A Project Report

on

ALGORITHM VISUALIZER: MITIGATE ALGORITHM COMPLEXITIES

Submitted in partial fulfillment of the
requirement for the award of the degree of

Bachelor of Technology in Computer Science and Engineering

Under The Supervision of
Ms. AANCHAL VIJ:
ASSISTANT PROFESSOR

Submitted By

ADITYA BHARDWAJ – 19SCSE1180007
SHIVAM GOEL – 19SCSE1180006

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA
INDIA
DECEMBER, 2021

SCHOOL OF COMPUTING SCIENCE AND ENGINEERING
GALGOTIAS UNIVERSITY, GREATER NOIDA

CANDIDATE'S DECLARATION

I/We hereby certify that the work which is being presented in the project, entitled **"ALGORITHM VISUALIZER: Mitigate Algorithm Complexities"** in partial fulfillment of the requirements for the award of the **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING** submitted in the **School of Computing Science and Engineering** of Galgotias University, Greater Noida, is an original work carried out during the period of **JULY, 2021 TO DECEMBER, 2021**, under the supervision of **MS. AANCHAL VIJ, Assistant Professor, Department of Computer Science and Engineering** of School of Computing Science and Engineering , Galgotias University, Greater Noida

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

Aditya Bhardwaj - 19SCSE1180007

Shivam Goel - 19SCSE1180006

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

Ms. Aanchal Vij

Assistant Professor

CERTIFICATE

The Final Thesis/Project/ Dissertation Viva-Voce examination of **ADITYA BHARDWAJ: 19SCSE1180007 AND SHIVAM GOEL: 19SCSE1180006** has been held on _____ and his/her work is recommended for the award of **BACHELOR OF TECHNOLOGY IN COMPUTER SCIENCE AND ENGINEERING**.

Signature of Examiner(s) **Signature of Supervisor(s)**

Signature of Project Coordinator **Signature of Dean**

Date: December, 2021

Place: Greater Noida

Abstract

In this project we have made an Algorithm Visualizer. In that we have implemented a path finding visualizer and sorting visualizer as well. By this project we not only aim to demonstrate the uses of these various algorithms but also want to create a tool to be used for better understanding of these concepts. Algorithms like Dijkstra's , A* , Depth First Search (DFS), Breadth First Search (BFS) , Quick Sort, Heap Sort, Selection Sort, Bubble Sort, etc. are some of the most basic algorithms often used as building blocks for understanding design and analysis of various algorithms. These algorithms are also present in software all around us like google maps, internet searches, etc. In the past, algorithms have been taught through exhaustive chalkboard drawings and pseudo-code guidelines. Visualization tools are therefore an attractive learning essential for instructors and students alike . Hence here we have tried to create a tool using HTML, CSS, ExpressJS, etc. to visually depict the working of these algorithms. Our goal is to simplify the algorithm into a series of visual steps that can be understood by elementary computer science students.

Table of Contents

List of Figures

Acronyms

AWS	Amazon Web Services
AV	Algorithm Visualizer
Js	JAVA Script
DFS	Depth First Search
BFS	Breadth First Search

CHAPTER-1

Introduction

Sorting and path finding are some of the popular concepts and algorithms that any person working in software industry learns initially. This is because not only these have extensive applications in other algorithms but also because we can see this algorithm in work around us in common applications.

These reasons make these algorithms of extreme importance which made us decide to work on them and create a better medium to understand them. We tried to use various algorithms in our visualizer to depict the processes as accurately as possible.

The algorithms being used are –

- Selections Sort
- Bubble Sort
- Insertions Sort
- Quick Sort
- Merge Sort
- Heap Sort
- A* Algorithm
- Depth First Search (DFS)
- Breadth First Search (BFS)
- Dijkstra’s Algorithm

Based on a comparison operator on the elements, a Sorting Algorithm rearranges the items of an array or list. In this project we have randomly generated arrays with elements varying in magnitude. We have implemented the algorithms in such a way that not only do we get a visual of the finally sorted array but also get the visualization of the sorting process as it takes place in front of our eyes. This becomes a very great way to see all these algorithms in play. We have used colored animations to depict changes so that they are clearly visible.

Path Finding algorithms are a type of algorithm that determines the shortest path between two nodes on a graph. Here we use them to find the path between two points. We assume the entire grid as the graph and apply various graph algorithms on it. We can also place walls on the graph which are basically nothing just breaking the connections between two grid elements.

We can also attach weights to each grid element which could be used to visualize the working of path finding algorithms in weighted graphs. Also, we have used color scheme to highlight the various phases of recursion.
We have also made the speed of all the above processes alterable so any user can understand at his/her pace. Also, we have added many useful features like resetting the board, reloading the site or choosing new algorithms on the spot and picking up from where you left off without reloading the site. We have also added for the user to easily access the site by making it very responsive.

All of the above algorithms and their respective visualizations have been implemented using pure JavaScript and no external libraries. We have also added styling using CSS to make the site more appealing and easier to use for the user. A good knowledge of all these concepts is displayed throughout. While selecting the above algorithms we kept in mind to select a diversity of them representing various use cases and various time and space complexities ranging from $O(NlogN)$ to $O(N^2)$.

We chose algorithms like Breadth First Search (BFS) to show the shortest path edgewise or in an unweighted and graph and to show level wise traversal, we kept Depth First Search (DFS) to show how it is not a good path finding algorithm because of its approach of traversing each individual branch one after another, Dijkstra's to have an efficient path finding algorithm in weighted graphs/layouts.

We chose basic high time complexity sorting like bubble sort, selection sort and insertion sort to explain sorting and help people experience firsthand the reason why they are slow as they have to traverse twice through the array. Then we have also added optimized sorting algorithms like merge sort, quick sort and heap sort to show their edge over the other three algorithms.

All these algorithms are quite interesting and can be seen in real world in applications like google maps, Computational services like AWS and much more. The most common use case could be Google Maps finding us the shortest route to our work through the traffic and minimizing time. So, by visualizing and trying out various path finding algorithms we can easily identify that google treats its maps as a weighted graph with roads with higher traffic having higher weights and then it implements similar path finding algorithms to find you the best path so that you reach on time.

Formulation Of Problem

We specifically chose to work on this project because we experienced this problem first hand. During the first year of our college life when both us were learning these algorithms we faced numerous difficulties due to the extremely theoretical nature of all the leaning courses and we were never able to visualize these algorithms and there working.
Even while going through the time and space complexities analysis of all these algorithms it wasn't easy to visually understand reason for their logarithmic or second order complexities.

So, when we were presented with the opportunity of doing a project in algorithms this semester we just jumped upon this idea as we feel the mathematics behind these algorithms should be fun as well and our idea of making them fun is through direct visualization of working of these algorithms.

Furthermore choosing these algorithms was based upon our exploring various data structures and algorithms and realizing just how important these algorithms are . Many problems directly or indirectly use these algorithms which made it clear that these things even though taught in beginning as basics are building blocks for developing an interest in algorithms design and analysis. Hence working on this project was a no brainer for us.

Through this project we aim to create not just an sorting or path finding visualizer but aim to grow it into a collection of various algorithms visualizations including advanced topics like dynamic programming, backtracking other popular graph algorithms and much more

Tool and Technology Used

We have tried to create an intuitive UI that is easy to understand and use for the user.

Technology Stack –

- HTML
- CSS
- JavaScript
- NodeJS
- ExpressJS

CHAPTER-2

Literature Survey

AVs have a long history in computer science education, dating from the 1981 video “Sorting out Sorting” by Ronald Baeker and the BALSA system [Brown and Sedgewick 1984]. Since then, hundreds of AVs have been implemented and provided free to educators, and scores of papers have been written about them [AlgoViz.org 2010].
Good AVs bring algorithms to life by graphically representing their various states and animating the transitions between those states. They illustrate data structures in natural, abstract ways instead of focusing on memory addresses and function calls.

Algorithm visualization (AV) technology graphically illustrates how algorithms work. Despite the intuitive appeal of the technology, it has failed to catch on in mainstream computer science education. Some have attributed this failure to the mixed results of experimental studies designed to substantiate AV technology’s educational effectiveness.
An important conclusion from the literature is that to make AVs pedagogically useful, they must support student interaction and active learning [Naps et al. 2002].

Certainly, many AVs exist and are widely (and freely) available via the Internet. Unfortunately, those of high quality can be lost among the many of lower quality. The most common way to make an algorithm animation is to annotate the algorithm code with scripting instructions that produce the visualisation. John Stasko and his colleagues created the first system based on a scripting language, which is part of a larger family of algorithm visualisation systems (Tango, Polka, Samba and JSamba).

Animations are made up of a file that contains graphical instructions that correlate to key events in the algorithm being shown. Another set of systems, such as MatrixPro, Trakla2, and Ville, offer "Algorithm Simulation Exercises," in which the learner must manually carry out a particular algorithm, either by dragging pieces to new or target places or by clicking on buttons to fulfil a certain function.

The creation of frameworks for the so-called domain of algorithm visualisation, which is defined as the visualisation of a high-level description of a piece of software, has seen significant progress in recent years. People who develop algorithms are well aware that their brain (or, more precisely, its right side [6]) automatically performs imaginative work, consisting of conjuring mental images that correspond to the main characteristics of the various phases of algorithm analysis, development, and implementation.

As a result, automated visualisation may be a helpful assistance for teaching as well as designing and comprehending algorithms, as discussed in [8]. (both in the design phase and in the analysis and development ones). [11] contains a survey on the more broad topic of software visualisation (in short, SV), which includes twelve visualisation solutions (some of them includes (BALSA [7], TANGO [10], ANIM [4]).

On the other hand, demonstrating how the values of programme variables change will not expose the algorithm's logic. Students will want acceptable graphical representations of the algorithm that correspond to their conceptual ideas of how it works.
Ville is a new addition to the family and supports a variety of programming languages, including C++ and Java. It has a built-in editor for creating interactive quizzes and examinations that show as pop-up windows. JHave offers a big library of algorithm visualisations and has sparked a lot of attention among educators.

Alvis Live! is a modern algorithm animation system. It is a software development environment that uses the SALSA scripting language to facilitate the creation and interactive presentation of algorithm visualizations. It has tools that help with storyboarding.

Algorithm Visualization (AV) systems try to address this demand by visualizing abstract ideas and unravelling the algorithm's underlying logic, assisting students in the development of multiple mental models, the interconnection of construct hierarchies, and the generalization of problem-solving patterns. In the literature, the phrases static and dynamic algorithm visualizations are used to distinguish between the level of involvement and students' exploration of the visualization.

Algorithm Animations for Teaching and Learning Basic Sorting Concepts which has been extensively reviewed and explained in [5]. To increase the effect of visualization, the authors strive to use Bloom's taxonomy and establish an efficient model to describe algorithms and instructional material. They employ an unusual card view technique to depict the items on which the algorithm operates during the implementation phase.

