# **DAA Programming Project-Report**

****1. Project 1 (Sorting Algorithms)****

****Implement and compare the following sorting algorithm :****

* ****Mergesort****
* ****Heapsort****
* ****Quicksort (Regular quick sort\* and quick sort using 3 medians)****
* ****Insertion sort****
* ****Selection sort****
* ****Bubble sort****

**Generating Random Variables:**

We are going to use getTextData() to generate random number of array according to the given size of the input for the sorting functions to sort out.

function getTextData() {

  if (

    document.getElementById("inputValue").value.length > 0 &&

    arrSelected.length > 0

  ) {

    var elementData = document.getElementById("inputValue").value;

    elementData = parseInt(elementData);

    inputData = Array(elementData)

      .fill()

      .map(() => Math.floor(100 \* Math.random()));

    document.getElementById("inputData").innerHTML +=

      "<b>" + "Input Data : " + inputData + "<br>";

    sortingAlgo(arrSelected, inputData);

  } else inputData = [];

}

We used some pre-defined method random in this.

**Execution Time Calculation:**

The total execution time is calculated by storing the start time into a variable and end time into another variable . total time taken is subtracting end time from the start time.

function sortingAlgo(selectedValues, Data) {

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

    switch (selectedValues[i]) {

      case "selection\_sort":

        const SortStarted = performance.now() + performance.timeOrigin;

        const Sort\_Array = selectionSort(Data);

        const SortEnded = performance.now() + performance.timeOrigin;

        getExcecutionTime(

          SortStarted,

          SortEnded,

          "Selection sort",

          Sort\_Array

        );

        break;

The same logic is used for all the sorting algorithm individually.

**Merge sort:**

Mergesort() is the main function we use to divide the input array. Function call array\_sorter() is used

to sort the array .

function mergeSort(GivenArray) {

  if (GivenArray.length <= 1) {

    return GivenArray;

  }

  //get the mid and slice the array

  const mid = Math.floor(GivenArray.length / 2);

  const leftside = GivenArray.slice(0, mid);

  const rightside = GivenArray.slice(mid, GivenArray.length);

  return array\_sorter(leftside, rightside);

}

function array\_sorter(leftside, rightside) {

  let sorted\_Array = [];

  while (leftside.length && rightside.length) {

    if (leftside[0] <= rightside[0]) {

      //add the element to the array and remove it

      sorted\_Array.push(leftside.shift());

    } else {

      sorted\_Array.push(rightside.shift());

    }

  }

  //if any elements left add to the array

  while (leftside.length) {

    sorted\_Array.push(leftside.shift());

  }

  while (rightside.length) {

    sorted\_Array.push(rightside.shift());

  }

  return sorted\_Array;

}

**Heap sort:**

Heapsort() is the main function and we use heapify to sort the array

function heapSort(GivenArray) {

  let arraylen = GivenArray.length;

  // Building heap

  for (let i = Math.floor(arraylen / 2) - 1; i >= 0; i--)

    heapify(GivenArray, arraylen, i);

  for (let i = arraylen - 1; i > 0; i--) {

    // Moving current root element to end element

    var temp = GivenArray[0];

    GivenArray[0] = GivenArray[i];

    GivenArray[i] = temp;

    heapify(GivenArray, i, 0);

  }

  return GivenArray;

}

function heapify(GivenArray, arraylen, i) {

  let biggest = i;

  let leftside = 2 \* i + 1;

  let rightside = 2 \* i + 2;

  // If leftside child element is larger than root element

  if (leftside < arraylen && GivenArray[leftside] > GivenArray[biggest])

    biggest = leftside;

  // If rightside child element is larger and biggest so far

  if (rightside < arraylen && GivenArray[rightside] > GivenArray[biggest])

    biggest = rightside;

  // If biggest element is not root element

  if (biggest != i) {

    var swap = GivenArray[i];

    GivenArray[i] = GivenArray[biggest];

    GivenArray[biggest] = swap;

    heapify(GivenArray, arraylen, biggest);

  }

}

**Quick sort:**

In this Quick sort we are selecting last element as our pivot element. In this we use quicksort() for before partition number and after partition numbers sorting . we also use quickSortPartition() and quicksortswap()

function quickSort(GivenArray, Firstvalue, lastvalue) {

  //termination condition

  if (Firstvalue < lastvalue) {

    let pie = quickSortPartition(GivenArray, Firstvalue, lastvalue);

