Objective: Implementation of Counting Sort

Counting Sort

Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array. The count is stored in an auxiliary array and the sorting is done by mapping the count as an index of the auxiliary array.

Counting Sort Algorithm:

```
CountingSort(array, size)

  max <- find largest element in array

  initialize count array with all zeros

  for j <- 0 to size

    find the total count of each unique element and

    store the count at jth index in count array

  for i <- 1 to max

    find the cumulative sum and store it in count array itself

  for j <- size down to 1

    restore the elements to array

    decrease count of each element restored by 1
```

How Counting Sort Works?

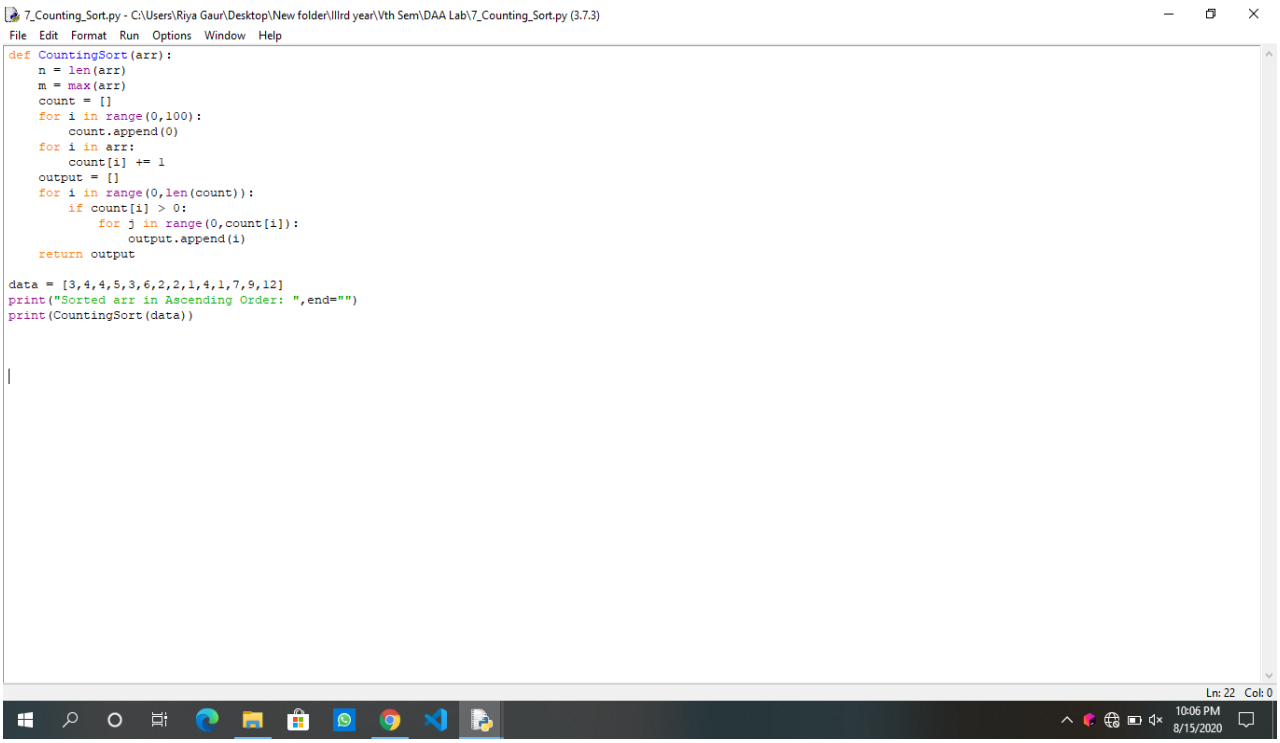
1. Find out the maximum element (let it be `max`) from the given array.
2. Initialize an array of length `max+1` with all elements 0. This array is used for storing the count of the elements in the array.

3. Store the count of each element at their respective index in `count` array.
4. Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.
5. Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated.
6. After placing each element at its correct position, decrease its count by one.

