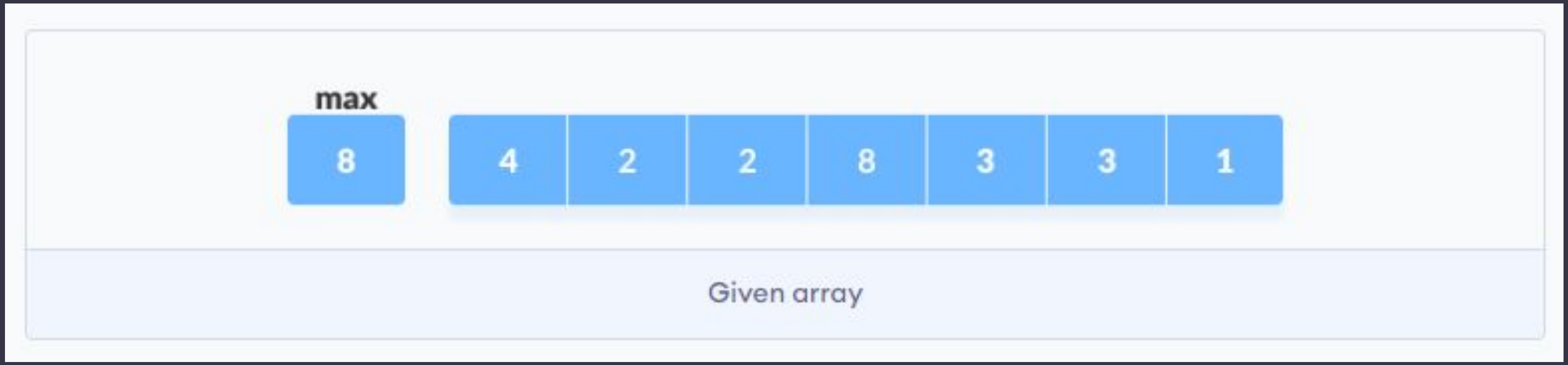
COUNTING SORT

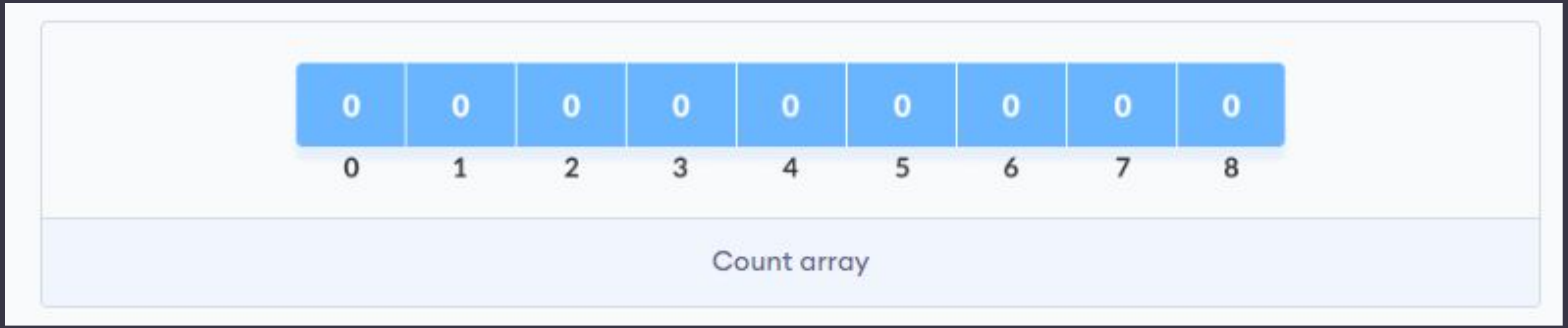
- Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array. The count is stored in an auxiliary array and the sorting is done by mapping the count as an index of the auxiliary array.

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 elements in the array.

