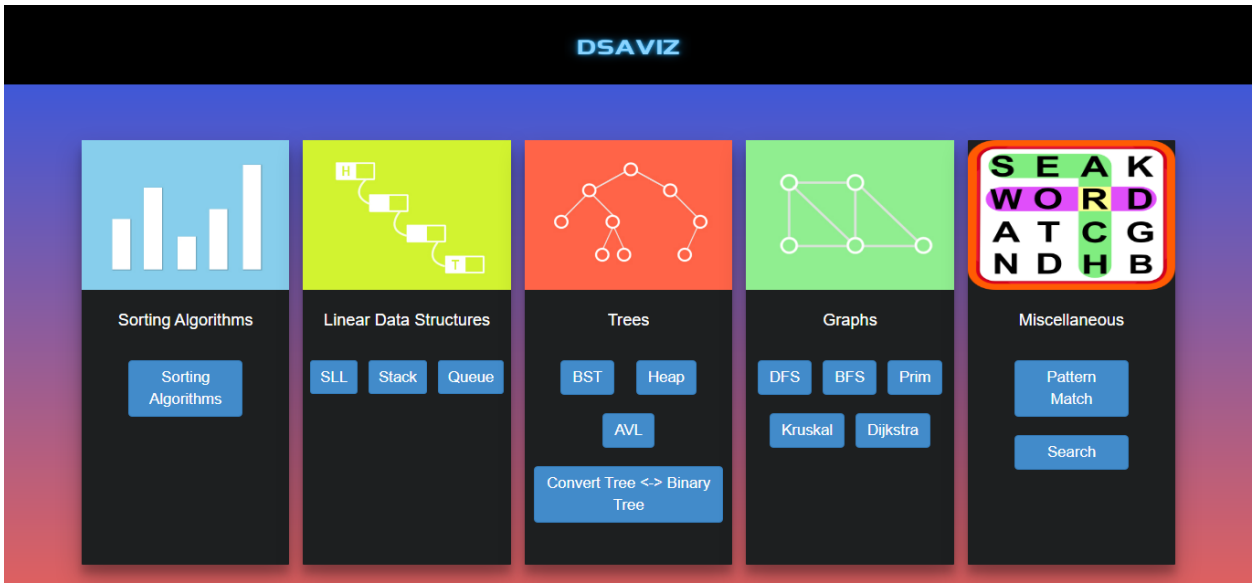
```
      this.cmd(
        "SetState",
        "Did not find it " +
          startVertex +
          " arrive " +
          i +
          " Path, its path is setNO PATH"
      );
      this.cmd("Step");
      this.cmd(
        "CreateHighlightRectangle",
        this.highlightRectangleKnownID,
        this.hintObjectWidth,
        this.hintObjectHeight,
        "left",
        "top",
        this.hintStartX + this.hintObjectWidth,
        this.hintStartY + i * this.hintObjectHeight
      );
      this.cmd(
        "SetForegroundColor",
        this.highlightRectangleKnownID,
        this.foregroundColor
      );
      this.cmd(
        "SetBackgroundColor",
        this.highlightRectangleKnownID,
        this.backgroundColor
      );
```

```
      this.cmd(
        "CreateHighlightRectangle",
        this.highlightRectangleCostID,
        this.hintObjectWidth,
        this.hintObjectHeight,
        "left",
        "top",
        this.hintStartX + 2 * this.hintObjectWidth,
        this.hintStartY + i * this.hintObjectHeight
      );
      this.cmd(
        "SetForegroundColor",
        this.highlightRectangleCostID,
        this.foregroundColor
      );
      this.cmd(
        "SetBackgroundColor",
        this.highlightRectangleCostID,
        this.backgroundColor
      );
      this.cmd(
        "CreateHighlightRectangle",
        this.highlightRectanglePathID,
        2 * this.hintObjectWidth,
        this.hintObjectHeight,
        "left",
        "top",
        this.hintStartX + 3 * this.hintObjectWidth,
        this.hintObjectHeight + i * this.hintObjectHeight
```

