**INSERTION SORT VISUALIZER**

*Project report (CA3) submitted in the fulfilment of the requirements for the degree of*

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

By

**SURAJ SINGH**

**12005243**

SUBJECT

**INT219 FRONT END WEB DEVELOPMENT**



**School of Computer Science and Engineering**

Lovely Professional University

Phagwara, Punjab (India)

November 2022

**Table Of Contents**

1. Introduction
2. Technologies Used
3. Modules
4. Program and Website Snapshot
5. GitHub Link
6. References / Bibliography

**Introduction**

This sorting visualizer is based on the insertion sort algorithm only.

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

The algorithm is efficient for small data values and is adaptive in nature, i.e., it is appropriate for data sets which are already partially sorted.

The insertion sort visualizer makes use of insertion sort to sort the random array or a manually entered array of any size and works by depicting the flow of elements from unsorted place to sorted by using three different colours which are Red, Green, Blue. This keeps on going at a constant pace until all the elements are sorted.

**Technologies Used**

1. HTML (Hypertext Markup Language)

Used to create the structure of the page

1. CSS (Cascading Style Sheet)

Used for designing and giving animation effects in the entire page

1. JavaScript

Used for doing the real task of sorting and re-rendering the updated frames on the screen quickly

1. P5js

A custom JavaScript library for creating Canvas and frames in the project easily and conveniently

1. Responsive UI design

To ensure website runs on all major screen sizes

**Modules**

1. Module1: index.html file
2. Module2: stylesheet
3. Module3:

Module 3 is subdivided into many internal submodules which are:

randomize()

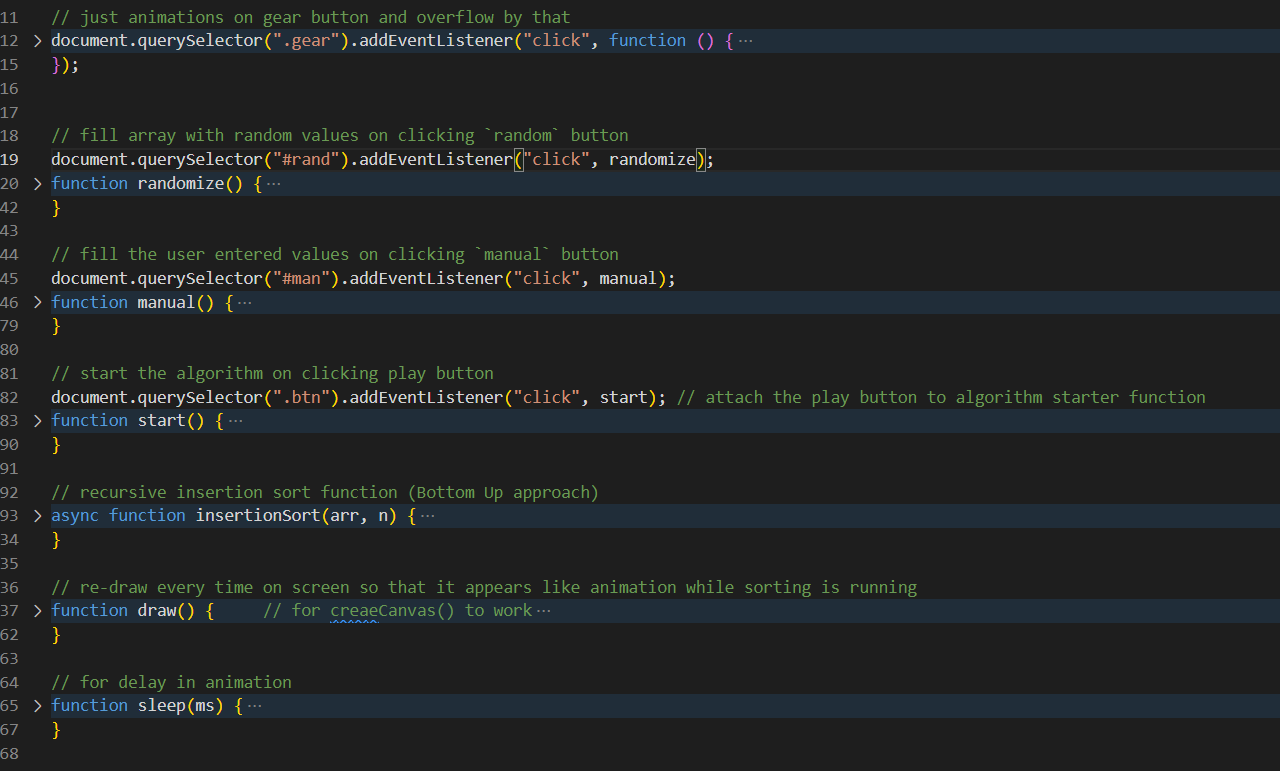
manual()