Sorting Algorithms in Pictures - The web-based platform employs sound effects as well as visual aids to improve user involvement. It tends to improve the pupils' understanding of sorting procedures. Despite the fact that arrays are an ubiquitous data structure in basic programming courses, there has been minimal study on students' mental models and programming challenges while utilizing arrays[2, 3].

According to a survey of computer academics, the arrays and loops are two of the three programming fundamentals that novice students struggle with. According to Du Boulay, students have trouble distinguishing between an array index and its cell, as well as working with arrays that contain indices as array items.

According to an unpublished study of 102 students in Greek secondary schools, the majority of pupils had inaccurate or partial models of the array notion, resulting in misunderstandings and substantial difficulty in solving simple algorithmic issues that involve the usage of array data structures (K-12).

In CS education, there are two types of visualisation systems: programme visualisation and algorithm visualisation. PV systems generate visual representations of programming structures and/or programme execution phases (e.g., values of variables, internal programme structures, method frames, data structures, objects etc.). Jype, UUhistle, and Online Python Tutor are among contemporary PV systems for visualising Python programmes; Jeliot 3 is a well-known PV system for visualising Java programmes.

The creation of frameworks for the so-called domain of algorithm visualisation, which is defined as the visualisation of a high-level description of a piece of software, has seen significant progress in recent years. People who develop algorithms are well aware that their brain (or, more precisely, its right side [6]) automatically performs imaginative work, consisting of conjuring mental images that correspond to the main characteristics of the various phases of algorithm analysis, development, and implementation.

Project Design

The major features of this project are described below. We have two main sections on our front page using which the user can choose to either go to the sorting visualizer or the path finding visualizer.

The website is divided into two sections whose features are described further-

Path Finding Visualizer

- We have created an option to easily browse through and choose among the various path finding algorithms.
- There is also the option to choose whether the layout should be weighted or unweighted so that you can see and understand the benefit of algorithms such as Dijkstra's.
- We have also created an option to alter speed to understand and see things working at your own convenience.
- Also, there are three centrally placed buttons to start the process, reset the layout and reload the site.

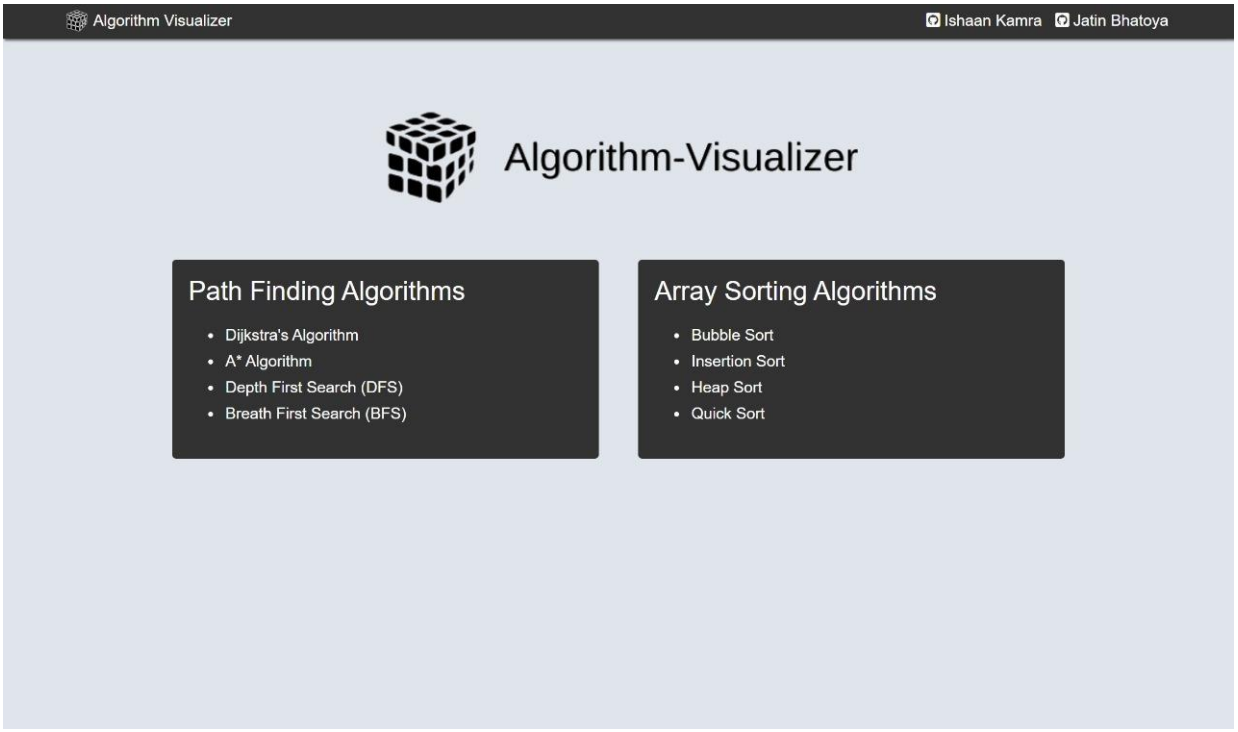
Sorting Visualizer

- There is the option to browse among and choose from various sorting algorithms.
- We have also kept the option to change the array size as per your requirement.
- We have also created an option to alter speed to understand and see things working at your own convenience.

CHAPTER – 3

Functionality/Working of Project

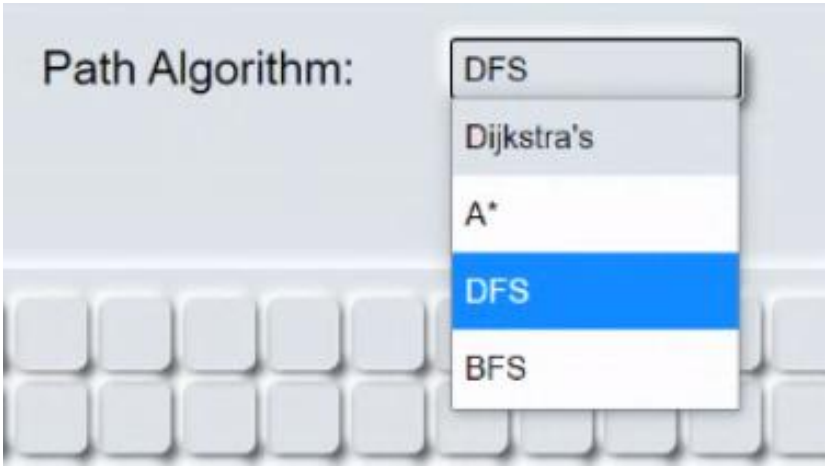
We have tried to create an intuitive UI that is easy to understand and use for the user . The major features of this project are described below. We have two main sections on our front page using which the user can choose to either go to the sorting visualizer or the path finding visualizer.



The website is divided into two sections whose features are described further-

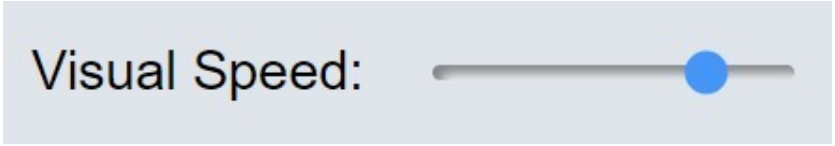
Path Finding Visualizer

- We have created an option to easily browse through and choose among the various path finding algorithms



- There is also the option to choose whether the layout should be weighted or unweighted so that you can see and understand the benefit of algorithms such as Dijkstra's

- We have also created an option to alter speed to understand and see things working at your own convenience.



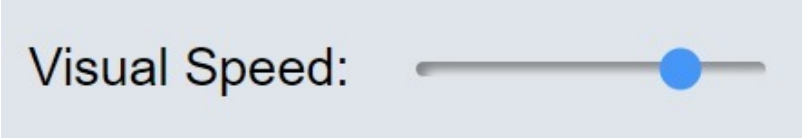
- Also there are three centrally placed buttons to start the process, reset the layout and reload the site

Sorting Visualizer

- There is the option to browse among and choose from various sorting algorithms

- We have also kept the option to change the array size as per you requirement .

- We have also created an option to alter speed to understand and see things working at your own convenience.



- Also there are two centrally placed buttons to start the process and reload the site

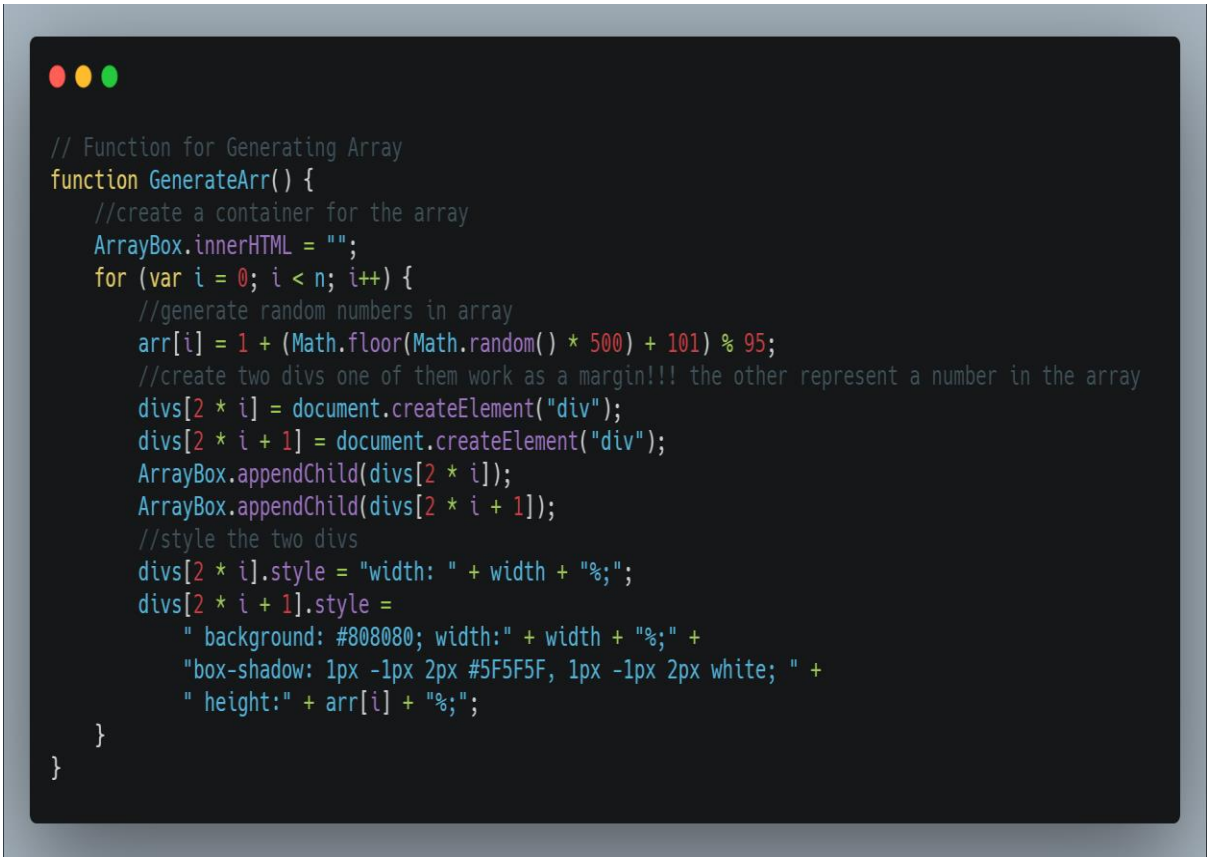
Project Deployment Link –
https://adityabhardwaj16-shivamgoel08.netlify.app/index.html

Project Github Repository Link –
https://github.com/adityabhardwaj16/algorithm-visualizer

Sorting Visualizer

In this there is a centered box inside which the generated array is display . We have used bars with heights proportional to their magnitudes in the array. This helps us in providing a visual representation of the array. We chose to particularly display in this way as this helps us intuitively identify which element is greater and which is smaller just by looking which is taller and which is shorter.