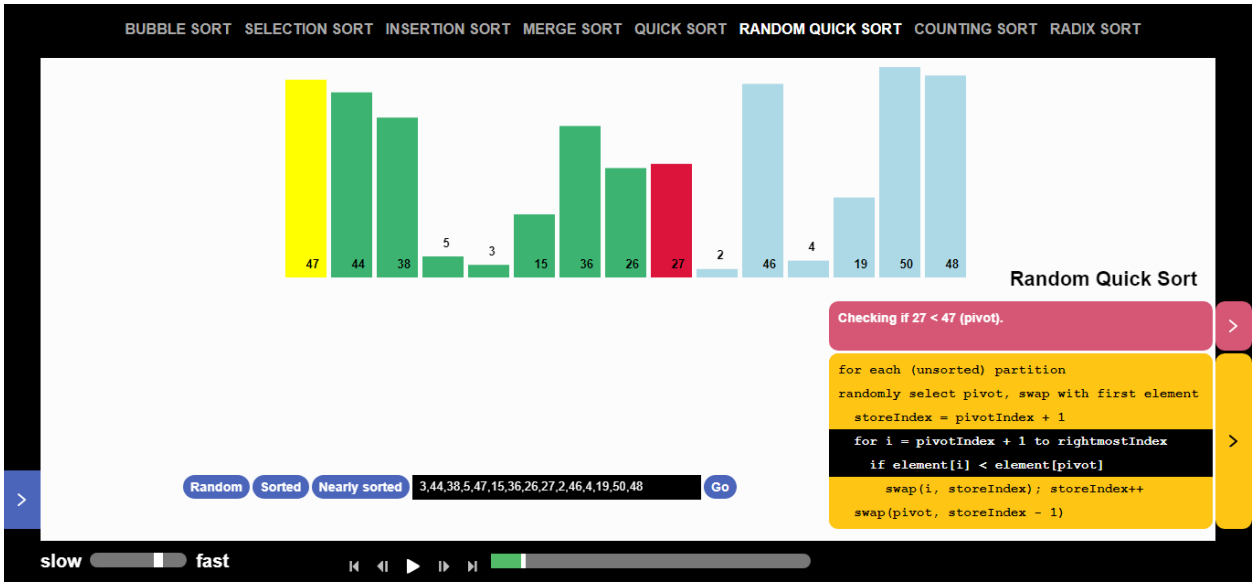
```
      );
      this.cmd(
        "SetForegroundColor",
        this.highlightRectanglePathID,
        this.foregroundColor
      );
      this.cmd(
        "SetBackgroundColor",
        this.highlightRectanglePathID,
        this.backgroundColor
      );
      this.cmd("Step");
           this.cmd("SetLabel",  this.hintPathColumnID[i],  "NO
PATH");
      this.cmd("Setp");
      this.cmd("Delete", this.highlightRectanglePathID);
      this.cmd("Delete", this.highlightRectangleCostID);
      this.cmd("Delete", this.highlightRectangleKnownID);
      this.cmd("Step");
    }
  }
  this.cmd(
    "SetState",
      "The algorithm is executed, and the results are shown
below"
  );
  return this.commands;
};
```