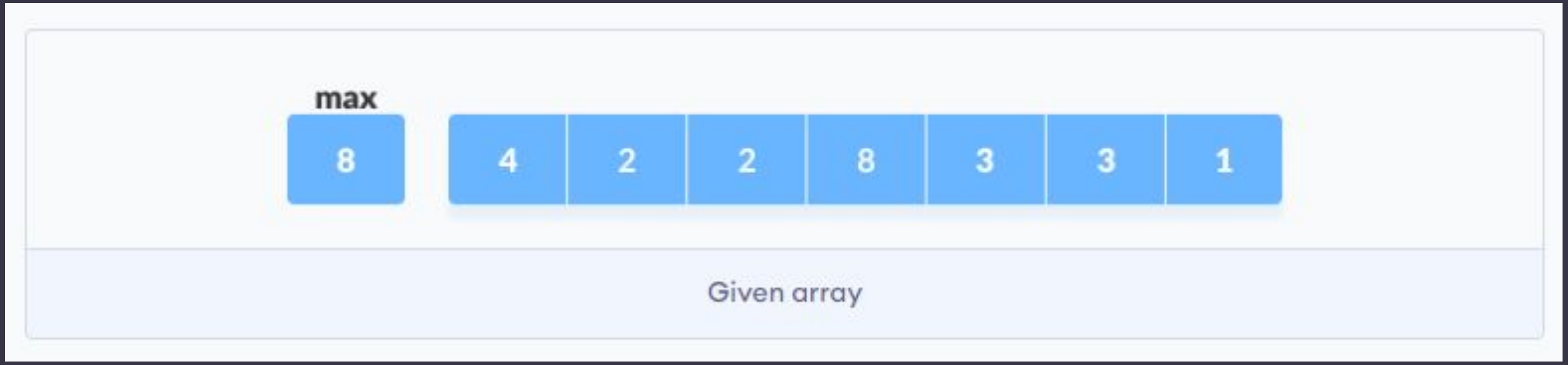


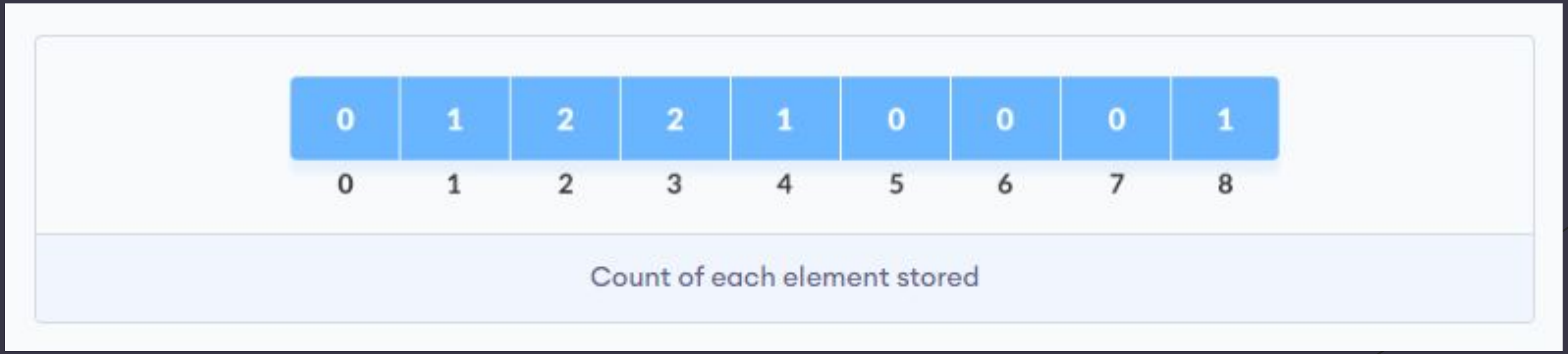
Count array

3. Store the count of each element at their respective index in *count* array.

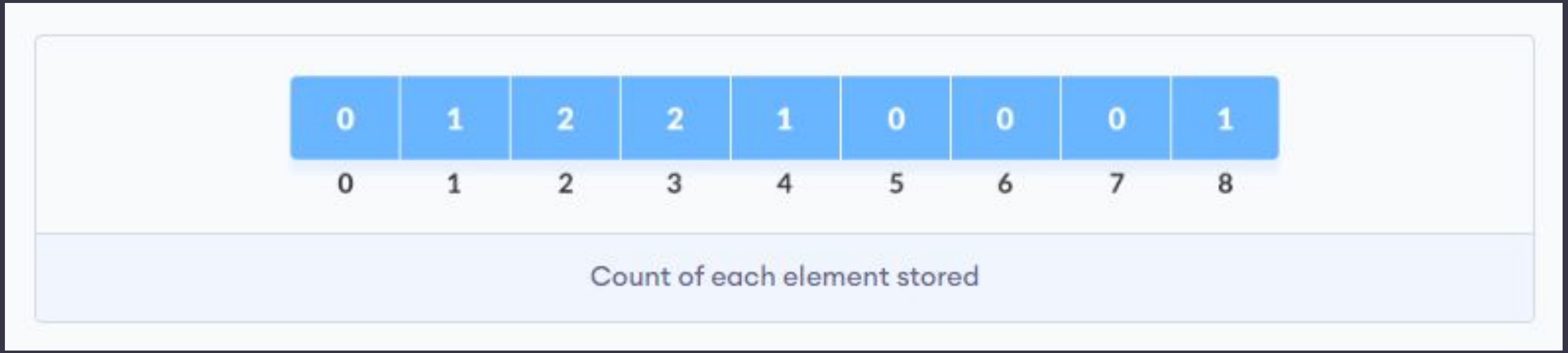
For example:

If the count of element “3” is 2 then, 2 is stored in the 3rd position of *count* array. If element “5” is not present in the array, then 0 is stored in the 5th position.

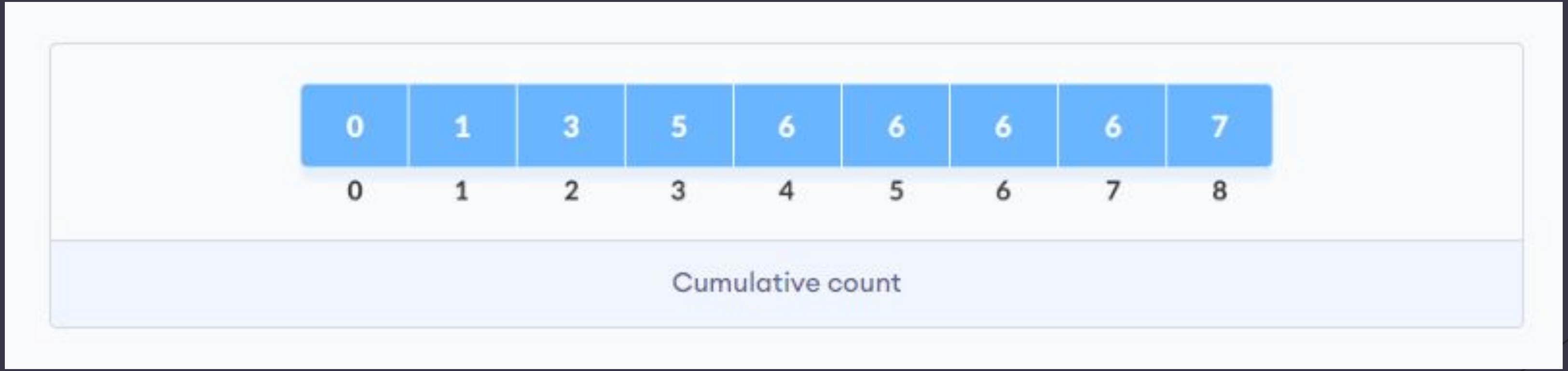




4. Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.

