Samar Ibrahim Antar
Divide And Conquer Count Inversions

- Count the inversions in the given array. Two elements a[i] and a[j] form an inversion if a[i] > a[j] and i < j

- Sample Test Case:
  A: [30, 20, 40, 5, 90, 80, 10]
  Inversions: 10
  Explanation: (30,20), (40,5), (30,5), (20,5), (90,80), (90,10), (80,10),(40,10), (30,10), (20,10)

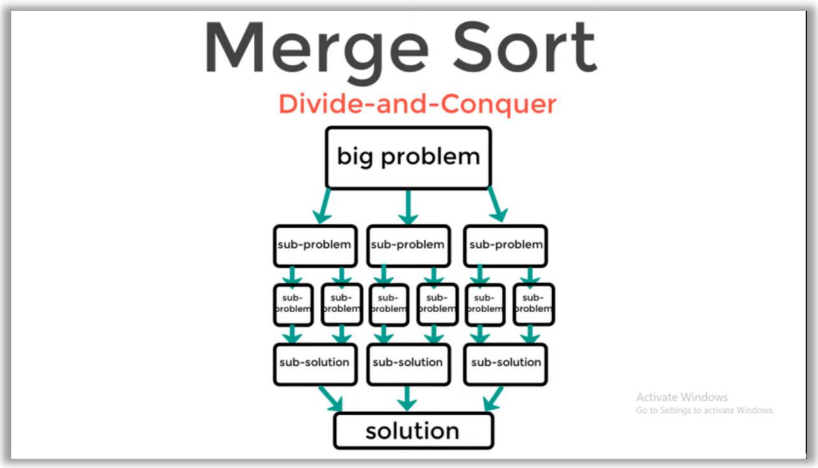
- **Using Naïve Approach:**

✓
```
 int inv = 0;
for(int i = 0; i < n; i++)
{
   for(int j = i + 1; j < n; j++){
       if(a[i] > a[j]) inv++;
    }
 }
 cout<<inv;
```

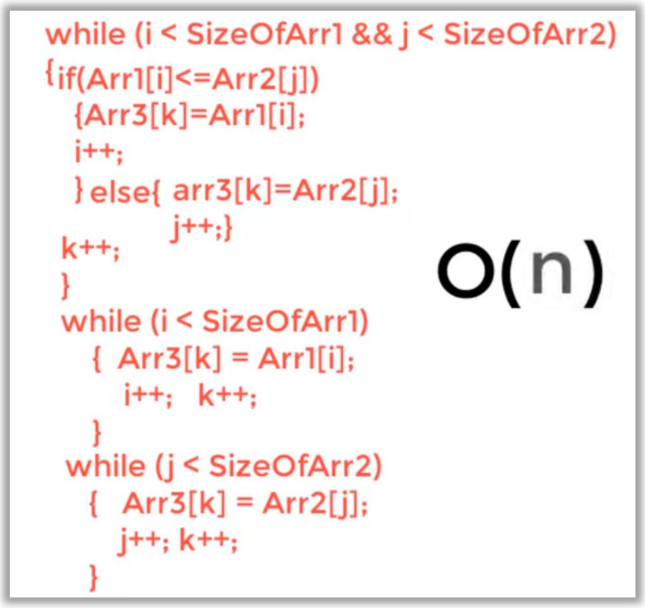
✓ So, Time Complexity: O(N^2)

- **Using Divide and Conquer (Merge Sort):**

✓ Idea >> Divide the array into two parts, get the inversions for the left part and get the inversions for the right part recursively and merge the two arrays, and get their inversions.



✓ pseudo code for merge function: n1 >> size of array1(left) & n2 >> size of array2(right)
So, Complexity: O(n1+n2) >> O(n)