start()

insertionSort()

draw()

sleep()



The different modules in JS file:

// initial variables

let timer = 50;

let values = [];

let w = null;

let started = false;

let states = [];

let sort\_start = false;

let screenWidthMargin = 10;

let screenHeightMargin = 10;

// just animations on gear button and overflow by that

document.querySelector(".gear").addEventListener("click", function () {

  document.querySelector(".set").classList.toggle("opened");

  document.querySelector(".gear").classList.toggle("click");

});

// 🡪 fill array with random values on clicking `random` button

document.querySelector("#rand").addEventListener("click", randomize);

function randomize() {

  // validation: check if sorting is not going on, only then proceed

  if (sort\_start != true) {

    document.getElementsByClassName('initial-instr')[0].style.display = 'none';   // hide the initial instruction

    // set the required global values

    started = true;

    w = 50;

    // create a canvas in current window

    createCanvas(windowWidth - screenWidthMargin, windowHeight - screenHeightMargin);

    // create empty arrays of n = (windowWidth - 10) / W

    values = new Array(floor((windowWidth - screenWidthMargin) / w));

    states = new Array(floor((windowWidth - screenHeightMargin) / w));

    // fill the values and initialize state with -1

    for (let i = 0; i < values.length; i++) {

      values[i] = random(height - 120);

      states[i] = -1;

    }

  }

}

// 🡪 fill the user entered values on clicking `manual` button

document.querySelector("#man").addEventListener("click", manual);

function manual() {

  // validation: check if sorting is not going on, only then proceed

  if (sort\_start != true) {

    document.getElementsByClassName('initial-instr')[0].style.display = 'none'; // hide the initial instruction

    started = true;

    let textarr = []

    // if user has entered manual values, then get the space separated values

    if (document.querySelector(".arr-inp").value != "") {

      textarr = document.querySelector(".arr-inp").value.trim().split(/\s+/);

    }

    createCanvas(windowWidth - screenWidthMargin, windowHeight - screenHeightMargin);    // createCanvas() provided by p5

    values = new Array(textarr.length);

    states = new Array(textarr.length);

    const temp = []

    for (let i = 0; i < textarr.length; i++) {

      temp[i] = parseInt(textarr[i]);

    }

    w = ((windowWidth - screenWidthMargin) / temp.length);

    let max = Math.max(...temp);    // find maximum of all elements

    for (let i = 0; i < temp.length; i++) {

      values[i] = (temp[i] / max) \* (height - 110);

      states[i] = -1;

    }

  }

}

// 🡪 start the algorithm on clicking play button

document.querySelector(".btn").addEventListener("click", start);

function start() {

  document.querySelector(".set").classList.remove("opened");

  console.log(sort\_start, " : ", started);

  // validate the array, and ensure algo is not started yet

  if (sort\_start != true && started == true) {

    sort\_start = true;

    insertionSort(values, values.length);   // call the algo

  }

}

// 🡪 recursive insertion sort function (Bottom Up approach)

async function insertionSort(arr, n) {

  if (n <= 1)   // base case

    return;

  await insertionSort(arr, n - 1);    // recursively call until n = 1

  let last = arr[n - 1];

  let j = n - 2;

  states[j] = 1;

  while (j >= 0 && arr[j] > last) {

    // shift all the elements to right by one step, so as to make room for last element