    //sort elements before partition

    quickSort(GivenArray, Firstvalue, pie - 1);

    //sort elements after partition

    quickSort(GivenArray, pie + 1, lastvalue);

  }

  return GivenArray;

}

function quickSortPartition(GivenArray, firstvalue, lastvalue) {

  //lastvalue is selected as pivotElement

  const pivot = GivenArray[lastvalue];

  let i = firstvalue - 1;

  for (let j = firstvalue; j <= lastvalue - 1; j++) {

    // If current element is smaller than the pivot

    if (GivenArray[j] < pivot) {

      // Increment index of smaller element

      i++;

      quickSortSwap(GivenArray, i, j);

    }

  }

  quickSortSwap(GivenArray, i + 1, lastvalue);

}

function quickSortSwap(GivenArray, i, j) {

  const temp = GivenArray[i];

  GivenArray[i] = GivenArray[j];

  GivenArray[j] = temp;

}

**Quick Sort using 3 medians:**

function quickSort3Waypartition(array, left, right) {

  i = left - 1;

  j = right;

  var one = left - 1,

    two = right;

  var three = array[right];

  while (true) {

    //finding the element greater than or equal to three

    while (array[++i] < three);

    //finding the element smaller than or equal to three

    while (three < array[--j]) if (j == left) break;

    if (i >= j) break;

    //swapping smaller on left and larger on right

    var temp = array[i];

    array[i] = array[j];

    array[j] = temp;

    //moving same left occurences of pivot to the start of array

    if (array[i] == three) {

      one++;

      temp = array[i];

      array[i] = array[one];

      array[one] = temp;

    }

    //moving same right occurences of pivot to the start of array

    if (array[j] == three) {

      two--;

      temp = array[two];

      array[two] = array[j];

      array[j] = temp;

    }

  }

  // Move pivot element to its correct index

  var temp = array[i];

  array[i] = array[right];

  array[right] = temp;

  //moving all left occurrences from end to besides of arr[i]

  j = i - 1;

  for (k = left; k < one; k++, j--) {

    temp = array[k];

    array[k] = array[j];

    array[j] = temp;

  }

  //moving all right occurrences from end to besides of arr[i]

  i = i + 1;

  for (k = right - 1; k > two; k--, i++) {

    temp = array[i];

    array[i] = array[k];

    array[k] = temp;

  }

}

// 3-way partition based quick sort

function quickSort3Way(array, left, right) {

  if (right <= left) return;

  i = 0;

  j = 0;

  quickSort3Waypartition(array, left, right);

  // Recursion

  quickSort3Way(array, left, j);

  quickSort3Way(array, i, right);

  return array;

}

**Selection Sort:**

function selectionSort(GivenArray) {

  let min;

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

    //Taking the min value and comparing it with the rest

    min = i;

    for (let j = i + 1; j < GivenArray.length; j++) {

      if (GivenArray[j] < GivenArray[min]) {

        //if less than min replace it

        min = j;

      }

    }

    //swapping element with current element

    if (min !== i) {

      [GivenArray[i], GivenArray[min]] = [

        GivenArray[min],

        GivenArray[i],

      ];

    }

  }

  return GivenArray;

}

**Insertion Sort:**

function insertionSort(GivenArray) {

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

    //take the first element as key

    let val = GivenArray[i];

    let j = i - 1; //Last element of the array

    while (j >= 0 && val < GivenArray[j]) {

      GivenArray[j + 1] = GivenArray[j];

      j--;

    }

    GivenArray[j + 1] = val;

  }

  return GivenArray;

}

**Bubble Sort:**

function bubbleSort(GivenArray) {

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

    for (let j = 0; j < GivenArray.length; j++) {

      //check if j is greaterthan j+1

      if (GivenArray[j] > GivenArray[j + 1]) {

        //swapping

        let temp = GivenArray[j];

        GivenArray[j] = GivenArray[j + 1];

        GivenArray[j + 1] = temp;