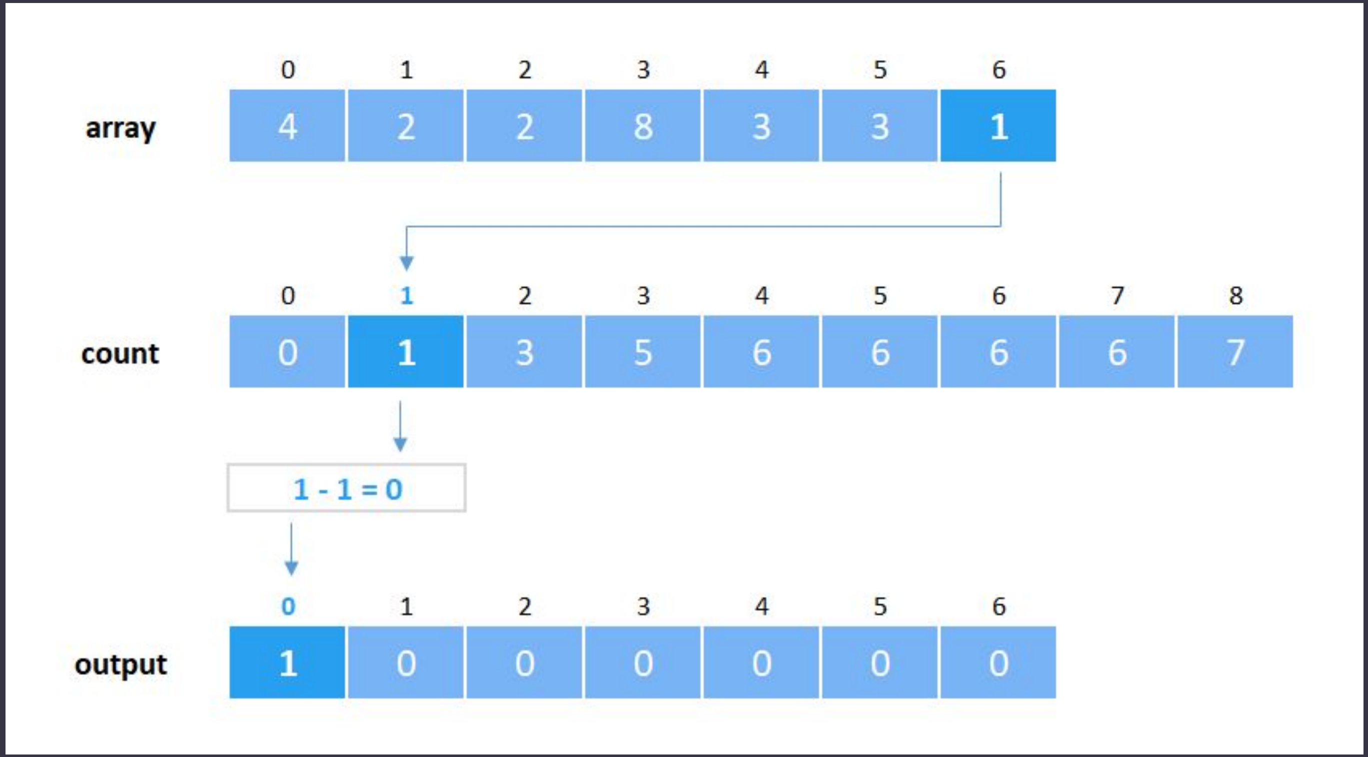


Count of each element stored

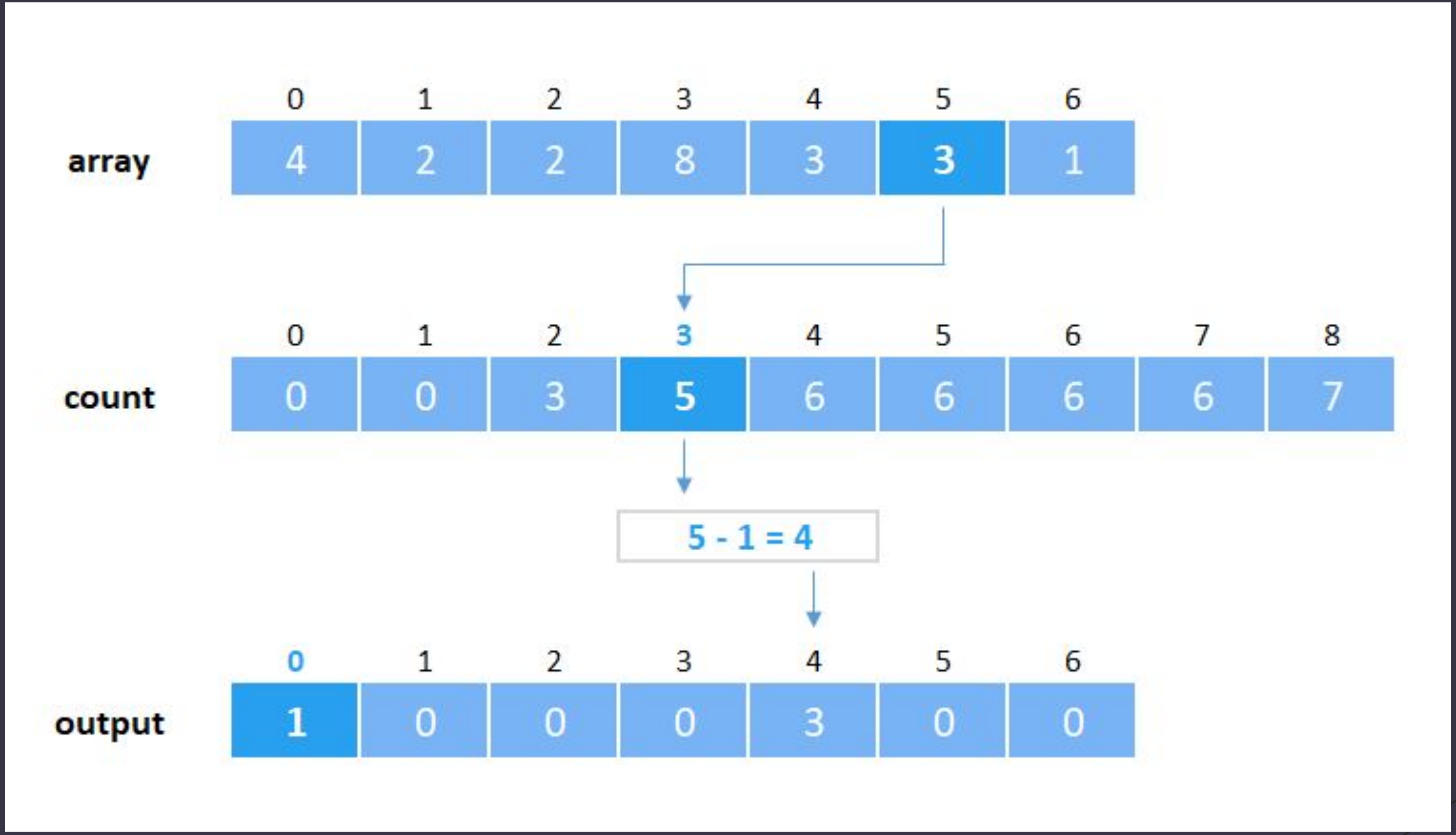


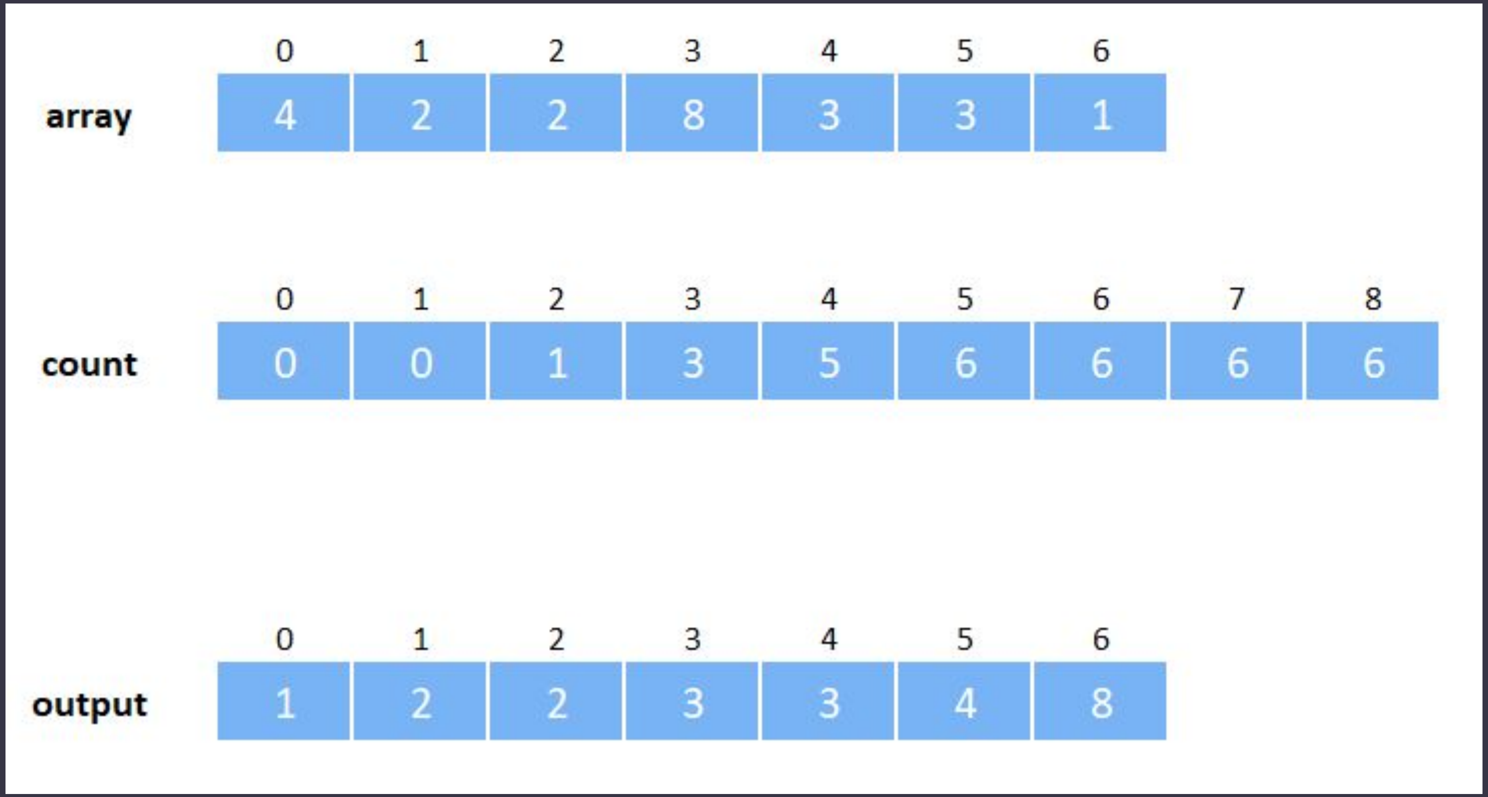
Cumulative count

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 as shown in the figure below.