    // so it can be placed at its sorted position

    await sleep(timer);

    arr[j + 1] = arr[j];

    // change the colors

    temp = states[j];

    states[j] = 1;        // current state = red (element moving to its sorted place from unsorted)

    states[j + 1] = temp;   // next state = previous current state

    j--;

  }

  // turn red element to blue as it reaches its sorted place

  states[j + 1] = 2;

  await sleep(timer);

  arr[j + 1] = last;            // put the last element to it's sorted place

  //Turn all the sorted elements to green

  //color red=1 blue=2 green=3

  for (let k = 0; k <= j + 1; k++) {

    states[k] = 3;

  }

  // if this is the last last element of the array then make it green too (happens in the outermost recursion call (last call))

  if (n == arr.length) {

    states[n - 1] = 3

    sort\_start = false;

  }

}

// 🡪 re-draw every time on screen so that it appears like animation while sorting is running

function draw() {     // for createCanvas() to work

  // validate the starting of the canvas

  if (started) {

    // background

    background('#7494EA');

    // draw according to states (color: 1, 2, 3 : red, blue, green)

    for (let i = 0; i < values.length; i++) {

      // yeh kabab ke bich ki haddi

      stroke(0);

      if (states[i] == 1) {

        fill("#B9314F");

      }

      if (states[i] == 2) {

        fill("#255C99");

      }

      else if (states[i] == 3) {

        fill("#68EDC6");

      }

      else if (states[i] == -1) {

        fill("#DEC3BE");    // default color

      }

      // create the rectangle with given values

      rect(i \* w, height - values[i], w, values[i]);