4.2 Implementation and Results

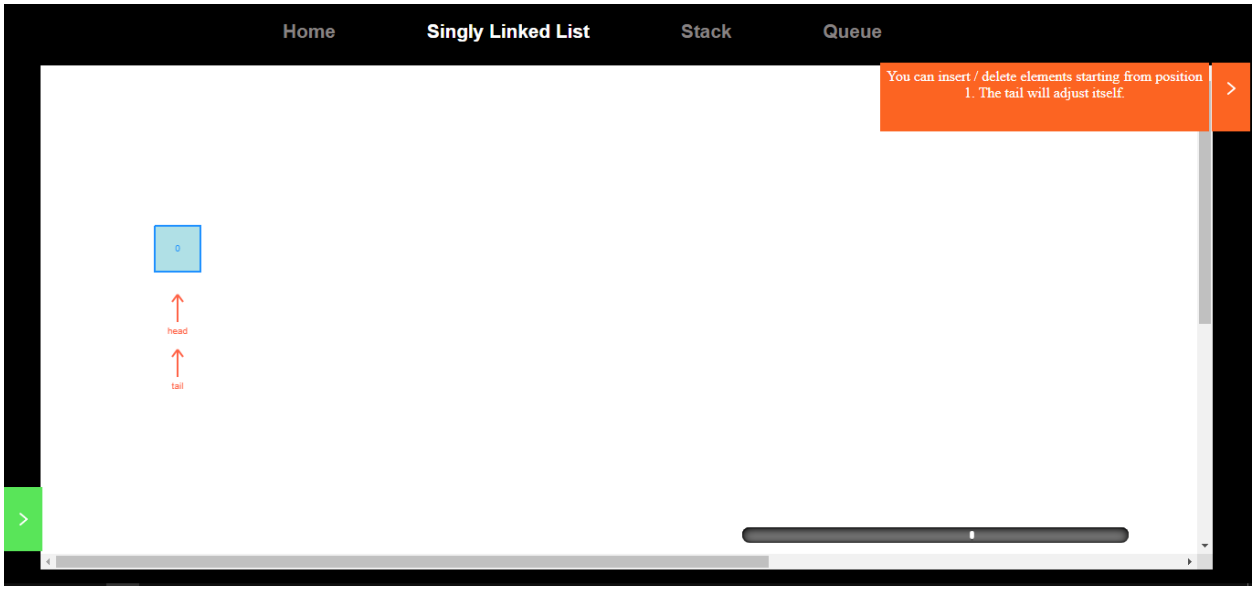
Landing page:



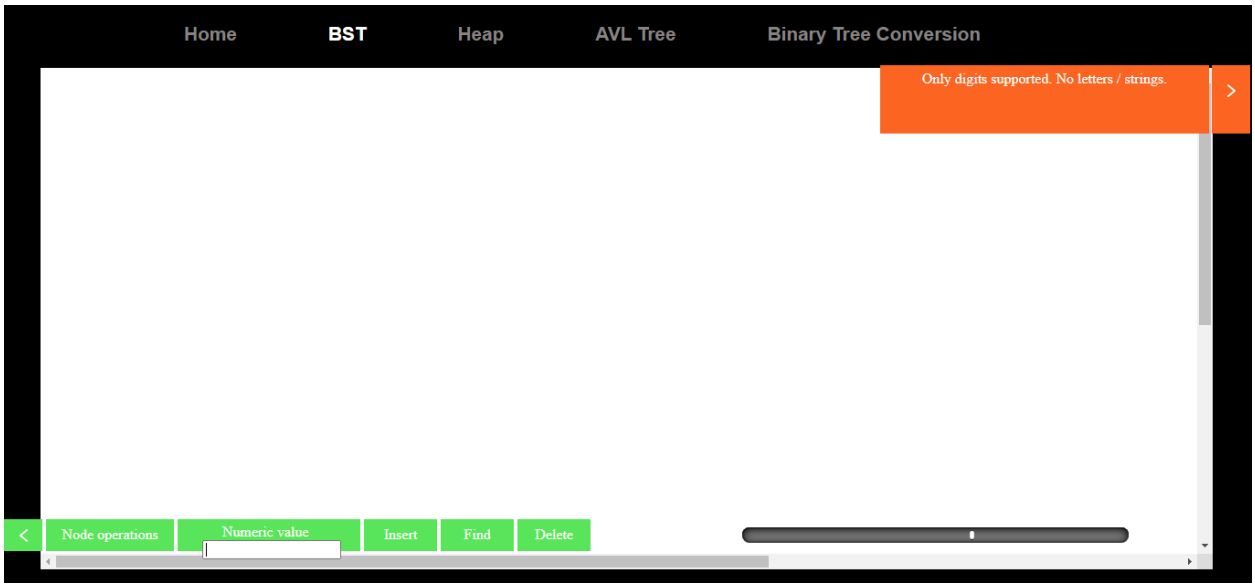
Sorting algorithm visualizer:



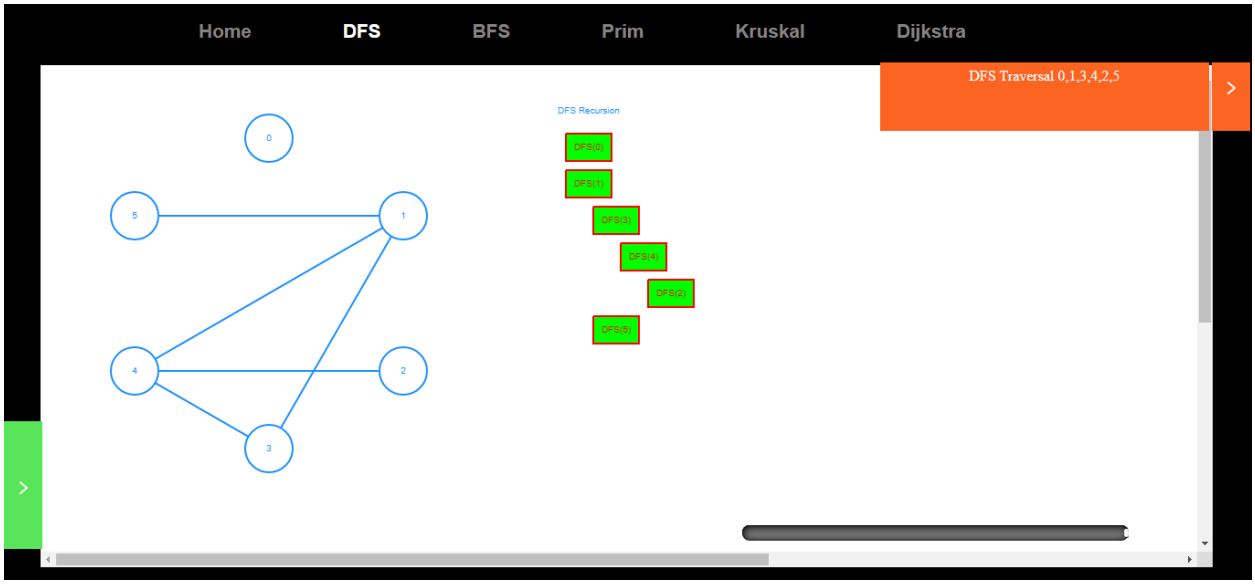
Linear data structures visualizer:



Trees visualizer:



Graphs visualizer:



Miscellaneous operations visualizer:

Visualizers for the following were implemented:

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

Appendix A: Definitions, Acronyms and Abbreviations

- **JS**: JavaScript (Language used to implement all the algorithms, add interactivity to HTML elements)

- **CSS**: Cascading style sheets (Styling language that makes use of rules that apply to HTML elements to edit how they look on the webpage)

Appendix B: References

- HLD Document from Phase 1 of the Capstone Project
- Project Report from Phase 1 of the Capstone Project
- Building a sorting algorithm visualizer in python: Link
- 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.

Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	**01/10/2021**	**1.0**	Phase 2 Review 2	Initial content
2.	**12/12/2021**	**2.0**	Phase 2 Review Final	Modify with all the latest implementation changes.

Appendix D: Traceability Matrix

Project Requirement Specification Reference Section No. and Name.	DESIGN / HLD Reference Section No. and Name.	LLD Reference Section No. Name
Project HLD - User interface diagrams and external interfaces	Reference Section 5	4.1 - Implementation and results