There is also the feature to generate new array to try these algorithms with various different arrays and each array that is generated is random and not hard coded. The following is the method used to achieve the above result.

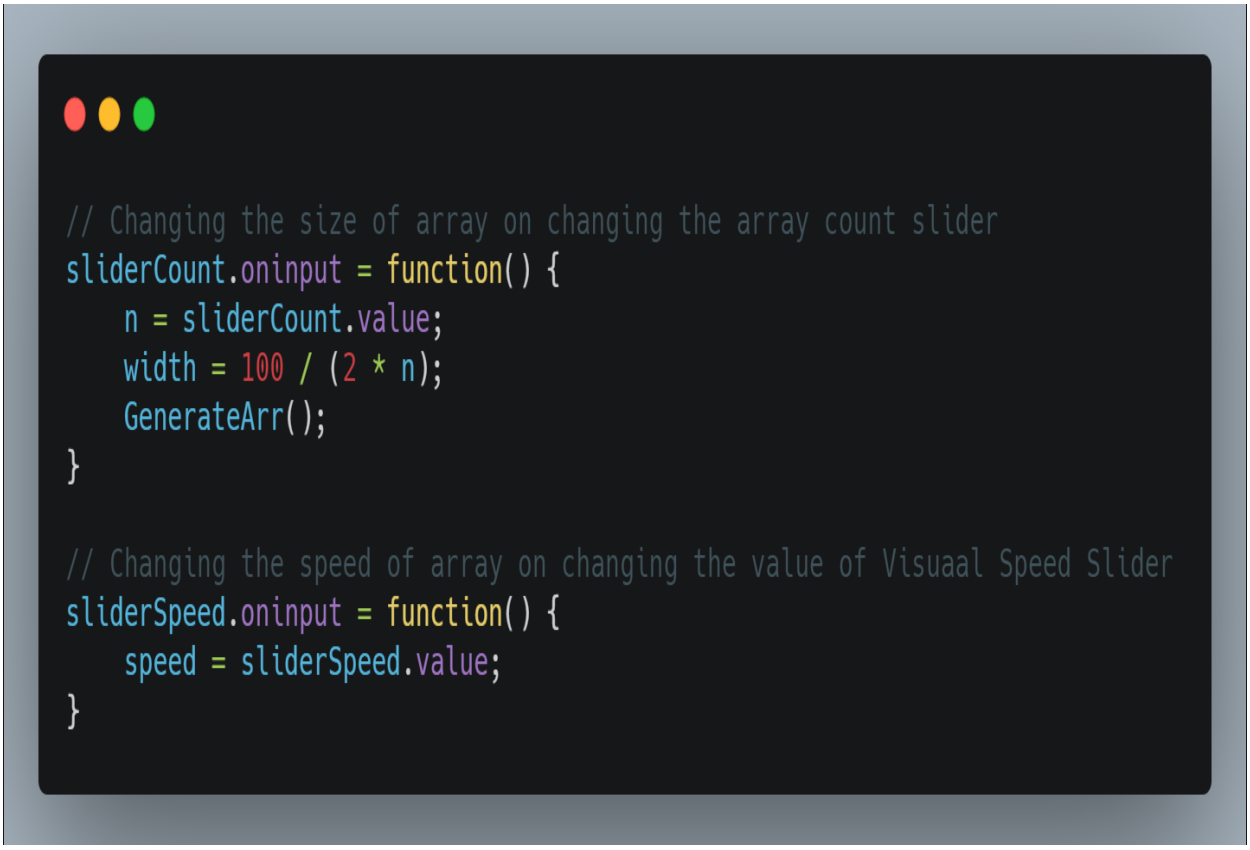


```
// Function for Generating Array
function GenerateArr() {
    //create a container for the array
    ArrayBox.innerHTML = "";
    for (var i = 0; i < n; i++) {
        //generate random numbers in array
        arr[i] = 1 + (Math.floor(Math.random() * 500) + 101) % 95;
        //create two divs one of them work as a margin!!! the other represent a number in the array
        divs[2 * i] = document.createElement("div");
        divs[2 * i + 1] = document.createElement("div");
        ArrayBox.appendChild(divs[2 * i]);
        ArrayBox.appendChild(divs[2 * i + 1]);
        //style the two divs
        divs[2 * i].style = "width: " + width + "%;";
        divs[2 * i + 1].style =
            " background: #808080; width:" + width + "%;" +
            "box-shadow: 1px -1px 2px #5F5F5F, 1px -1px 2px white; " +
            " height:" + arr[i] + "%;";
    }
}
```

Furthermore, each time a bar or a value of the array is being worked upon the corresponding bar for it gets red thereby showing where and on which index the work is being performed at that instant. Then we can see dynamically being moved to its next position achieved using JavaScript .

For this the way we achieve this is when we are in the loop executing for that particular array element we find its corresponding bar using DOM API built in JavaScript and color that red and just before executing that iteration of the loop we turn it back to grey.

We have implemented the speed controller by using a global variable for speed and having a function that is attached with an event listener that updates the global speed whenever speed is altered.



```
// Changing the size of array on changing the array count slider
sliderCount.oninput = function() {
    n = sliderCount.value;
    width = 100 / (2 * n);
    GenerateArr();
}

// Changing the speed of array on changing the value of Visuaal Speed Slider
sliderSpeed.oninput = function() {
    speed = sliderSpeed.value;
}
```

After this we obviously also have implemented all the above mentioned sorting algorithms.

As mentioned earlier we have implemented 5 sorting algorithms.

- Selection Sort

```
for(var i=0; i<height.length;i++)
{
    var s = i;
    var g = s;
    for(var j=i+1;j<height.length ;j++)
    {
        div_update(divs[2*(j)+1], height[j],'red');
        div_update(divs[2*(j)+1], height[j],'#808080');
        if(height[j] < height[s])
        {
            s = j;
            g = s;
        }
    }
    div_update(divs[2*(s)+1], height[s],'blue');
    if(s != i)
    {
        var temp = height[s];
        height[s] = height[i];
        height[i] = temp
    }
    div_update(divs[2*(s)+1], height[s],'#808080');
    div_update(divs[2*(i)+1], height[i],'green');
    div_update(divs[2*(i-1)+1], height[i-1],'green');
}
```

- Bubble Sort

```
//bubble sort normal algorithm
for (j = 1; j < n - i; j++) {
    if (height[j] < height[j - 1]) {
        var temp = height[j];
        height[j] = height[j - 1];
        height[j - 1] = temp;
    }

    //swap colors
    div_update(divs[2 * (j - 1) + 1], height[j - 1], "#808080"); //Color update
    div_update(divs[2 * j + 1], height[j], "red"); //Color update
}
```

- Insertion Sort

```
for (i = 1; i < n; i++) {
    div_update(divs[2 * i + 1], height[i], "red");
    keyH = height[i];
    j = i - 1;
    while (j >= 0 && height[j] > height[j + 1]) {
        var temp = height[j];
        height[j] = height[j + 1];
        height[j + 1] = temp;
        div_update(divs[2 * (j + 1) + 1], height[j + 1], "green");
        div_update(divs[2 * j + 1], height[j], "red");
        j = j - 1;
    }
    div_update(divs[2 * (j + 1) + 1], height[j + 1], "green");
}
```

- Heap Sort

```
for (let i = Math.floor(n / 2) - 1; i >= 0; i--)
        heapify(divs, height, n, i);
    let len = n;
    for (let i = n - 1; i > 0; i--) {
        div_update(divs[2 * 0 + 1], height[0], "#red");

        [height[0], height[i]] = [height[i], height[0]];

        //Color update
        heapify(divs, height, --len, 0);
        div_update(divs[2 * i + 1], height[i], "green");

    }
```

- Quick Sort

```
function quicksort(divs,height, l, r){
    if(l<=r){
        div_update(divs[2*(l-1)+1], height[l-1],'yellow');
        div_update(divs[2*(l)+1], height[l],'yellow');

        div_update(divs[2*(r-1)+1], height[l-1],'blue');
        div_update(divs[2*(r)+1], height[l],'blue');

        var p = partition(divs,height,l,r);
```

```
        div_update(divs[2*(l-1)+1], height[l-1],'#808080');
        div_update(divs[2*(l)+1], height[l],'#808080');

        div_update(divs[2*(r-1)+1], height[l-1],'#808080');
        div_update(divs[2*(r)+1], height[l],'#808080');

        quicksort(divs,height,l,p-1)
        quicksort(divs,height,p+1,r);

        div_update(divs[2*(p-1)+1], height[p-1],"green");
        div_update(divs[2*(p)+1], height[p],'green');
    }
}
```

Similarly other sorting algorithms have also been implemented. In the project we have tried to maintain a good overall file structure so that anyone can easily go through the project understand its working.

Path Finding Visualizer

In our path finding visualizer we have created a grid like layout similar to very basic map that can be edited and played upon by the user . The user can build walls and choose the algorithms as well as reset the board or reload the page or start the entire path finding process by just the simple click of a button.

We have implemented all this using pure vanilla JavaScript and no external libraries for that matter.

There is also the choice for using weighted or unweighted scenario .
If supposing a user chooses weighted graphs we already have predefined weights for each grid element to the delight of our user so that the user is always ready to experiment.

The user can also choose and change the algorithms being used to get the path according to his/her wish .

We have also restricted the weighted graphs option to Dijkstra’s so as that is the only algorithm that works for finding path with weighted graphs rest don’t consider the weights.

```
function updateweight() {
    weighttype = weightbtn.options[weightbtn.selectedIndex].value;
    if (weighttype == 'Unweighted') refreshEmptyBoard();
    else {
        if (algorithm != 'Dstr') {
            algobtn.value = 'Dstr';
            algorithm = algobtn.options[algobtn.selectedIndex].value;
        }
        refreshBoard();
    }
    changeStart(10, 10);
    changeEnd(10, 30);
}

function updatealgo() {
    algorithm = algobtn.options[algobtn.selectedIndex].value;
    if (algorithm != 'Dstr') {
        weightbtn.value = 'Unweighted';
        weighttype = weightbtn.options[weightbtn.selectedIndex].value;
        refreshEmptyBoard();
    }
    else if (algorithm == 'Dstr') {
        if (weightbtn.value == 'Unweighted') refreshEmptyBoard();
        else refreshBoard();
    }
    changeStart(10, 10);
    changeEnd(10, 30);
}
```

Apart from this we have implemented the algorithms in a clean and concise manner trying to follow good software development practices.

