Sorting Algorithms

Below are the Sorting Algorithms which are implemented in JavaScript:

1. Merge Sort
2. Heap Sort
3. Quick Sort
4. Insertion Sort
5. Selection Sort
6. Bubble Sort

**Merge Sort:**

Merge Sort is a Divide and Conquer algorithm. It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves. The mergeSort() function will divide the array recursively and then merge() function will merge the left array and right array.

**Time Complexity of Merge Sort:**  
Best case : O(nlogn)  
Average case : O(nlogn)  
Worst case : O(nlogn)

**Heap Sort:**

Heap sort is a comparison-based sorting technique based on Binary Heap data structure. It is similar to selection sort where we first find the minimum element and place the minimum element at the beginning. We repeat the same process for the remaining elements. There are 2 functions used in the Heap Sort: first one is heapSort() which is main function of our algorithm and calling the heapify based on comparison and another is heapify which is converting the binary tree into Heap data structure.

**Time Complexity of Heap Sort:**  
Best case : O(nlogn)  
Average case : O(nlogn)  
Worst case : O(nlogn)

**Quick Sort:**

QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways. The quickSort() function will pick the last element as pivot and then it will arrange the elements around it in recursive manner. You can pick the element as pivot in below ways:

1. Pick first element as pivot.
2. Pick last element as pivot
3. Pick a random element as pivot.
4. Pick median as pivot.

**Time Complexity:**  
Best case : O(nlogn)  
Average case : O(nlogn)  
Worst case : O(n^2)

**Insertion Sort:**

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'insert'ed in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, insertion sort**.** The insertionSort() function is main function of our algorithm which is comparing the element and then swaping it with indexed elements.

**Time Complexity:**  
Best case: O(n)  
Average case: O(n^2)  
Worst case: O(n^2)

**Selection Sort:**

The selection sort algorithm sorts an array by repeatedly finding the minimum element from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.  
1) The subarray which is already sorted.   
2) Remaining subarray which is unsorted.  
In every iteration of selection sort, the minimum element from the unsorted subarray is picked and moved to the sorted subarray.

**Time Complexity:**  
Best case : O(N^2)  
Average case : O(N^2)  
Worst case : O(N^2)

**Bubble Sort:**

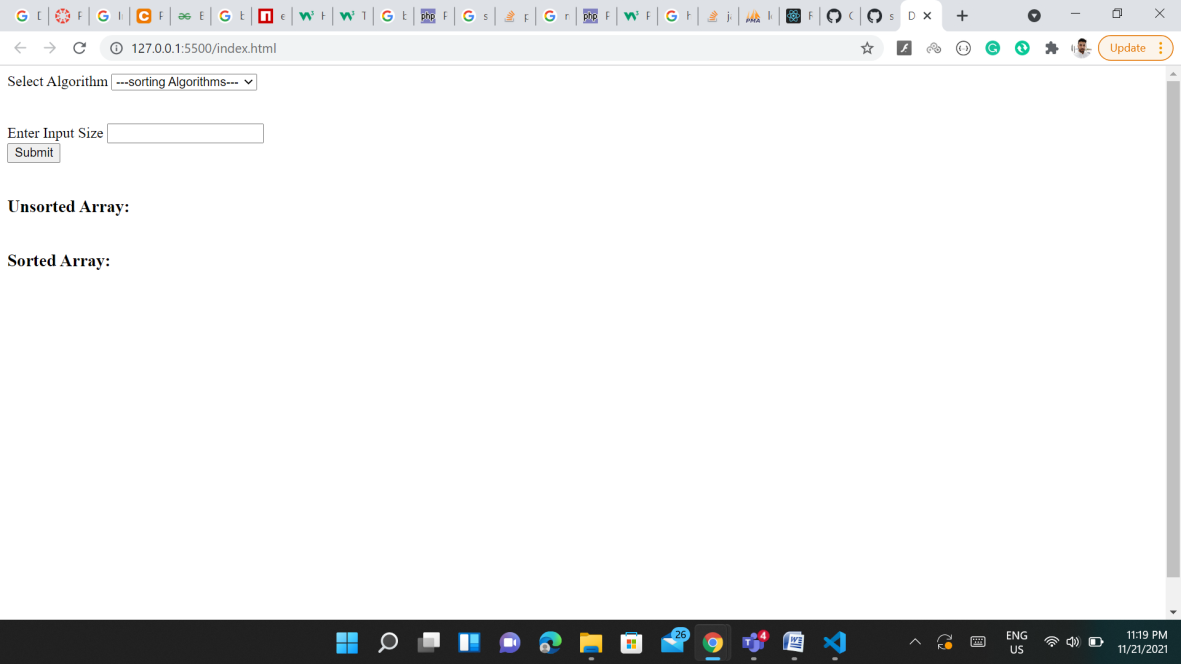
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

**Time Complexity:**  
Best case: O(n)  
Average case: O(n^2)  
Worst case: O(n^2)

**GUI:**

For Plotting the charts, <https://canvasjs.com/assets/script/canvasjs.min.js> is used. User will be able to select the sorting algorithm and input size from UI and then once he hits Submit, User will be able to see the Unsorted Array , Sorted Array and Chart will be plotted. Chart will represent Execution time of All sorting algorithms on Y-axis and algorithms on X-axis. For calculating time, Performance.now() is being used before the sorting function execution and then again it is being used after it’s execution. These Execution time is being stored in HASHMAP where key is algorithm name and value is execution time.

Before Submitting the Algorithm and Input Size from User:

****

After Submitting the Algorithm and Input Size from User: