Assignment
Analysis of Algorithms
(Count Sort)

Group Members:

ZIA UR REHMAN (1802034)

SAIF ULLAH KHAN (1802012)

WISHA KHURSHEED (1802007)

7-March-2020

Count Sort:

Till now, all the sorting algorithms we have learned were comparison sort i.e., they compare the values of the elements to sort them but the counting sort is a non-comparison sort. It means that it sorts the input without comparing the values of the elements.

Because by comparing elements of array for sorting will takes at least **Big-O(***nlog (n)***)** number of steps **(**Merge sort**)**. But by **Count Sort** approach we can sort the entire array in **Big-O(n)** complexity without doing any comparison.

Design:

Base Step: In the very first step find the max number in unsorted array and initialized empty array of that size.

Step1: Count the occurrence of each element and store the counts in a separate array at the index specified by the element i.e. if we get 2 then the number at index 2 (initially 0) is incremented by 1 and so on.

Step2: After counting the elements occurrence, now compute the running sum. Like if **count** [0, 1, 0, 3, 0, 0, 1] then **RunSum** [0,0,1,1,4,4,4,5]. Place "0" at the first index of running sum then add the index[0] of both arrays and store the result at index[1] , then again add index[1] of both arrays and add them store the result at index[2] and so on.
Note: you can also store the result in "count" array .

Step3: Now pick the first number from unsorted array and using that element as an index jump to the RunSum array. Now the Using the Value placed at that index in RunSum array, as an index jump to the New array, store the value and increment the index value at RunSum array by 1 and so on.

Pseudo code:

Def CountSort(array)

MAX = max(array)

CountArray = [0 range(from 0 to MAX)]

#Loop Start

forloop →0 to length(array)

CountSort[array[x]]++

#Now Compute the running sum in the same array (CountArray)

Temp = CountArray[0]

CountArray[0] = 0

forloop→ 0 to length(array)-1

next = CountArray[x+1]

CountArray[x+1] = CountArray[x] + y

Y = next

Repeat;

#This will store the running sum in the same array

#Now creating the Sorted array

Forloop→0 to length (array)

Index = array[x]

NextIndex = RunSum[index]

RunSum[index]++

SortedArray[NextIndex] = array[x]

#This will give the Sorted array

Implementation:

```
def CountSort(arr):

    #Find the Max Number
    mx = max(arr) + 1

    #initilized Count array of size mx
    Count = [0 for i in range(0, (mx))]

    for x in range(0, len(arr)):
        Count[arr[x]]+= 1
```

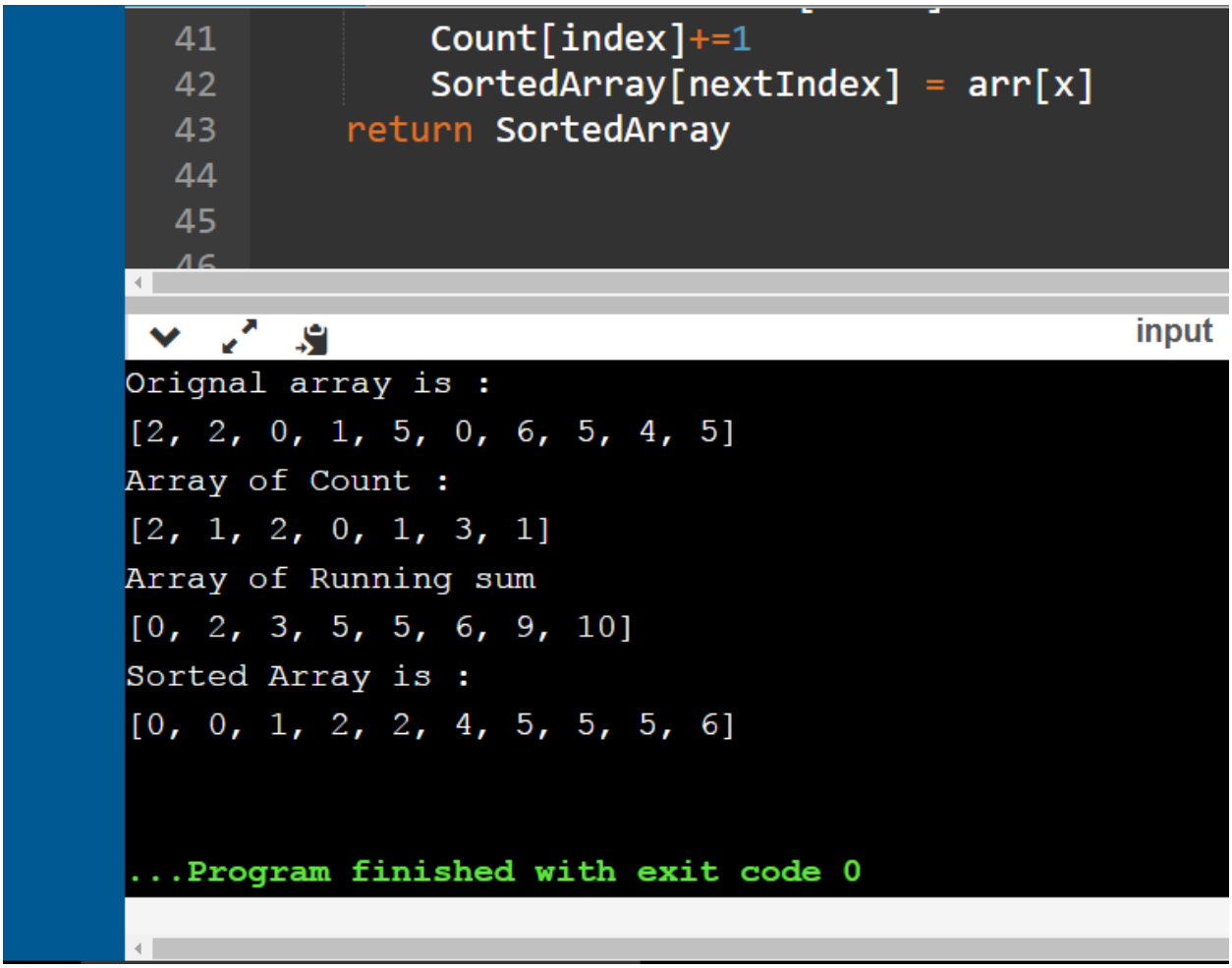
```
    #Count array is generated
print("Array of Count : ")
print(Count)
Count.append(0)


# Now we are computing the Running Sum


temp = Count[0]
Count[0] = 0
end = len(Count) - 1


x = 0
y = 0
for i in range(0, end):
    y = Count[x+1]
    Count[x+1] = Count[x] + temp
    temp = y
    x+=1
print("Array of Running sum ")
print(Count)
```

```
    #Running Sum has been calculated
    #We Use A single Array to reduce the space complexity


    SortedArray = [0 for i in range(0, len(arr))]


    for x in range(0, len(arr)):
        #Get the first element and use it as index for RunSum array
        nextIndex = Count[arr[x]]
        Count[arr[x]]+=1
        SortedArray[nextIndex] = arr[x]
    return SortedArray



#Driver Code
print("Orignal array is : ")
arr = [2,2,0,1,5,0,6,5,4,5]
print(arr)
SortedArray = CountSort(arr)
print("Sorted Array is : ")
print(SortedArray)
```

CODE PICTURE:

```
def CountSort(arr):

    #Find the Max Number
    mx = max(arr) + 1

    #initilized Count array of size mx
    Count = [0 for i in range(0, (mx))]

    for x in range(0, len(arr)):
        Count[arr[x]]+= 1
        #Count array is generated
    print("Array of Count : ")
    print(Count)
    Count.append(0)

    # Now we are computing the Running Sum

    temp = Count[0]
    Count[0] = 0
    end = len(Count) - 1

    x = 0
    y = 0
    for i in range(0, end):
        y = Count[x+1]
        Count[x+1] = Count[x] + temp
        temp = y
        x+=1
    print("Array of Running sum ")
    print(Count)

    #Running Sum has been calculated
    #We Use A single Array to reduce the space complexity

    SortedArray = [0 for i in range(0, len(arr))]

    for x in range(0, len(arr)):
        #Get the first element and use it as index for RunSum array
        nextIndex = Count[arr[x]]
        Count[arr[x]]+=1
        SortedArray[nextIndex] = arr[x]
    return SortedArray
```

Result:



Analysis & Complexity of Count Sort:

In this algorithm of Count Sort, we didn't do even a single comparison and we sort the input array in **Big-O(k+n)** complexity.

The analysis of the counting sort is simple. For the first for loop i.e., to initialize the temporary array, we are iterating from 0 to k, so its running time is Θ(k)Θ(k).

The next loop is running from 1 to A. length and thus has a running time of Θ(n)Θ(n). The next for loop is again iterating over the temporary array from 1 to k and thus has a running time of Θ(k)Θ(k). The last loop is again Θ(n)Θ(n).

1. The loop takes $O(k)$ time
2. The loop takes $O(n)$ time
3. The loop takes $O(k)$ time
4. The loop takes $O(n)$ time

Therefore, the overall time of the counting sort is $O(k) + O(n) + O(k) + O(n) = O(k + n)$

In practice, we usually use counting sort algorithm when have $k = O(n)$, in which case running time is $O(n)$.

So final Complexity is **T = Big-O(n)**

So from the above calculations, we conclude that the overall time complexity of **Count Sort** algorithm is Big-O (n).Because there is no nested loop. It only contain 4 loops.

The Counting sort is a stable sort i.e., multiple keys with the same value are placed in the sorted array in the same order that they appear in the input array.

Note that Counting sort beats the lower bound of $\Omega(n \log n)$, because it is not a comparison sort. There is no comparison between elements. Counting sort uses the actual values of the elements to index into an array.

Worst-Case Complexity:

The Worst-Case complexity of Count Sort is Big-O(k+n) irrespective of input. Because we are not comparing the elements but only applying some process on array every time .

The initializing of array, counting the occurrence, producing the Running sum and generating the Sorted array takes same number of steps every times. It doesn't depends on the input.

Best-Case Complexity:

The best-case complexity of Count Sort is same as the Worst-case complexity.