- Dijkstra’s Algorithm

```
// Algo here
var seen = [startNode];
var checker = [startNode];
var counter = 1;
while (checker.length != 0) {
    checker.sort(function (a, b) {
        if (parseInt(a.getAttribute('cost')) < parseInt(b.getAttribute('cost'))) return 1;
        if (parseInt(a.getAttribute('cost')) > parseInt(b.getAttribute('cost'))) return -1;
        return 0;
    });
    let curr = checker.pop();

    // Important to parse string to integer
    let row = parseInt(curr.getAttribute('row'));
    let col = parseInt(curr.getAttribute('col'));
    if (weighttype == "Unweighted" && row == x2 && col == y2) break;
    let wall = parseInt(curr.getAttribute('wall'));
    if (wall == 1) continue;

    // Check up down left right
    let nextRow = row + 1;
    let prevRow = row - 1;
    let leftCol = col - 1;
    let rightCol = col + 1;
    let a = checkNode(nextRow, col, curr, checker, seen, counter);
    let b = checkNode(prevRow, col, curr, checker, seen, counter);
    let c = checkNode(row, leftCol, curr, checker, seen, counter);
    let d = checkNode(row, rightCol, curr, checker, seen, counter);
    counter++;
}
```

- Depth First Search (DFS)

```
/* ################################## Algo here ############################3*/

var seen = [];
let counter = 1;
fl=false;
traverse(startNode, seen, counter, endNode);

// Draw out best route
setTimeout(() => {
    startNode.setAttribute('class', 'ends')
    while (endNode.getAttribute('parent') != 'null') {
        endNode.setAttribute('class', 'Path_green')
        var coor = endNode.getAttribute('parent').split('|');
        var prow = parseInt(coor[0]);
        var pcol = parseInt(coor[1]);
        endNode = document.querySelector(`div[row="${prow}"][col="${pcol}"]`);
    }
    endNode = document.querySelector(`div[row="${x2}"][col="${y2}"]`);
    endNode.setAttribute('class', 'ends');
}, counter * time + 100);
```

- A* Algorithm

```
// Algo here
var seen = [startNode];
var checker = [startNode];
var counter = 1;
while (checker.length != 0) {
    checker.sort(function (a, b) {
        if (calc(a, x2, y2) < calc(b, x2, y2)) return 1;
        if (calc(a, x2, y2) > calc(b, x2, y2)) return -1;
        return 0;
    });
```

```
    let curr = checker.pop();
    console.log("Curr", curr);
    // Important to parse string to integer
    let row = parseInt(curr.getAttribute('row'));
    let col = parseInt(curr.getAttribute('col'));
    if (row == x2 && col == y2) break;
    let wall = parseInt(curr.getAttribute('wall'));
    if (wall == 1) continue;

    // Check up down left right
    let nextRow = row + 1;
    let prevRow = row - 1;
    let leftCol = col - 1;
    let rightCol = col + 1;
    let a = checkNode(nextRow, col, curr, checker, seen, counter);
    let b = checkNode(prevRow, col, curr, checker, seen, counter);
    let c = checkNode(row, leftCol, curr, checker, seen, counter);
    let d = checkNode(row, rightCol, curr, checker, seen, counter);
    counter++;
}
```

- Breadth First Search (BFS)

We have implemented the remaining algorithms in a similar fashion. We have also resolved many bugs that came us during testing of this application and have created something that we feel delivers on its expectation.

CHAPTER – 4

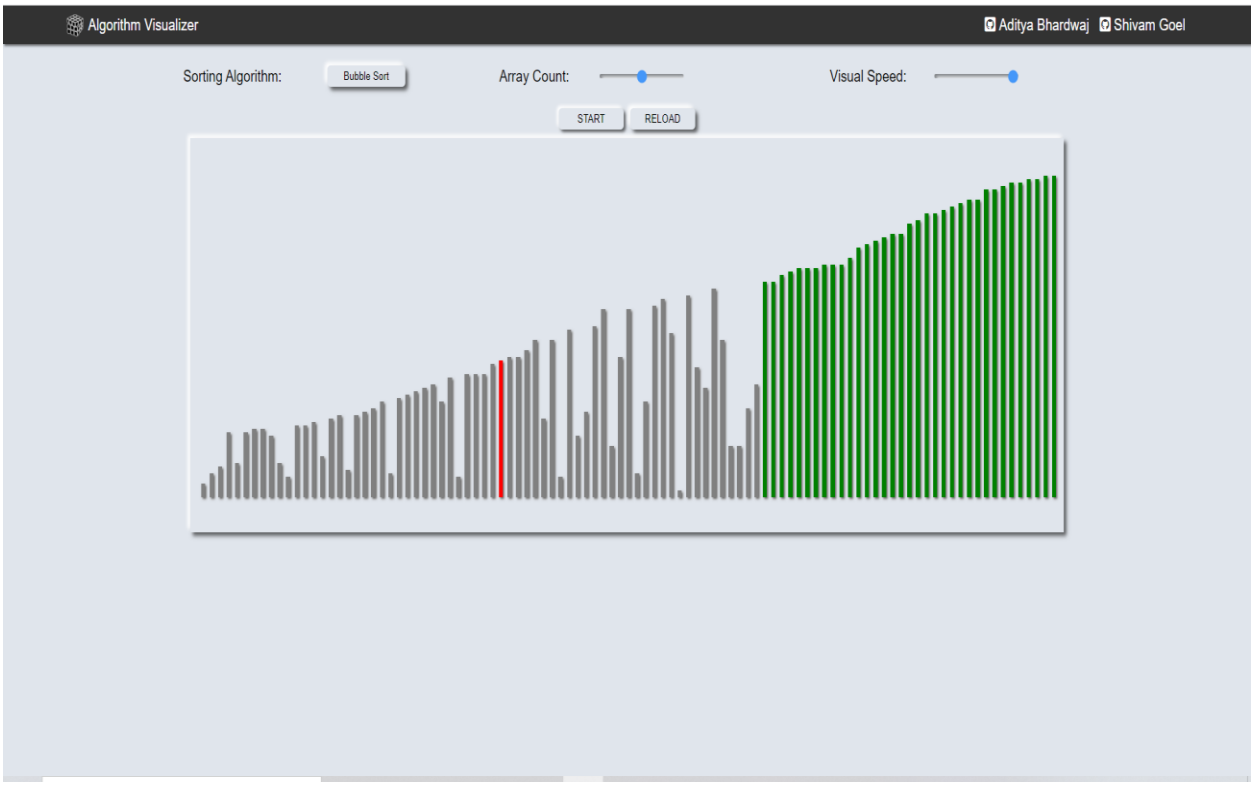
Results and Discussion

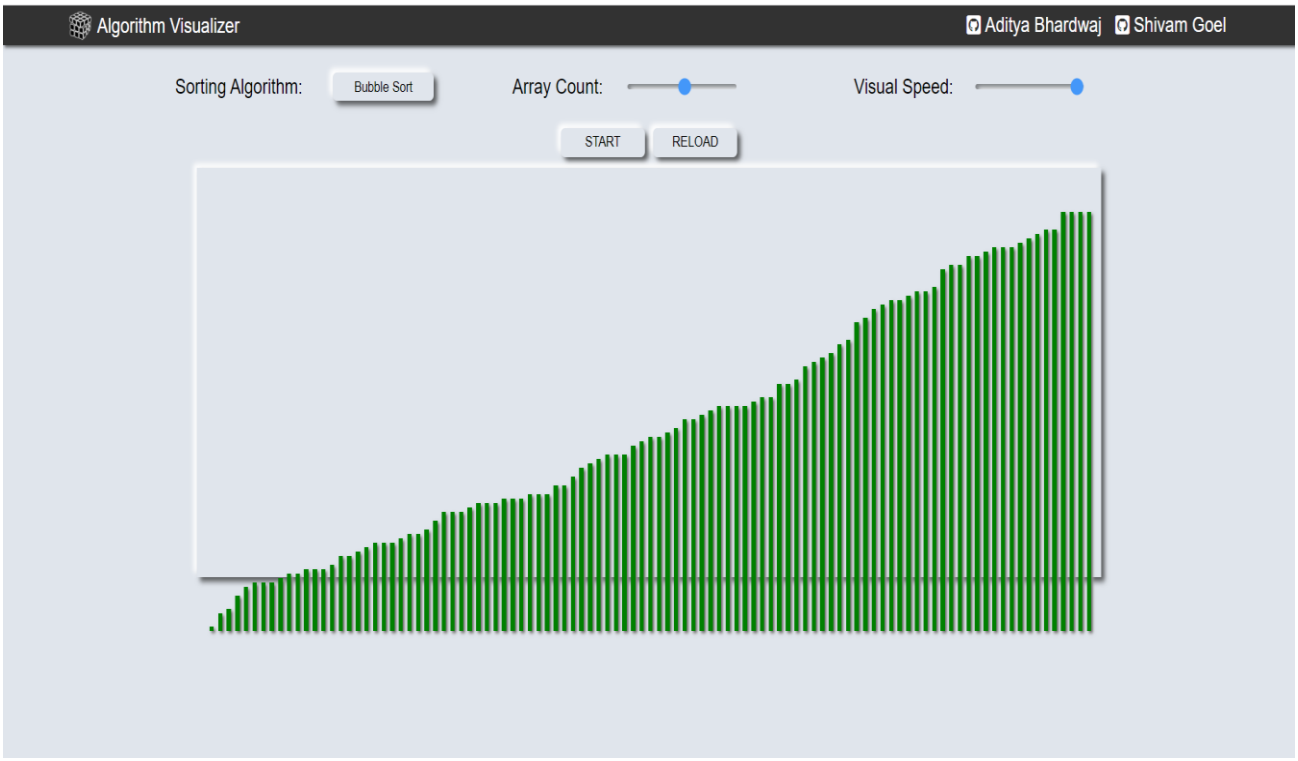
We successfully implemented our both sorting and pathfinding visualizers using all the above mentioned algorithms.

Sorting Visualizer

In this we have used red color to denote bars of array elements on which work is currently being done while green represents the ones that have been sorted and work on them is complete.

