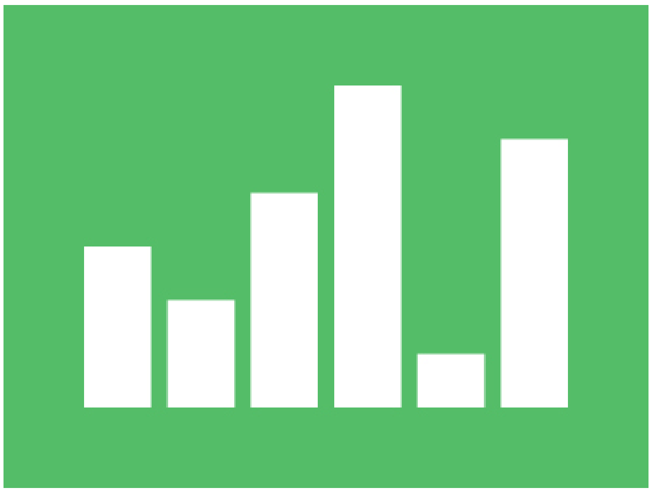


SORTING VISUALISER

- **Bubble Sort**
- **Selection Sort**
- **Insertion Sort**
- **Merge Sort**
- **Quick Sort**

ABSTRACT

Sorting algorithms are used to sort a data structure according to a specific order relationship. Visualising algorithms can be a great way to understand their functioning.

Team Member

- Sarbananda Maji
- Suryadeep Sen

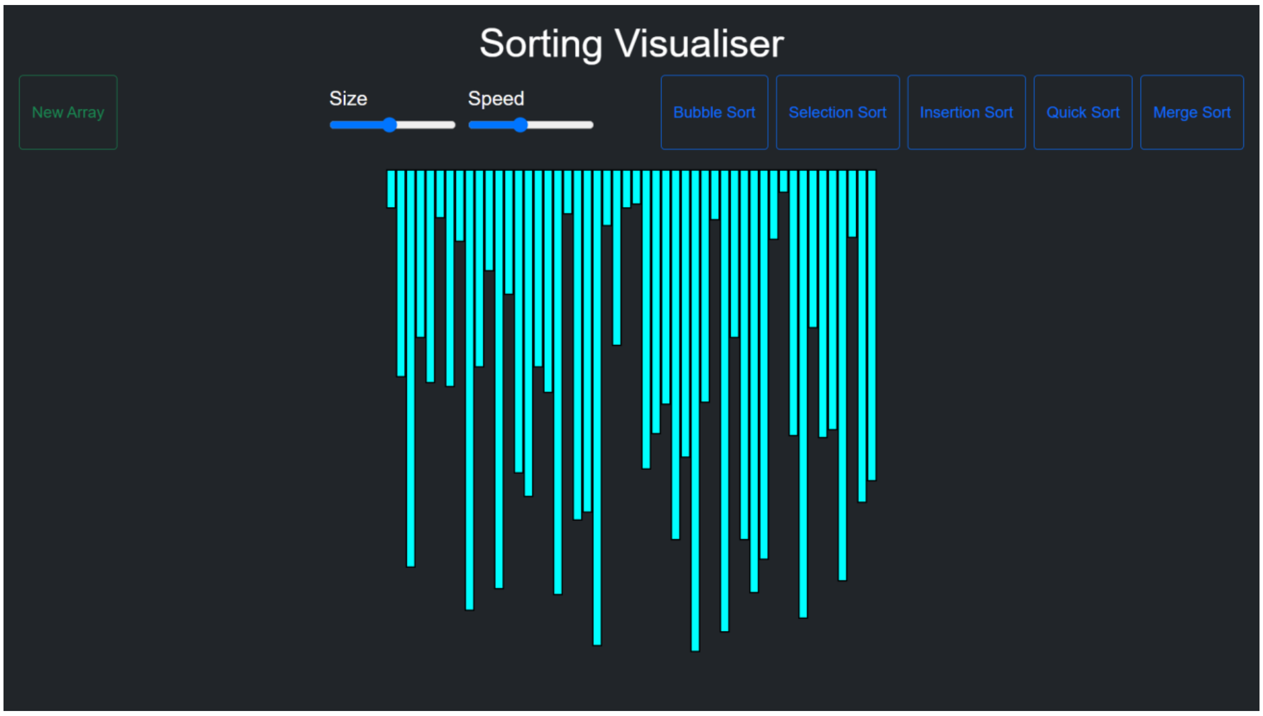
College : Asansol Engineering College
Department : Computer Science and Engineering
Year : 3rd