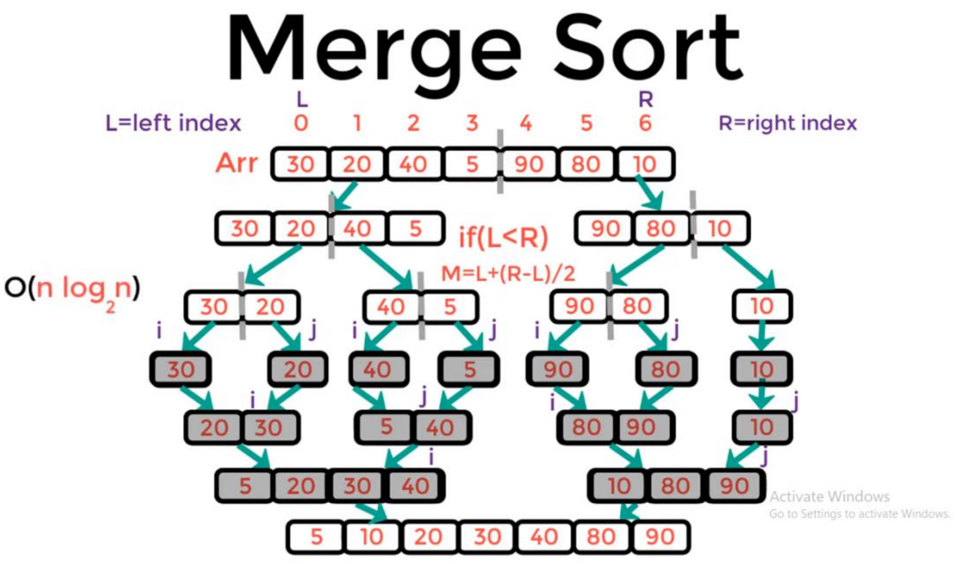


```
while (i < SizeOfArr1 && j < SizeOfArr2)
{if(Arr1[i]<=Arr2[j])
   {Arr3[k]=Arr1[i];
   i++;
   }else{ arr3[k]=Arr2[j];
          j++;}
  k++;
  }
  while (i < SizeOfArr1)
    { Arr3[k] = Arr1[i];
      i++;   k++;
    }
  while (j < SizeOfArr2)
    {  Arr3[k] = Arr2[j];
      j++; k++;
    }
```

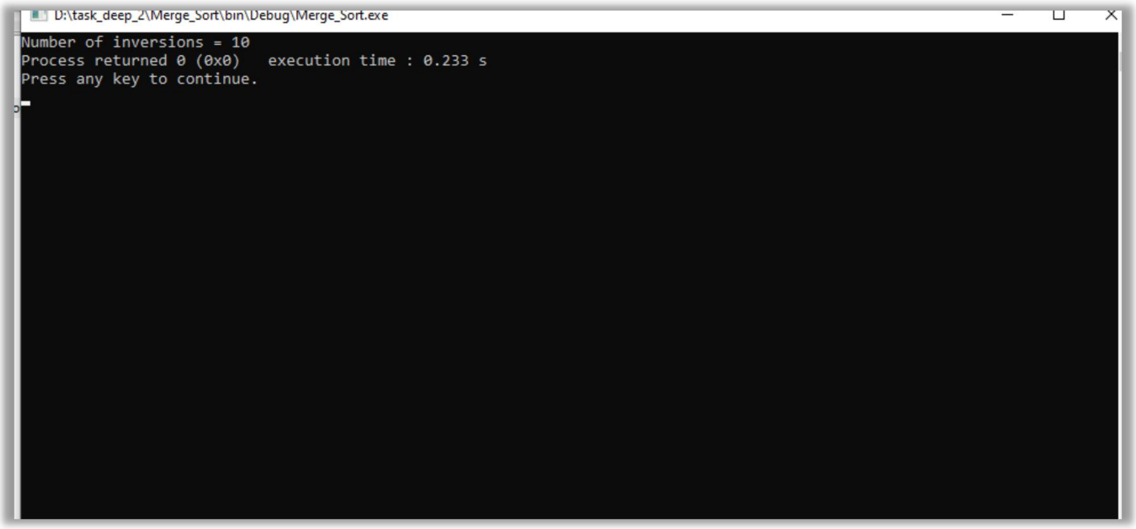
O(n)

✓ Merge sort function recursive:
-divide the original array to sub-arrays
-sort them then call merge function to merge these sub-arrays until reaching to final sorted array

✓ So, total Time Complexity: O(n $\log_2$ n)



- Output of merge_sort.cpp: given array[30, 20, 40, 5, 90, 80, 10]



```
Number of inversions = 10
Process returned 0 (0x0)   execution time : 0.233 s
Press any key to continue.
```