      }

    }

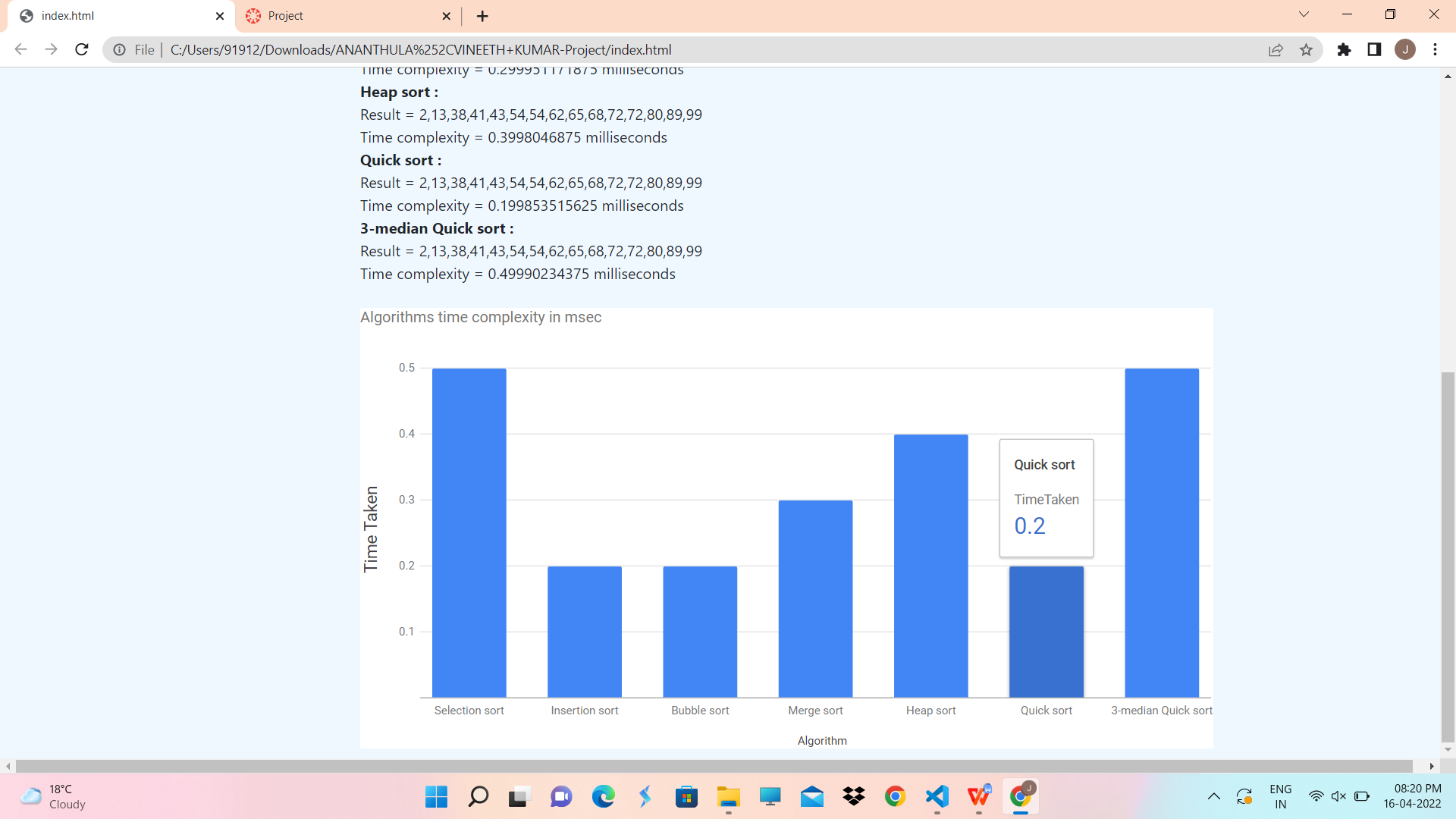
  }

  return GivenArray;