    }

  }

}

// 🡪 for delay in animation

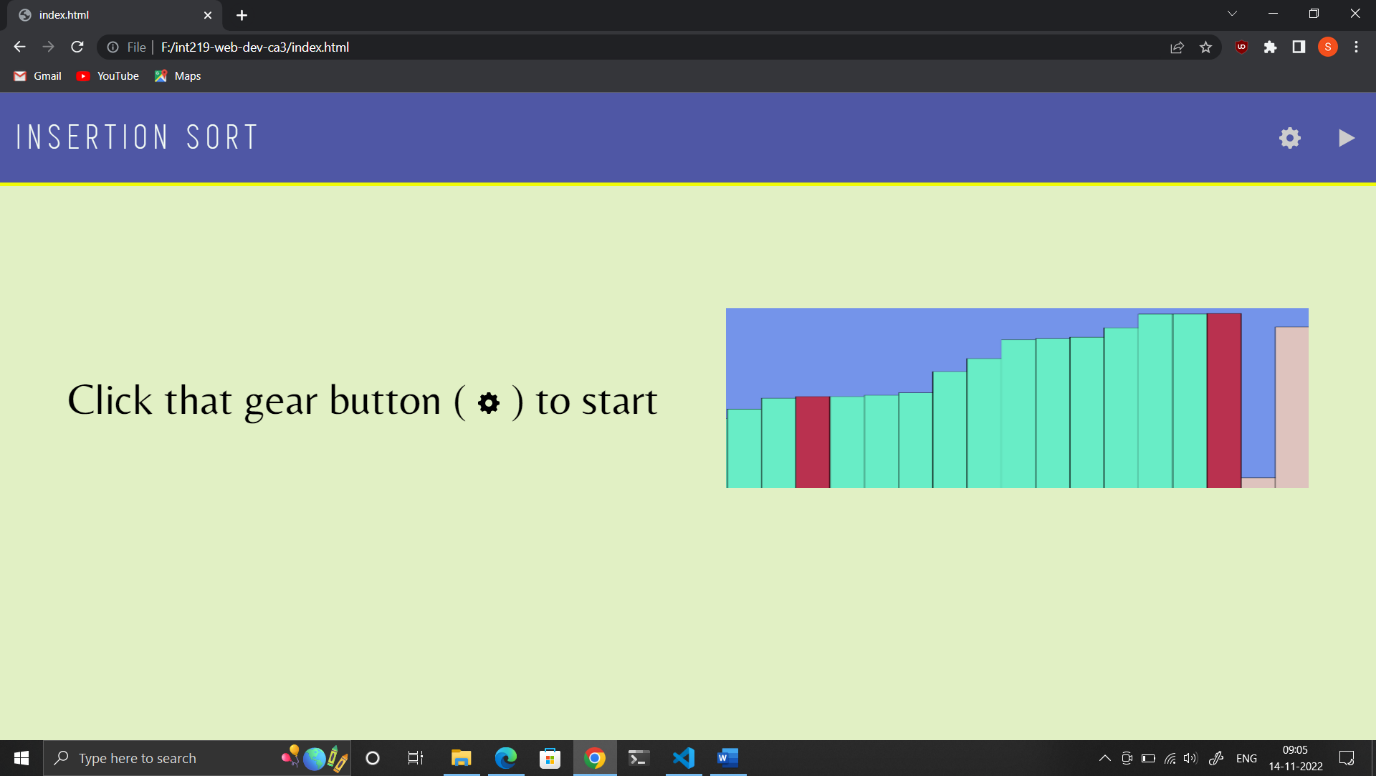
function sleep(ms) {

  return new Promise(resolve => setTimeout(resolve, ms));