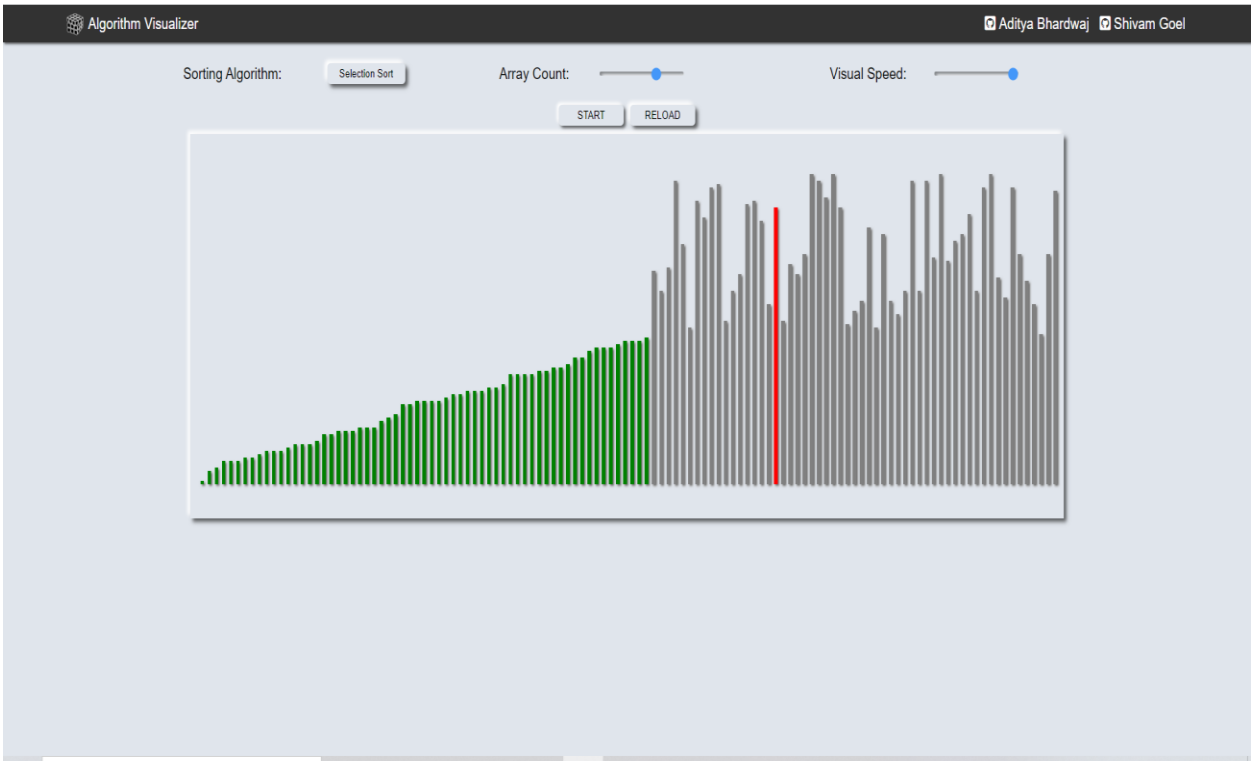
- Bubble Sort :