}

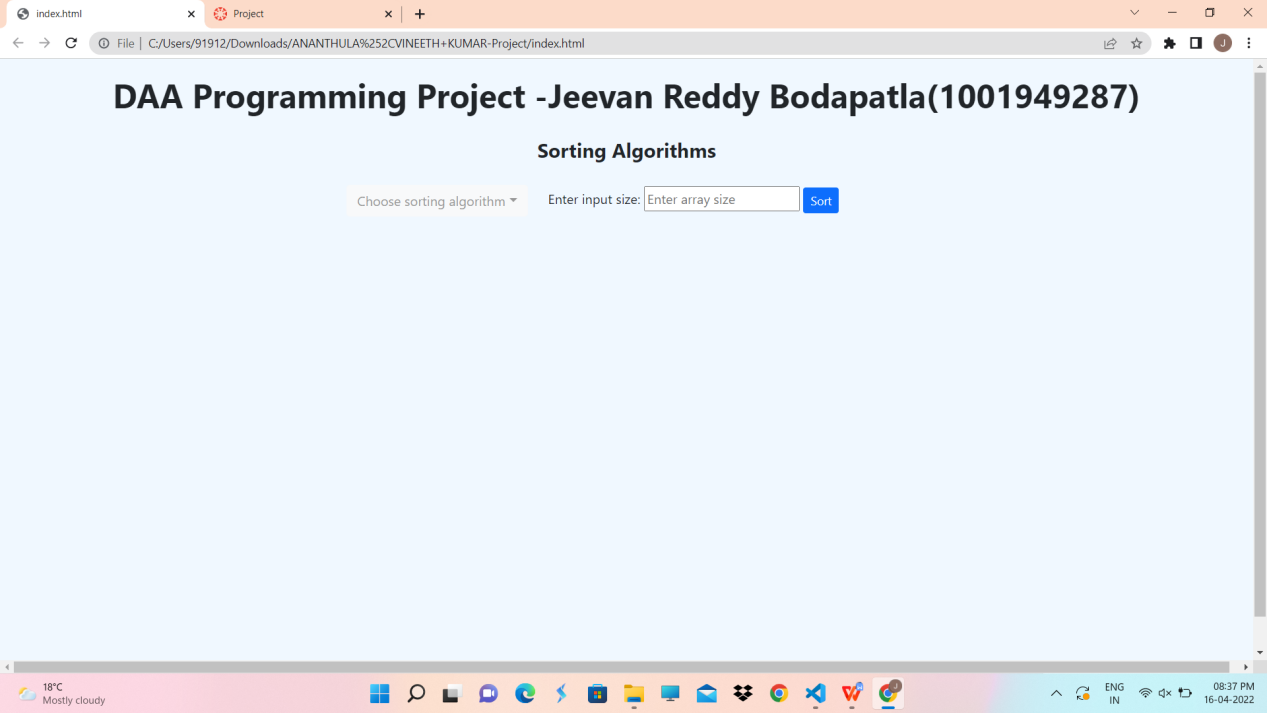
**Comparing all the Sortings:**

In here we have given 15 as the array size.we are comparing the time taken by all the algorithms for the same input.

