Counting Sort:

Counting sort is a sorting technique that is based on the keys between specific ranges. This sorting technique doesn't perform sorting by comparing elements. It performs sorting by counting objects having distinct key values like hashing. After that, it performs some arithmetic operations to calculate each object's index position in the output sequence. Counting sort is not used as a general-purpose sorting algorithm.

Counting sort is effective when range is not greater than number of objects to be sorted.

Code->

```
public class CountingSort {

int getMax(int[] a, int n) {
  int max = a[0];
  for(int i = 1; i<n; i++) {
      if(a[i] > max)
         max = a[i];
  }
  return max;
}
void countSort(int[] a, int n)
 {
   int[] output = new int [n+1];
   int max = getMax(a, n);
   //int max = 42;
   int[] count = new int [max+1];

  for (int i = 0; i <= max; ++i)
  {
    count[i] = 0;
```

```
  }

  for (int i = 0; i < n; i++)
  {
   count[a[i]]++;
  }

   for(int i = 1; i<=max; i++)
      count[i] += count[i-1];
  for (int i = n - 1; i >= 0; i--) {
   output[count[a[i]] - 1] = a[i];
   count[a[i]]--;
}

   for(int i = 0; i<n; i++) {
      a[i] = output[i];
   }
}


void printArray(int a[], int n)
{
   int i;
   for (i = 0; i < n; i++)
       System.out.print(a[i] + " ");
}

public static void main(String args[])
{
   int a[] = { 11, 30, 24, 7, 31, 16, 39, 41 };
   int n = a.length;
   CountingSort c1 = new CountingSort();
   System.out.println("\nBefore sorting array elements are -
   "); c1.printArray(a, n);
```

```
    c1.countSort(a,n);
    System.out.println("\nAfter sorting array elements are - ");
    c1.printArray(a, n);
    System.out.println();
}
}
```

Output->
Before sorting array elements are -
11 30 24 7 31 16 39 41
After sorting array elements are -
7 11 16 24 30 31 39 41

Screenshot->

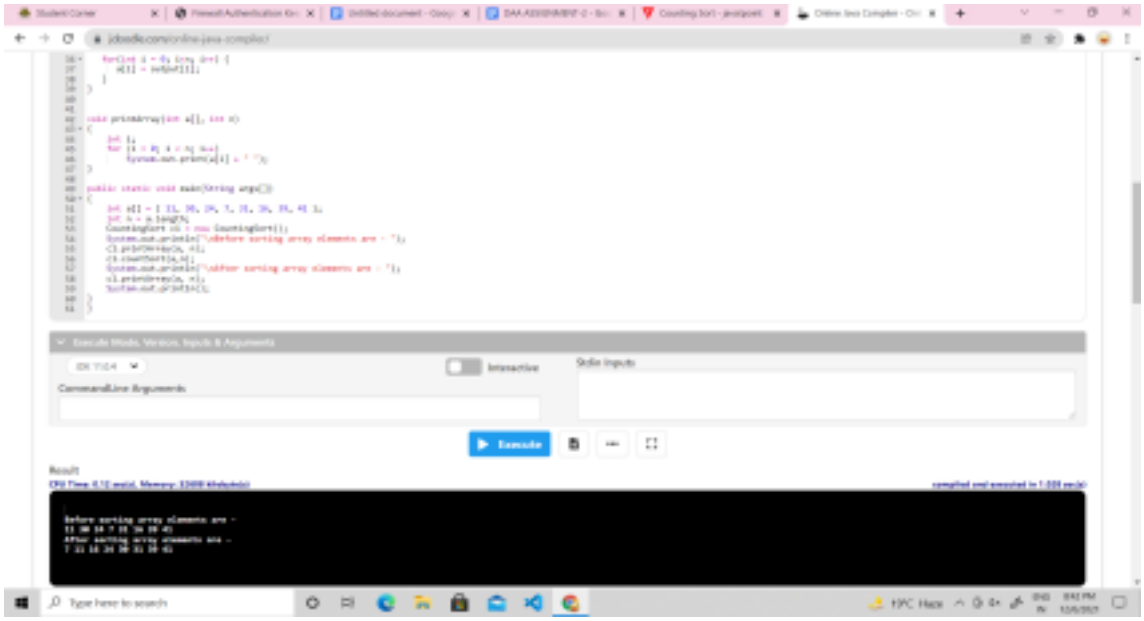

Figure. 1

Best Case Complexity ->

It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of counting sort is **O(n + k)**.

Average Case Complexity ->

It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of counting sort is **O(n + k)**.

Worst Case Complexity ->

It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of counting sort is **O(n + k)**.