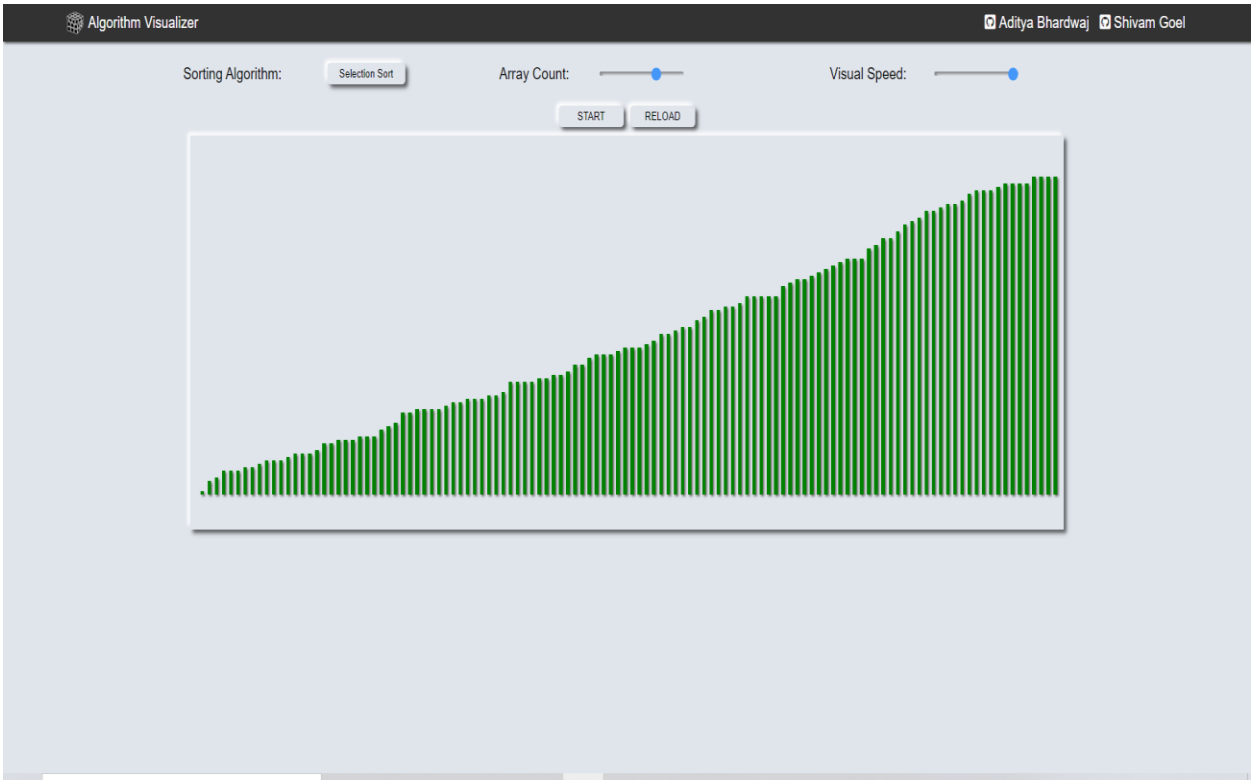




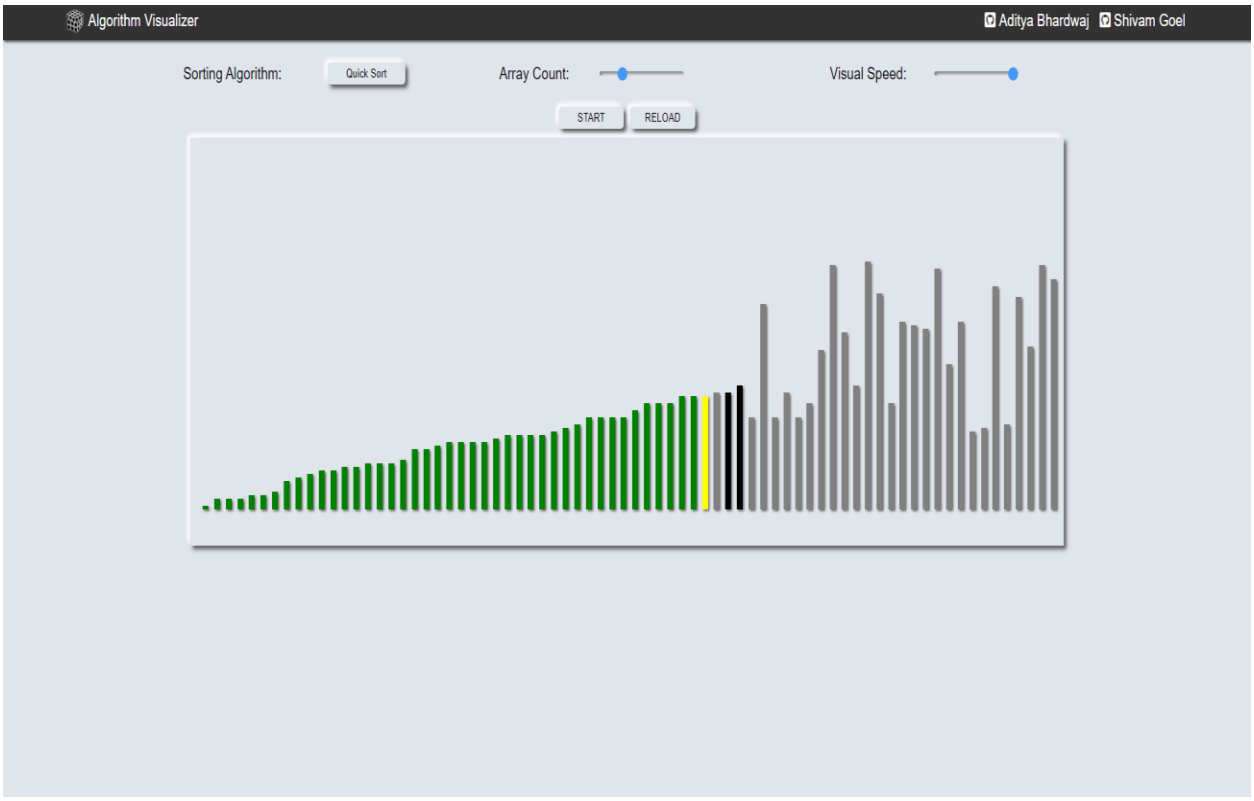
- Insertion Sort

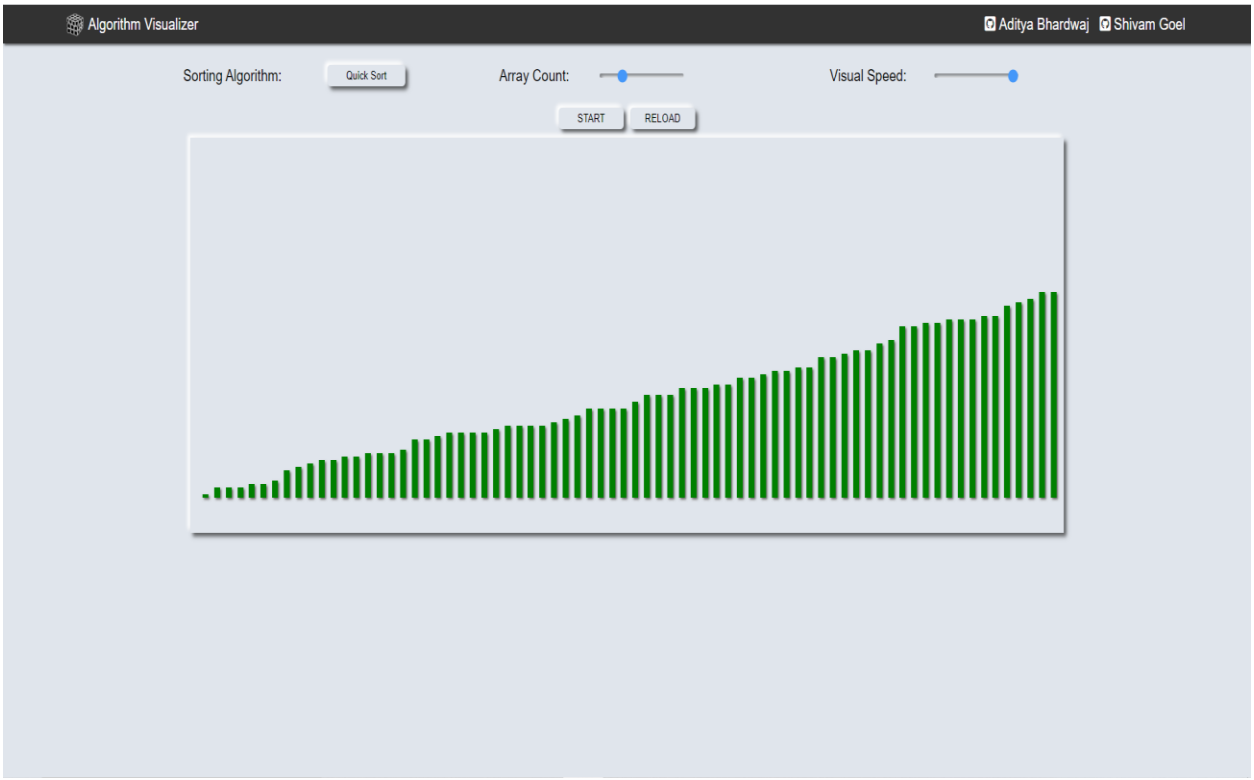
- Selection Sort



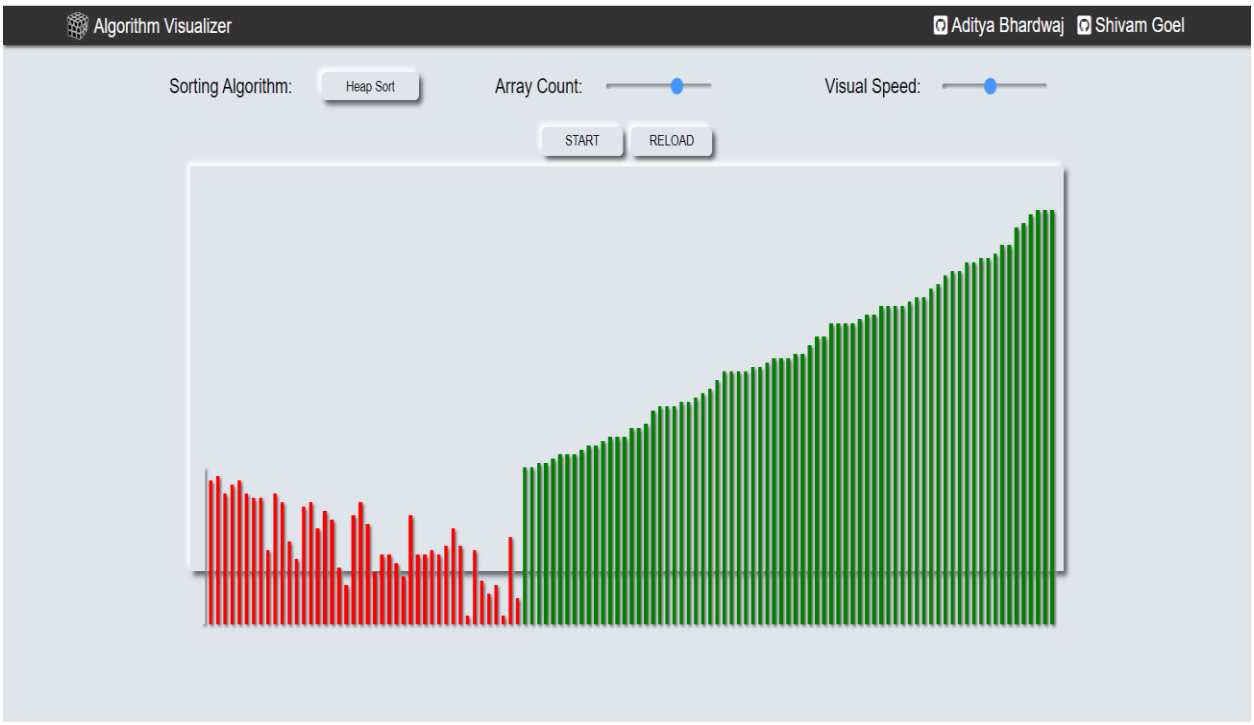

Algorithm Visualizer
Aditya Bhardwaj
Shivam Goel
Sorting Algorithm:
Selection Sort
Array Count:
Visual Speed:
START
RELOAD

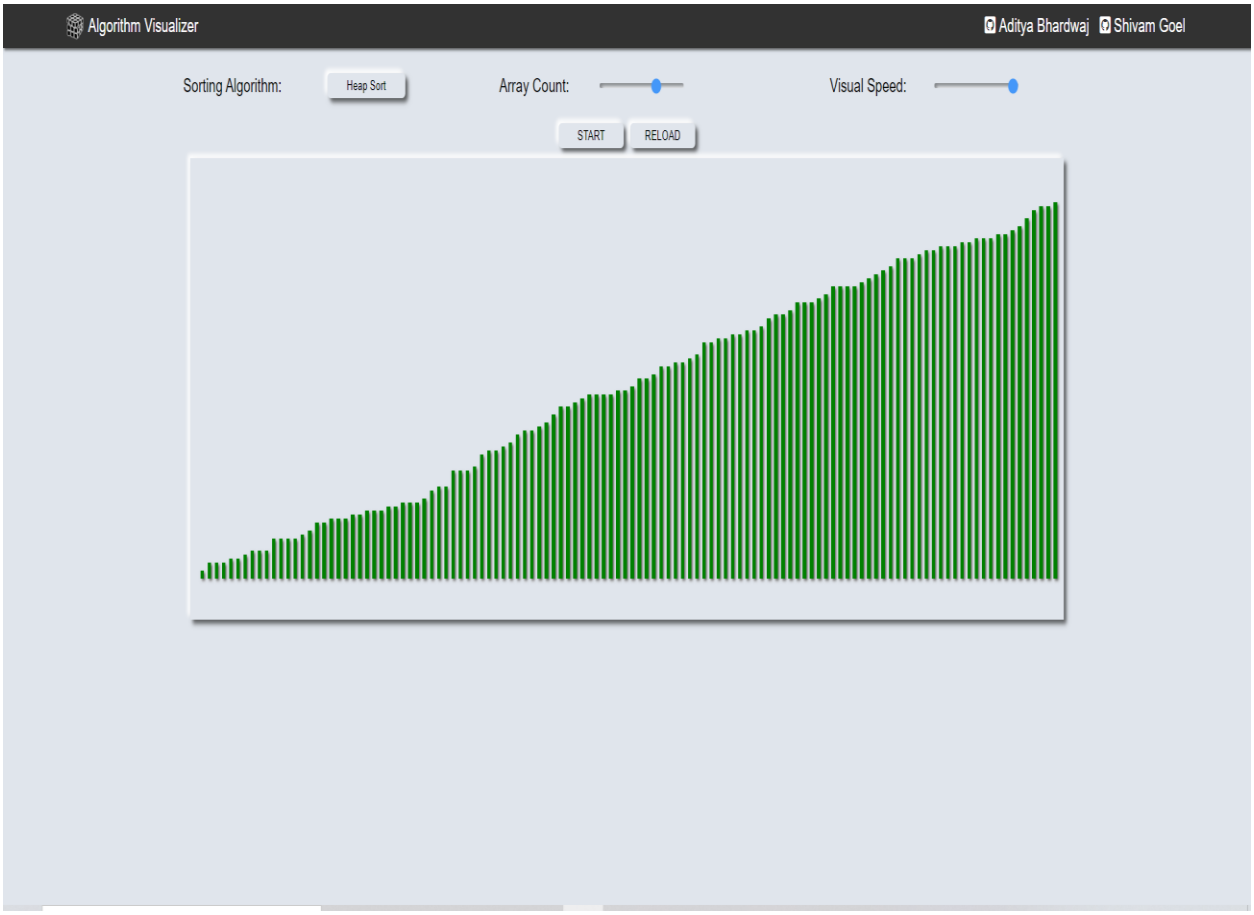
- Quick Sort


Algorithm Visualizer
Aditya Bhardwaj
Shivam Goel
Sorting Algorithm:
Quick Sort
Array Count:
Visual Speed:
START
RELOAD


Algorithm Visualizer
Aditya Bhardwaj
Shivam Goel
Sorting Algorithm:
Quick Sort
Array Count:
Visual Speed:
START
RELOAD

- Heap Sort


Algorithm Visualizer
Aditya Bhardwaj
Shivam Goel
Sorting Algorithm:
Heap Sort
Array Count:
Visual Speed:
START
RELOAD