PROJECT DESCRIPTION :-

IN THIS SORTING VISUALIZER WE HAVE IMPLEMENTED FEW OF THE SORTING TECHNIQUES LIKE BUBBLE SORT, SELECTION SORT, INSERTION SORT, QUICK SORT AND MERGE SORT.

TO IMPLEMENT THIS SORTING ALGORITHMS WE HAVE FOCUS ON THE PARTS WHERE ACTUAL SWAPPING AND OTHER KEY FUNCTIONALITIES ARE BEING DONE SO THAT WE CAN VISUALIZE THEM PROPERLY.

WE HAVE ALSO GIVEN THE FUNCTIONALITY TO CHANGE THE SPEED OF THE ALGORITHMS ACCORDING TO US.WE CAN CHANGE THE SPEED IN BETWEEN THE SORTING SO THAT WE CAN VISUALIZE IT AND UNDERSTAND IT BETTER.MOREOVER WE CAN ALSO GENERATE NEW ARRAYS BY USING THE NEW ARRAY BUTTON.WHEN WE START TO VISUALIZE ONE SORTING ALGORITHM THE OTHER SORTNG ALGORITHMS ARE DISABLED.



WHAT IS A SORTING ALGORITHM VISUALIZATION ?

A Sorting Algorithm is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

Visualization is the graphical representation of information. Information may be data, process or concepts. The main aim of visualization is visualize and communicate information clearly and effectively.

SORTING ALGORITHMS

1. BUBBLE SORT

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

Example:

First Pass:

(**5 1** 4 2 8) –> (**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps since 5 > 1.

(1 **5 4** 2 8) –> (1 **4 5** 2 8), Swap since 5 > 4

(1 4 **5 2** 8) –> (1 4 **2 5** 8), Swap since 5 > 2

(1 4 2 **5 8**) –> (1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) –> (**1 4** 2 5 8)

(1 **4 2** 5 8) –> (1 **2 4** 5 8), Swap since 4 > 2

(1 2 **4 5** 8) –> (1 2 **4 5** 8)

(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1 2** 4 5 8) –> (**1 2** 4 5 8)

(1 **2 4** 5 8) –> (1 **2 4** 5 8)

(1 2 **4 5** 8) –> (1 2 **4 5** 8)

(1 2 4 **5 8**) –> (1 2 4 **5 8**)

Worst and Average Case Time Complexity: O(n*n). Worst case occurs when array is reverse sorted.

Best Case Time Complexity: O(n). Best case occurs when array is already sorted.

Auxiliary Space: O(1)

2. SELECTION SORT

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two sub arrays in a given array.

1) The sub array which is already sorted.
2) Remaining sub array which is unsorted.
In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted sub array is picked and moved to the sorted sub array.

Example:
Consider the array:
[10, 5, 2, 1]
The first element is 10. The next part we must find the smallest number from the remaining array. The smallest number from 5 2 and 1 is 1. So, we replace 10 by 1.
The new array is [1, 5, 2, 10] Again, this process is repeated.
Finally, we get the sorted array as [1, 2, 5, 10].
Time Complexity: $O(n^2)$ as there are two nested loops.
Auxiliary Space: O(1)
The good thing about selection sort is it never makes more than O(n) swaps and can be useful when memory write is a costly operation.