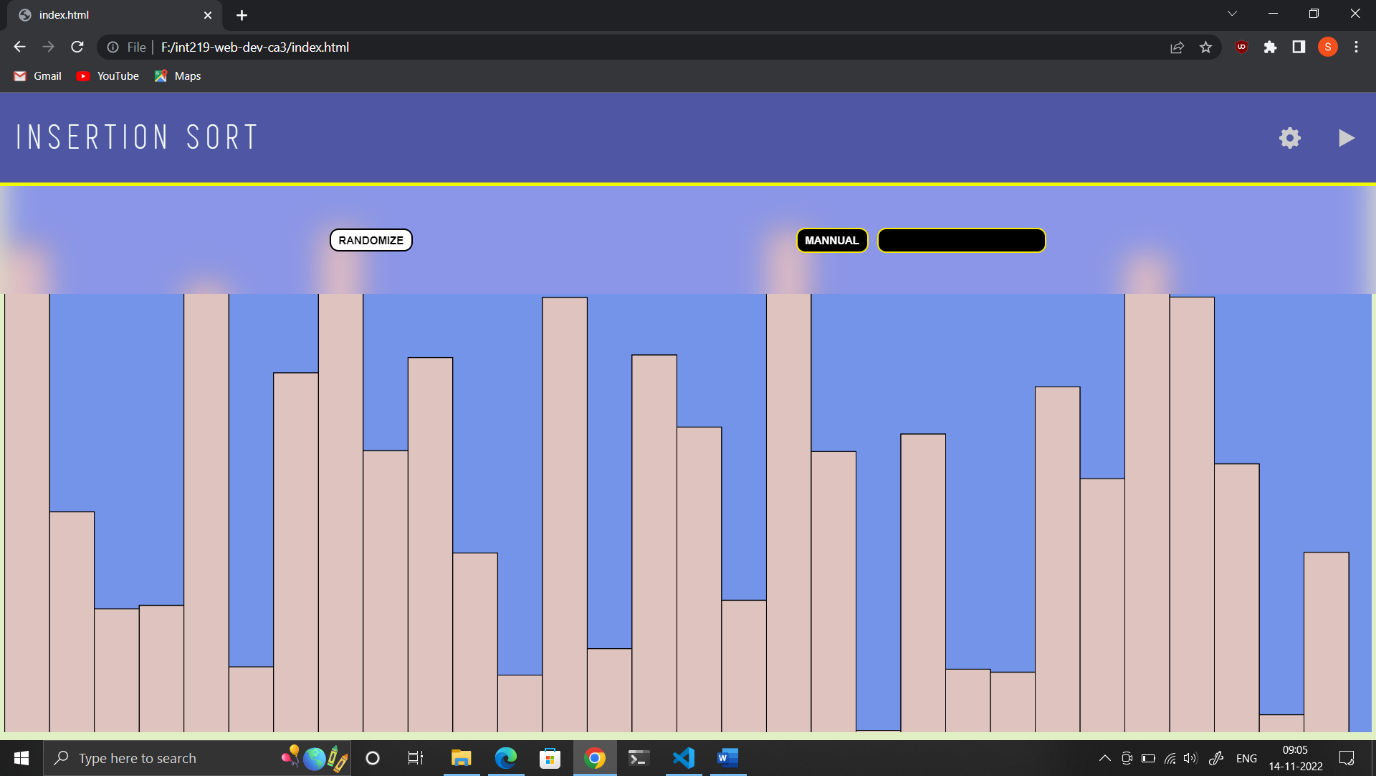
}

**Program and website snapshot**

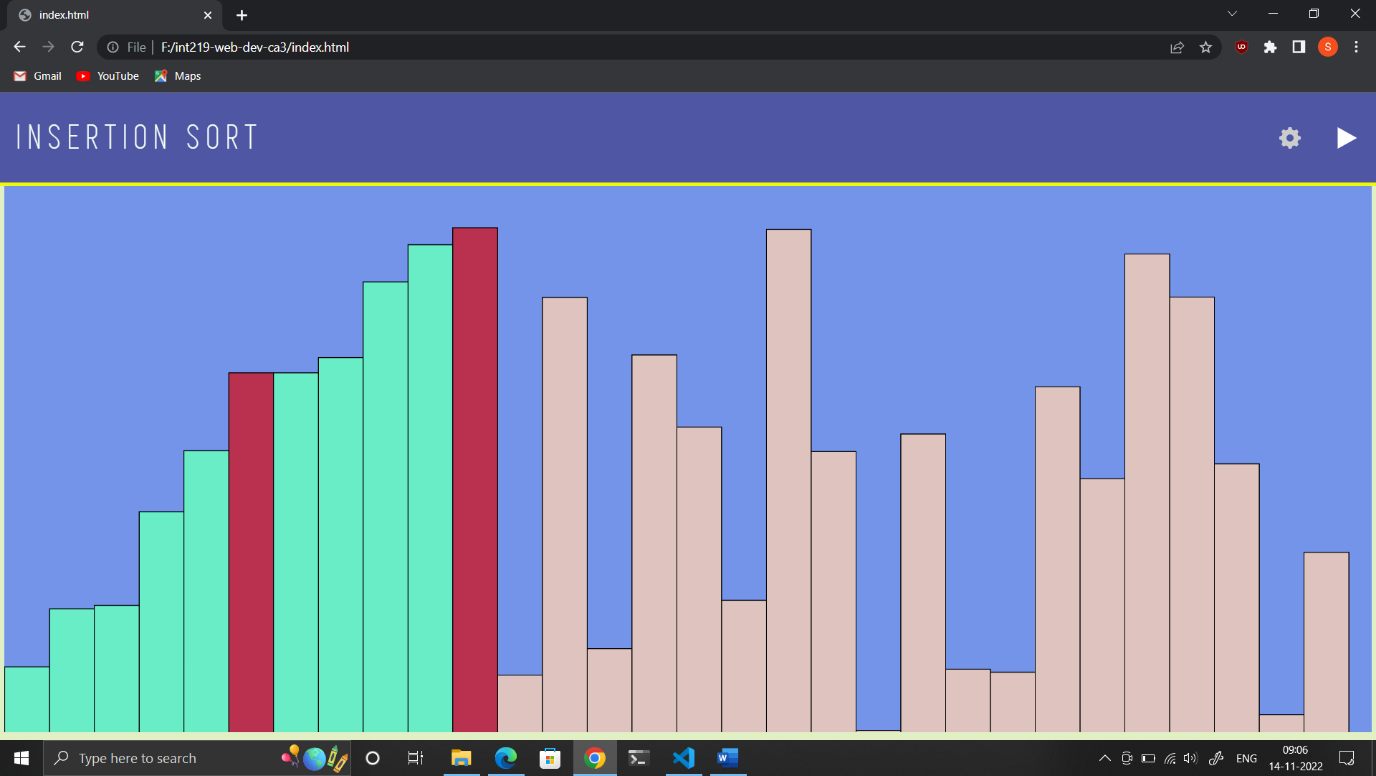
The Home / Landing Page

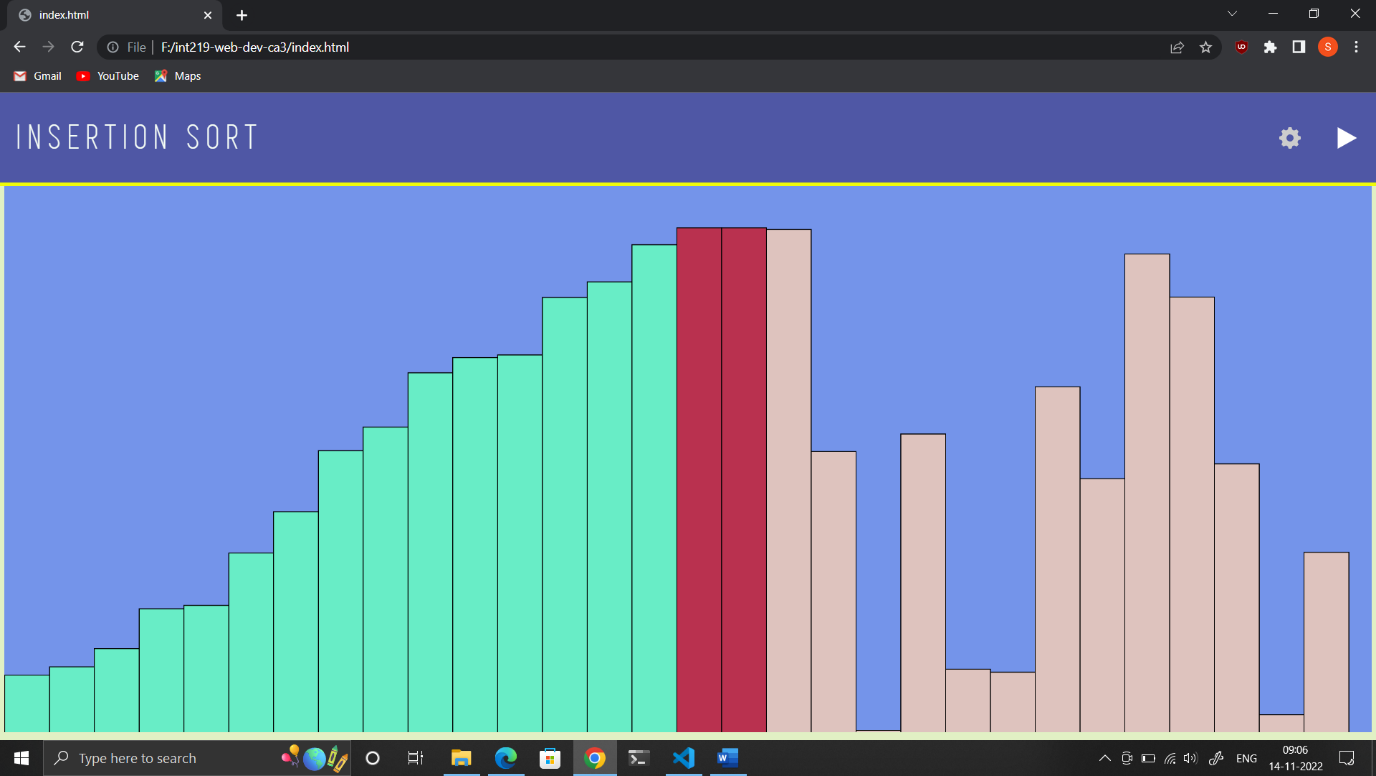