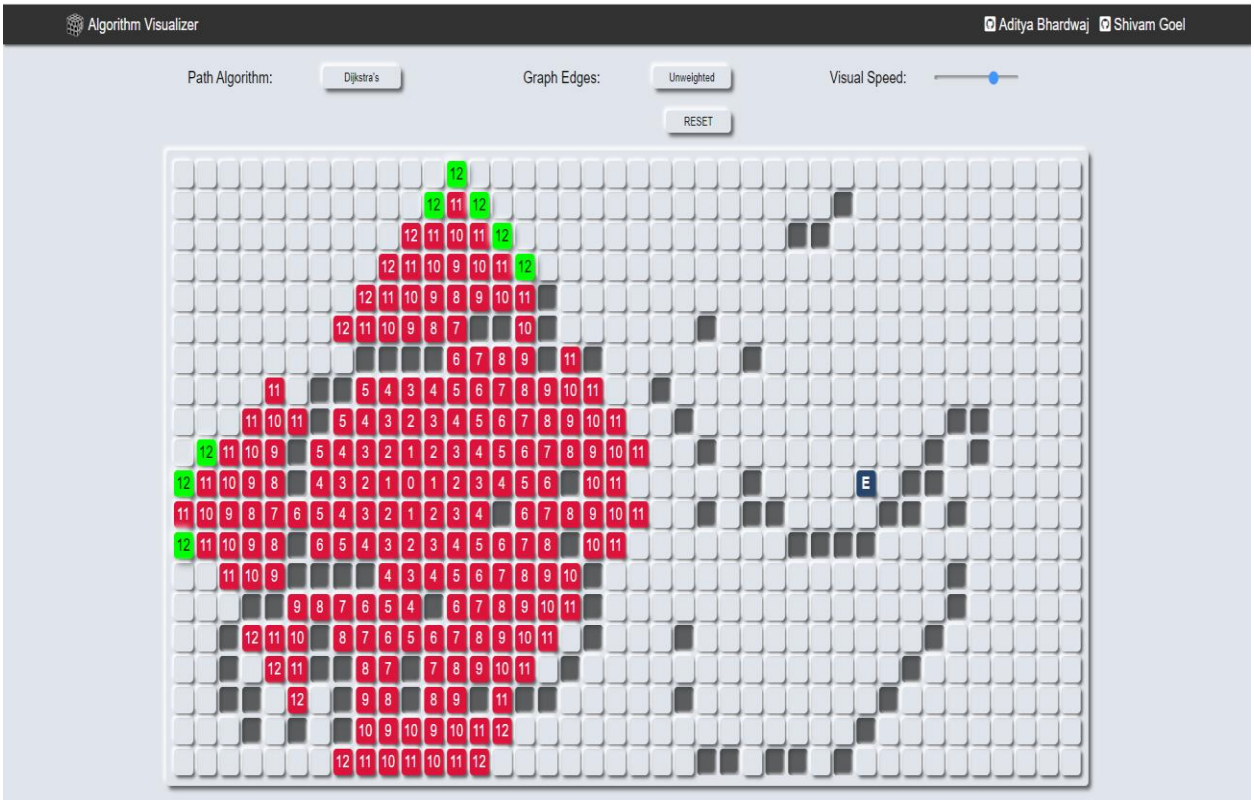


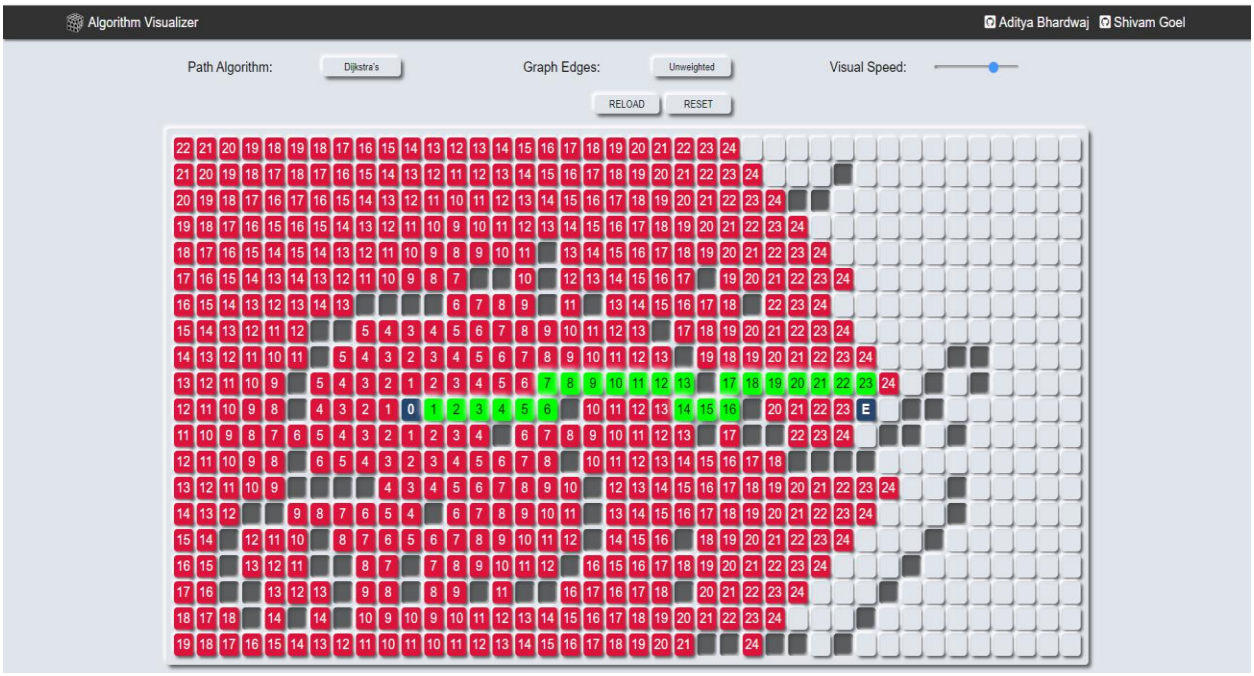
So, this way we have try to visualize the sorting algorithms where one graph depicts the sorting technique and the other graph depicts the final result of the sorting algorithm.
Similarly, we have implemented the Path Finding Algorithms.

Path Finder Visualizer

In this we have used numbers to depict their distance from the starting node in the manner the traversal is taking place for unweighted graphs while in case of weighted graphs we have used the numbers to display the respective weights of the grid elements . We have used green color to denote the path and red color to denote the visited nodes . Uncolored nodes are still unvisited and hence have no color.

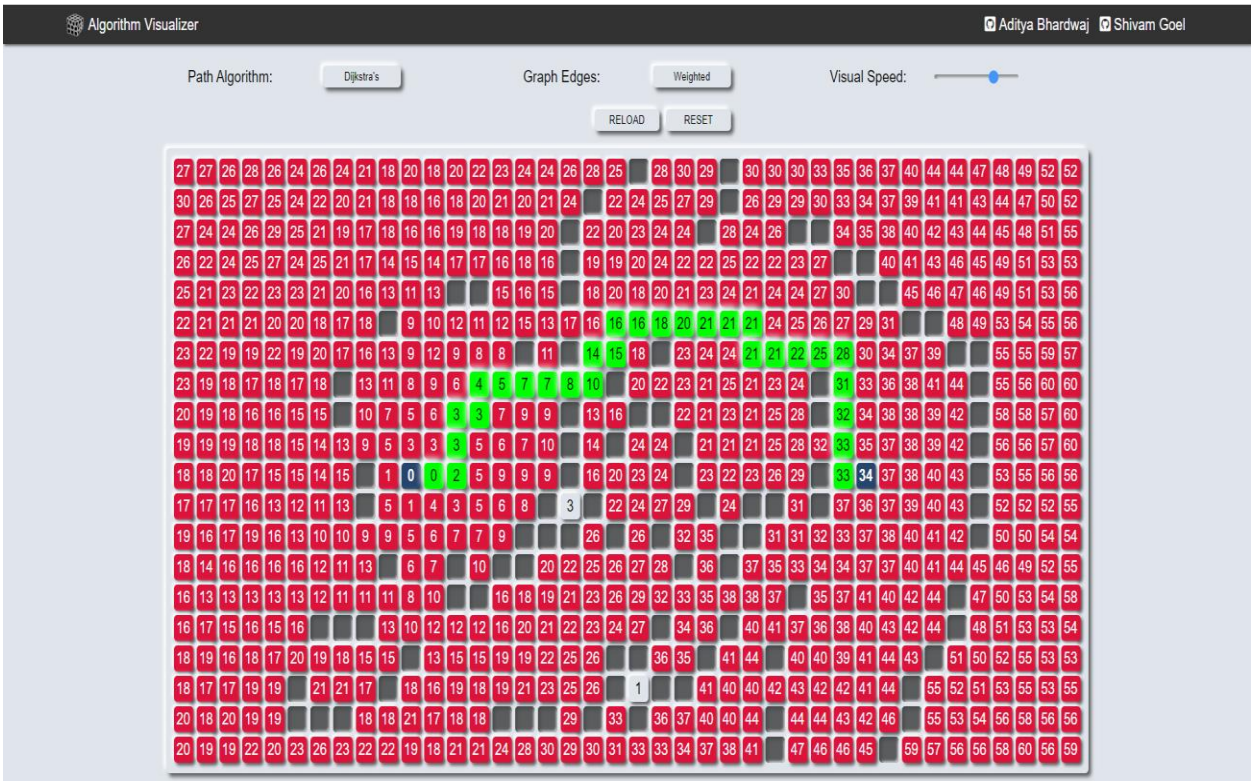
- Dijkstra’s Algorithm in unweighted graph



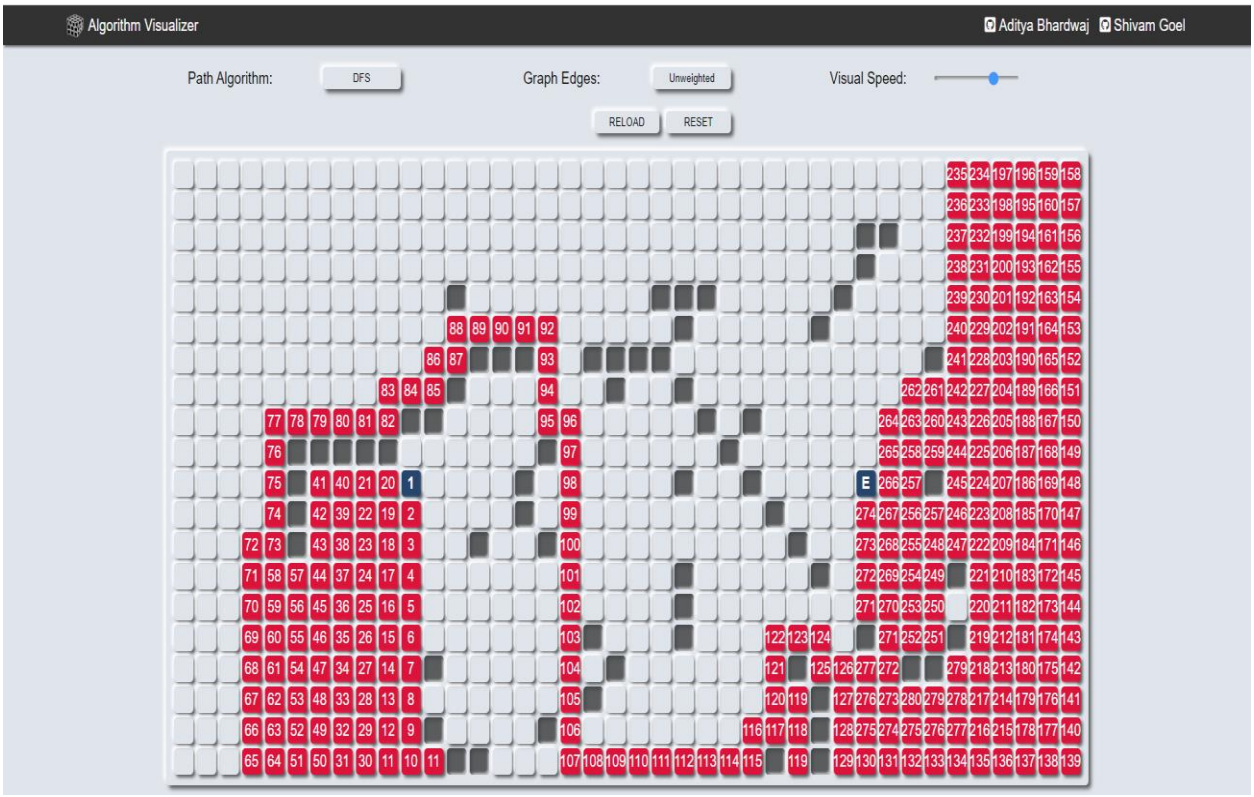


Now same we can visualize for a weighted graph.