3. INSERTION SORT

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 an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Time Complexity: O(n^2)
Auxiliary Space: O(1)

Uses: Insertion sort is used when number of elements is small. It can also be useful when input array is almost sorted, only few elements are misplaced in complete big array.

4. QUICK SORT

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot.

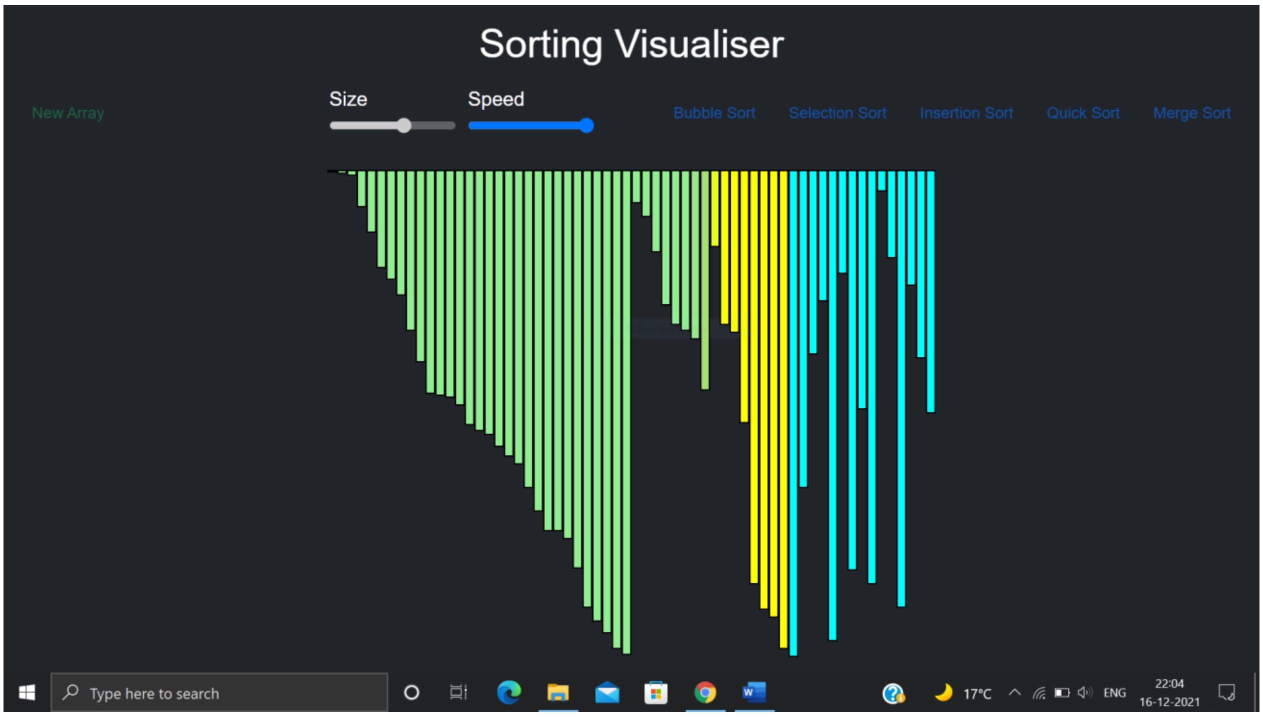
There are many different versions of quick Sort that pick pivot in different ways.