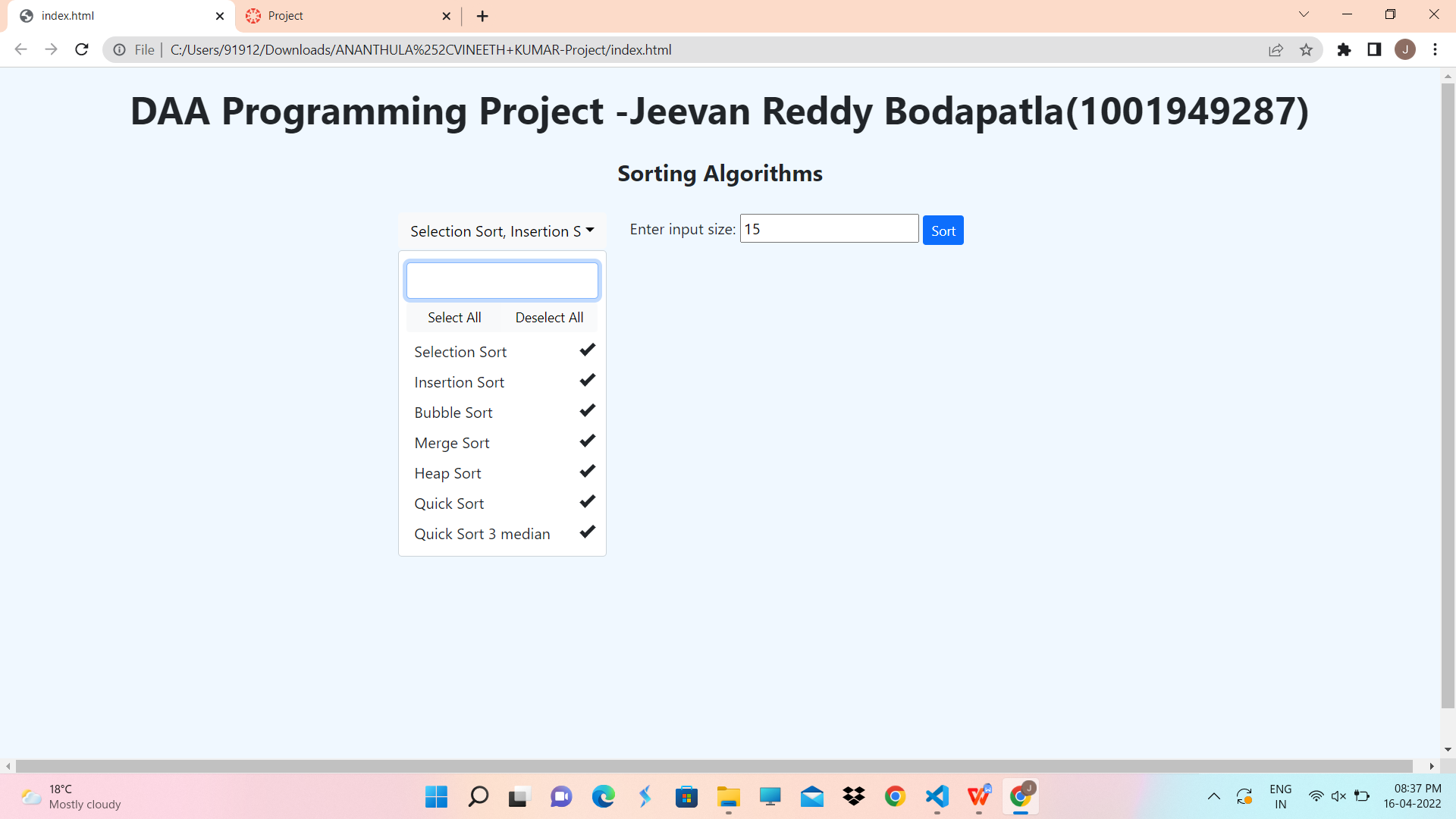


**GUI:**

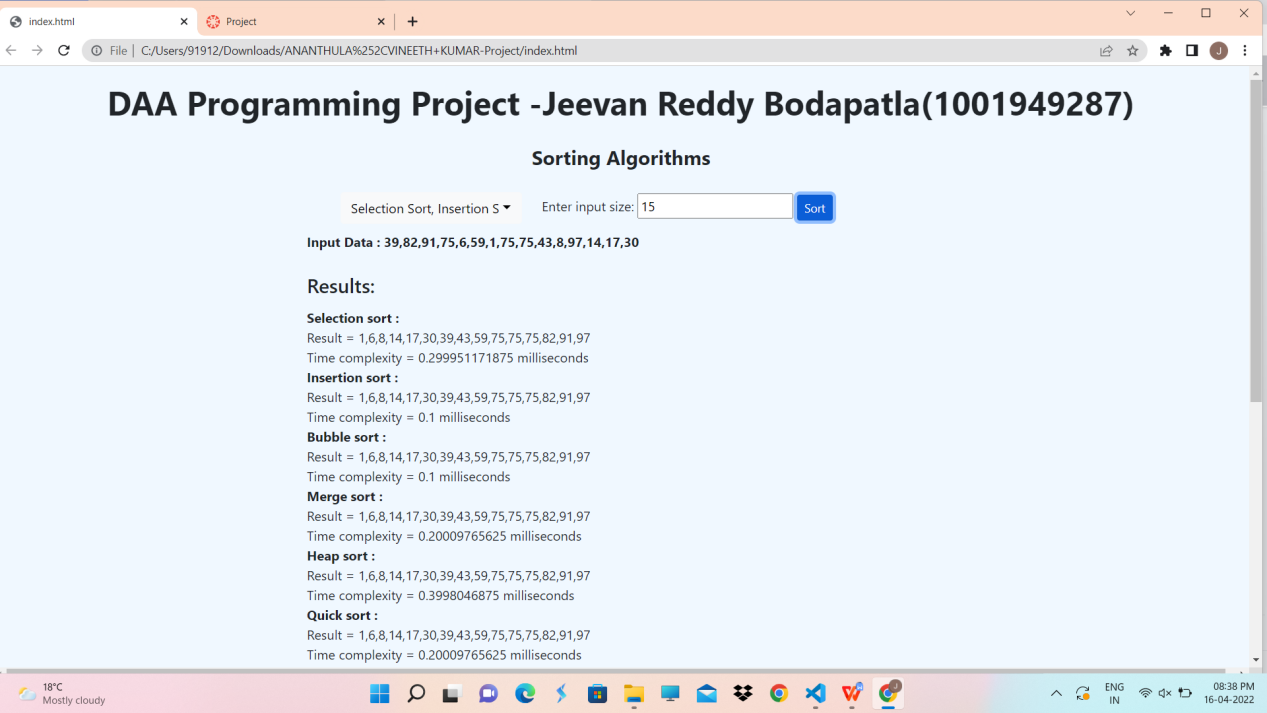
**Home page**

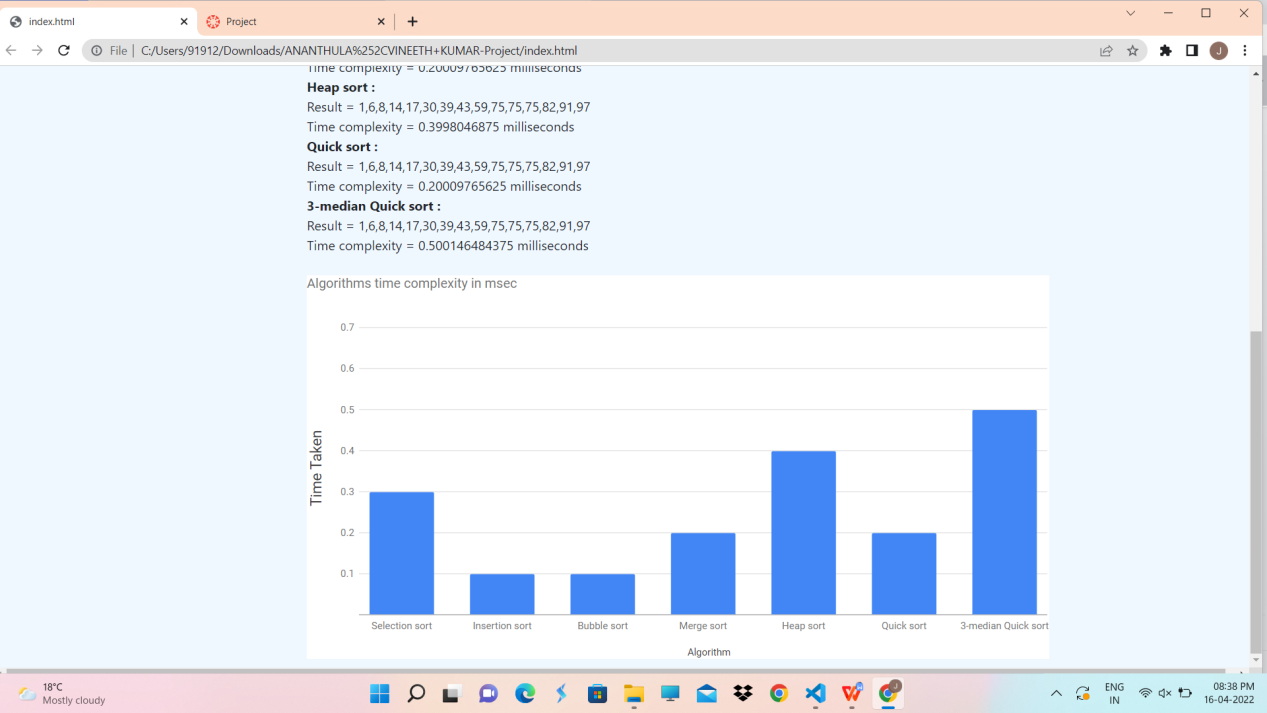
****

**Now select the required algorithms that you want to compare and enter the size**

****

**The output and the bar graph:**

****

****

**Jeevan Reddy Bodapatla**

**1001949287**