On click Randomize, the array is filled and program is awaiting start

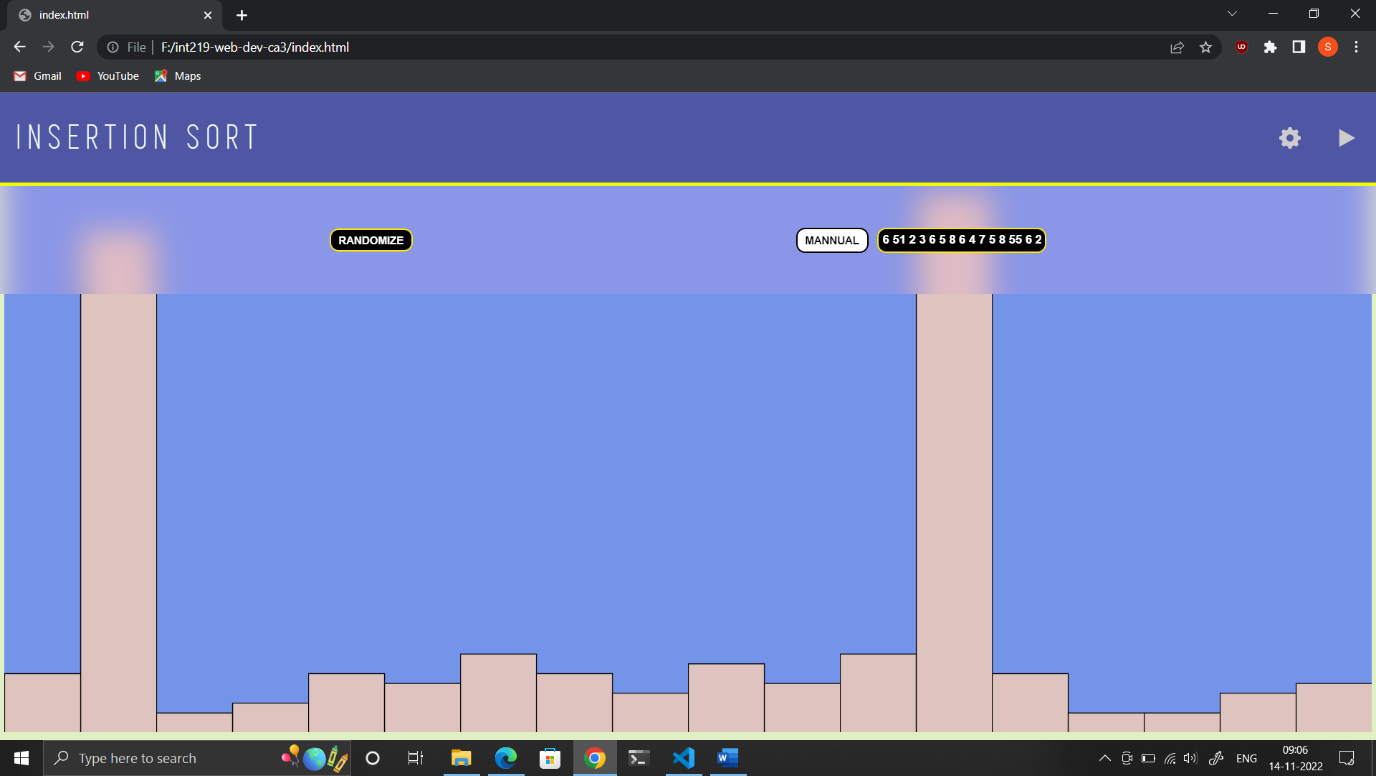


The sorting in visualization as the elements get sorted

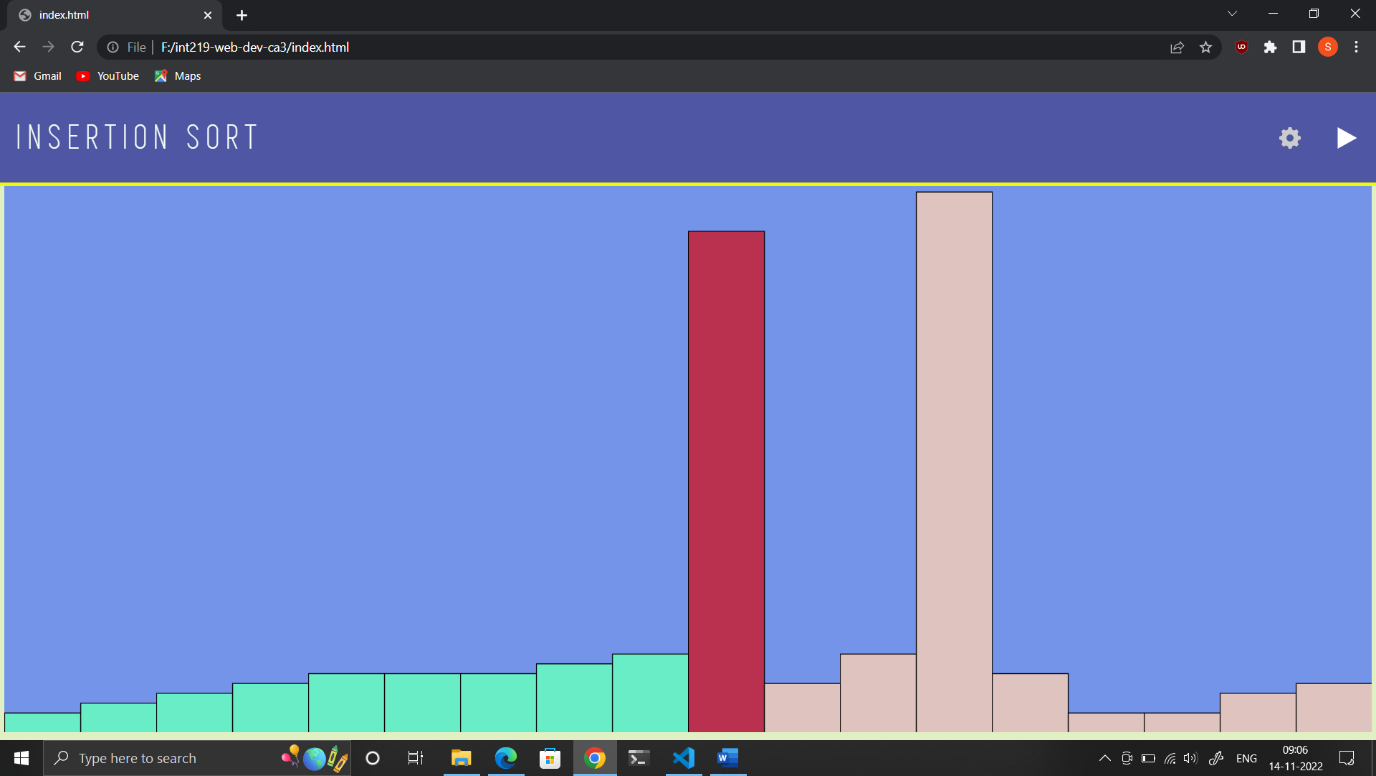




Manual creation of array by the user for sorting



Sorting of manually entered array elements



**GitHub Link**

Source code:

<https://github.com/suraj-singh12/int219-web-dev-ca3>

Live Deployed Project Link:

<https://suraj-singh12.github.io/int219-web-dev-ca3/>

**References / Bibliography**

1. <https://www.geeksforgeeks.org/insertion-sort/>

(Insertion sort algorithm)