Code:

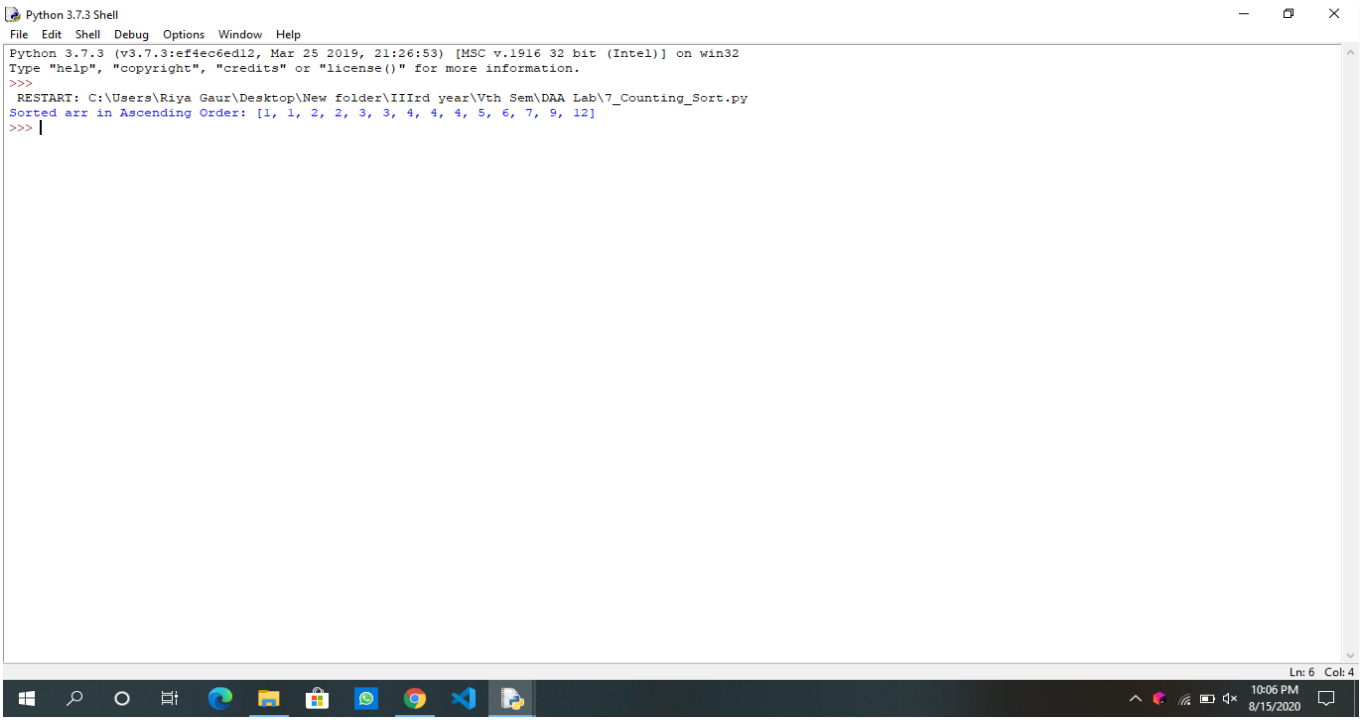
```
def CountingSort(arr):
    n = len(arr)
    m = max(arr)
    count = []
    for i in range(0,100):
        count.append(0)
    for i in arr:
        count[i] += 1
    output = []
    for i in range(0,len(count)):
        if count[i] > 0:
            for j in range(0,count[i]):
                output.append(i)
    return output

data = [3,4,4,5,3,6,2,2,1,4,1,7,9,12]
print("Sorted arr in Ascending Order: ",end="")
print(CountingSort(data))
```



Output:

Sorted arr in Ascending Order: [1, 1, 2, 2, 3, 3, 4, 4, 4, 5, 6, 7, 9, 12]



Time Complexities:

There are mainly four main loops. (Finding the greatest value can be done outside the function.)

for-loop	time of counting
1st	O(max)
2nd	O(size)
3rd	O(max)
4th	O(size)

Overall complexity = `O(max)+O(size)+O(max)+O(size)` = `O(max+size)`

- **Worst Case Complexity:** `O(n+k)`
- **Best Case Complexity:** `O(n+k)`
- **Average Case Complexity:** `O(n+k)`

Counting Sort Applications:

Counting sort is used when:

- there are smaller integers with multiple counts.
- linear complexity is the need.