6. Before placing each element at its correct position, decrease its count by one.



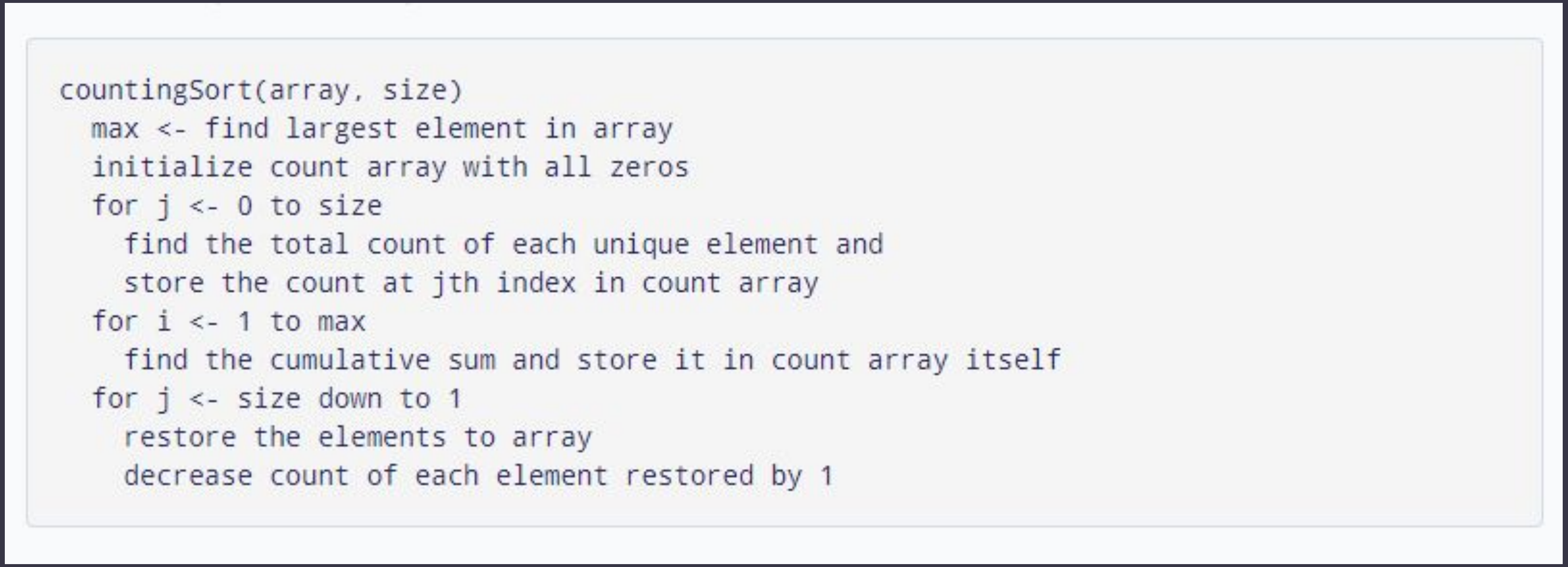


	0	1	2	3	4	5	6
array	4	2	2	8	3	3	1

	0	1	2	3	4	5	6	7	8
count	0	0	1	3	5	6	6	6	6

	0	1	2	3	4	5	6
output	1	2	2	3	3	4	8

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
```

Complexity

- Worst Case Complexity: 0(n + k)
- Best Case Complexity: 0(n + k)
- Average Case Complexity: 0(n + k)

n = max, k = size

In all the above cases, the complexity is the same because no matter how the elements are placed in the array, the algorithm goes through n+k times.

Exercise

2	5	6	2	3	10	3	6	7	8