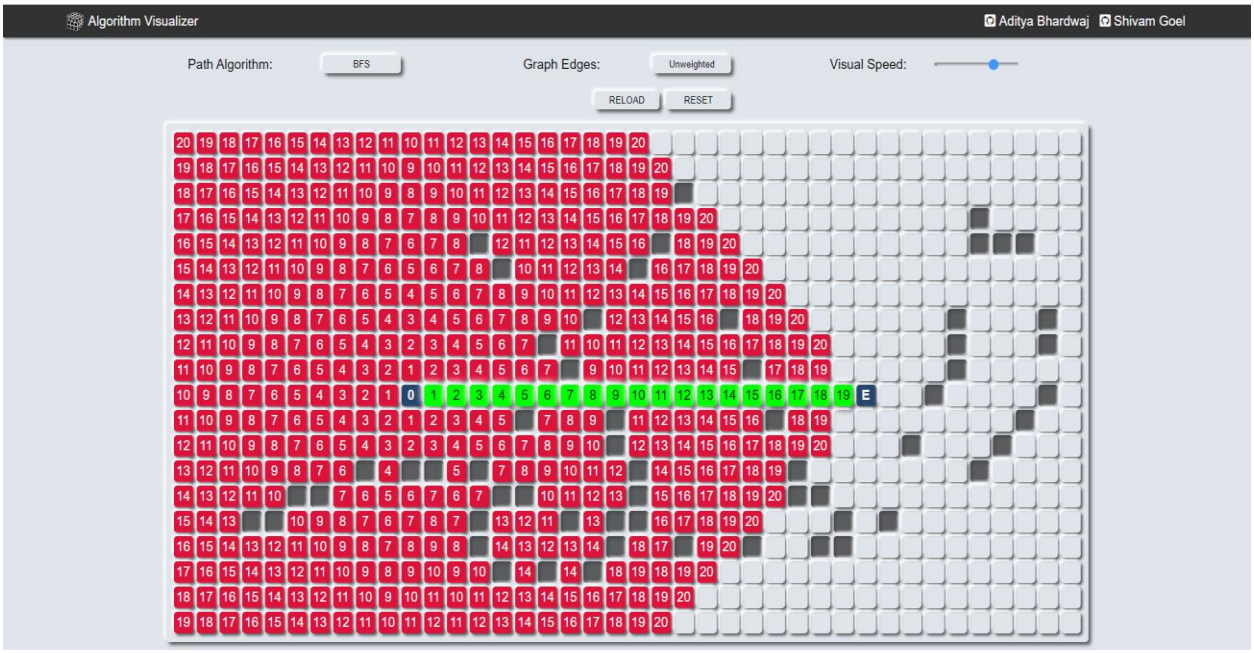
- Dijkstra's Algorithm in weighted graph



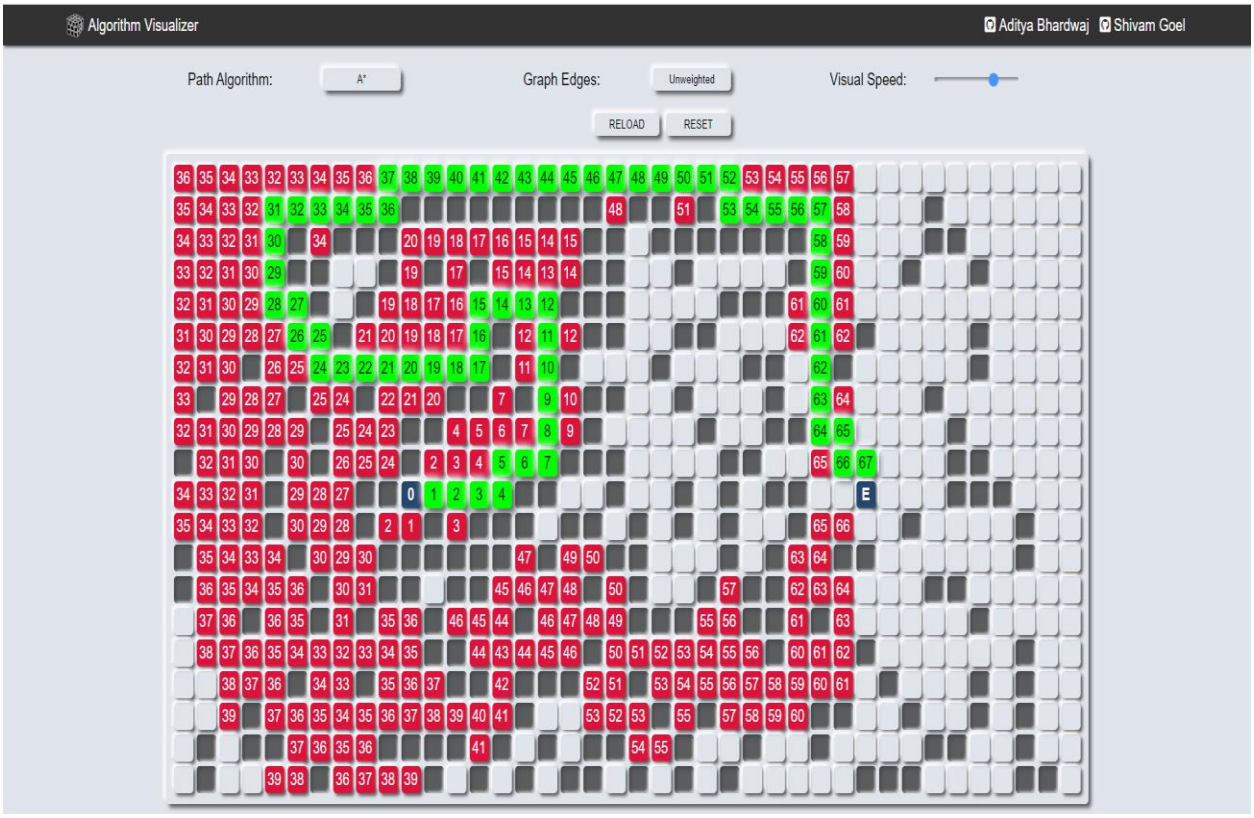
- Depth First Search (DFS)



- Breadth First Search (BFS)



- A* Algorithm


Algorithm Visualizer
Aditya Bhardwaj
Shivam Goel
Path Algorithm:
A*
Graph Edges:
Unweighted
Visual Speed:
RELOAD
RESET
E

CHAPTER – 5

Conclusion

We successfully implemented the algorithm visualizer in which we showed the real life visualization of various graph and sorting algorithms. We implemented depth first search (DFS) , breadth first search (BFS) , A* algorithm and Dijkstra's algorithm and tried to show that how maps work based on these algorithms. The very famous google maps are based on these graph algorithms. We also added various features like we can create walls or obstructions so as to make it more complex. We also used a very vibrant color scheme so as to make the project eye catchy. Moreover, the project also works for weighted edges.

We also implemented the various sorting algo's such as heap sort, quick sort, bubble sort and insertion sort. The project shows that how these sorting algorithms sort the array. We can also vary the speed of sorting so as to understand it clearly and one can also very the size of the array on which the sorting is to be applied. We also added the reset button and reload button for ease of accessibility of user. We also successfully implemented random generation of array feature so that the user can easily test various algorithms

When students initially study about these sorting algorithms they found it a little bit confusing and tricky especially quick sort. So, this project helps to understand the every aspect of these algorithms. The project gives a very clear perception about the algorithms and moreover when we see the algorithm actually working we will able to understand it in much better way and in very less time.

From development point of view, we got to learn about various frontend frameworks which will eventually help us in future. The overall experience we gained during making of this project is enormous.

Future Works

In this project we saw how we can use multiple graph and sorting algorithms together in one project for multiple use cases.

Not only that this project demonstrates how theoretical concepts like these can be implemented practically implemented and used create something that can be used in real world to impact a change and make our lives easier.

Now that we have made this project our future plan is to implement these on a proper platform so that students can use this project to increase their understanding of graph and sorting Algorithms. We plan to bring on more advanced algorithms like dynamic programming , backtracking, graph algorithms, etc. and establish it into a platform where students can code and visualize at the same time.

We also plan to add whiteboard functionality to the website so that students can perform dry run on the website itself and the website could function as a complete teaching tool for any teacher.

We would also like to give it a proper interface so as to convert it into a full-fledged working website. We will also try to improve the frontend part so as to improve the accuracy and speed of Algorithms.

Furthermore , we would like to add a discussion feature to the website like a forum with sections and pages for various topics where students and teachers can contribute quality work , articles, interesting and related things to foster research and development in the field of algorithm design and analysis.

Acknowledgement

We'd want to thank our professors and Galgotias University from the bottom of our hearts for giving us the excellent opportunity to work on this fantastic project on Algorithm Visualizer: Mitigate Algorithm Complexities, which has allowed us to conduct significant study and learn about many new things. Second, we'd like to thank Ms. Aanchal Vij, our guide, for her essential support in finishing this project in such a short amount of time.

References

We have taken help from the following resources to implement out project:

[1] K. Mehlhorn and P. Sanders, *Algorithms and Data Structures*. Berlin Heidelberg: Springer-Verlag, 2008.

[2] J. Genci, "Possibilities to solve some of the Slovak higher education problems using information technologies," in *2012 IEEE 10th International Conference on Emerging eLearning Technologies and Applications (ICETA)*, 2012.

[3] C. D. Hundhausen, S. A. Douglas, and J. T. Stasko, "A Meta- Study of Algorithm Visualization Effectiveness," *J. Visual Lang. Comput*, vol. 13, pp. 259–290.

[4] J. L. Bentley and B. W. Kernighan, "A System for Algorithm Animation," *Ooniptitinp Systems*, vol. 4, pp. 5–30, 1991.

[5] V. Ladislav and S. Veronika, "Algorithm Animations for Teaching and Learningthe Main Ideas of Basic Sortings"," *Informatics in Education*, vol. 16, no. 1, pp. 121–140, 2017.

[6] B. Edwards, "Drawing on the right side of the brain," in *CHI '97 extended abstracts on Human factors in computing systems looking to the future - CHI '97*, 1997.

[7] M. H. Brown and R. Sedgewick, “A system for algorithm animation,” *Comput. Graph. (ACM)*, vol. 18, no. 3, pp. 177–186, 1984.

[8] C. V. Jones, “Visualization and Optimization,” *J. Oper. Res. Soc.*, vol. 48, no. 9, pp. 964–964, 1997.

[9] B. A. Price, R. M. Baecker, and I. S. Small, “A Principled Taxonomy of Software Visu- alization,” *Journal of Visual languages and Computing*, vol. 4, no. 14, pp. 211–266, 1993.

[10] J. T. Stasko, “Tango: A Framework and System for Algorithm Animation,” *Jñññ Cont- puter*, vol. 23, pp. 27–39, 1990.

[11] B. A. Price, R. M. Baecker, and I. S. Small, “A Principled Taxonomy of Sofware Visualization’,” *Journal of Visual Languages and Computing*, vol. 4, no. 3, pp. 211–266.

[12] Wikipedia contributors, “A* search algorithm,” *Wikipedia, The Free Encyclopedia*, 13-Dec-2021. [Online]. Available:https://en.wikipedia.org/w/index.php?title=A*_search_algorithm&oldid=1060160033.