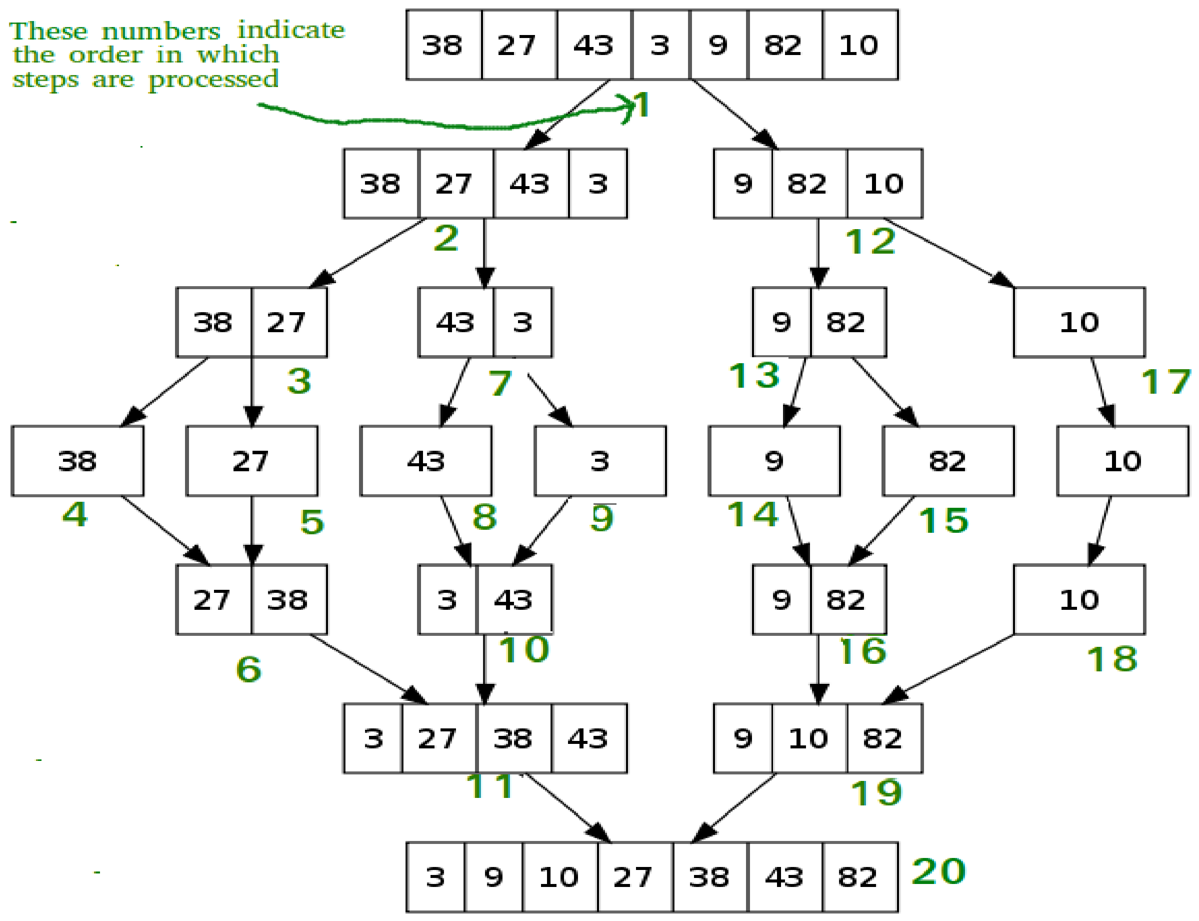
1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

The key process in quick Sort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x. All this should be done in linear time.

5. 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 merge() function is used for merging two halves. The merge(arr, l, m, r) is a key process that assumes that arr[l..m] and arr[m+1..r] are sorted and merges the two sorted sub-arrays into one.





TIME COMPLEXITY

BEST CASE	O(N*LOGN)
AVERAGE CASE	O(N*LOGN)
WORST CASE	O(N*LOGN)

FUTURE ADDITION :-

- TO ADD OTHER SORTING TECHNIQUES FOR VISUALIZATION.
- TO ADD THE TIME COMPLEXITY BUTTON.

PURPOSE OF THE PROJECT :-

It is a basic DSA sorting project where we can visualize different sorting techniques and the basic working of these sorting algorithms. Visualization can motivate the student to learn and understand easily.

CONCLUSION :-

Sorting algorithm can be difficult to understand and it's easy to be confused. We believe visualising sorting algorithms can be a great way to better understand their functioning while having fun!.