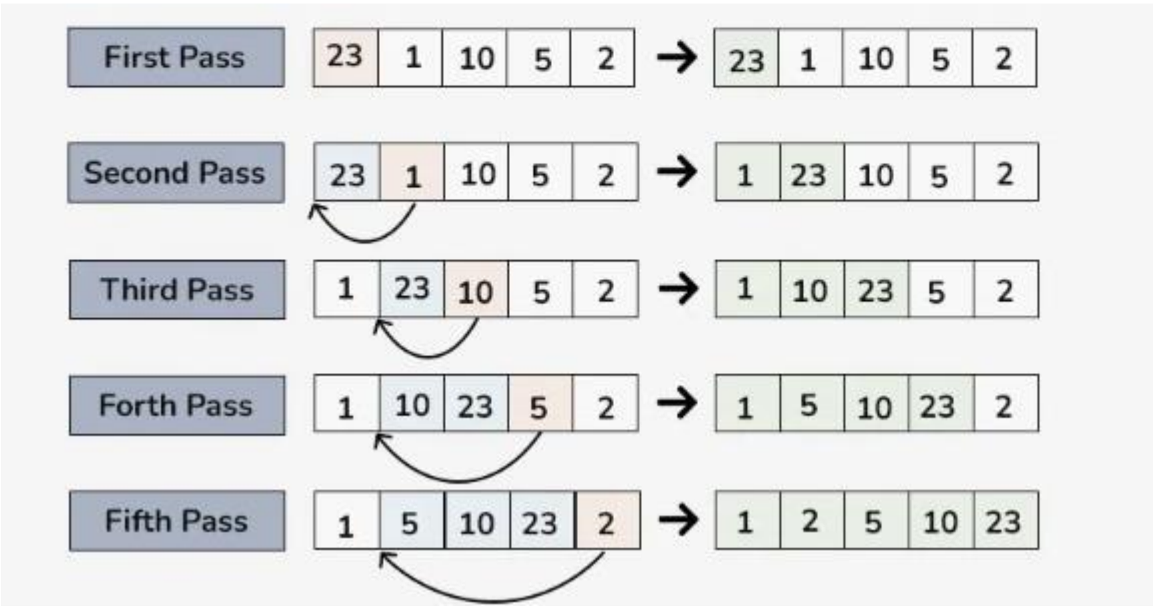
Theory:

1) Insertion Sort

- Insertion sort is a simple sorting algorithm that works by iteratively inserting each element of an unsorted list into its correct position in a sorted portion of the list.
- It is a stable sorting algorithm, meaning that elements with equal values maintain their relative order in the sorted output.
- Insertion sort is a simple sorting algorithm that works by building a sorted array one element at a time. It is considered an ” in-place ” sorting algorithm, meaning it doesn’t require any additional memory space beyond the original array.

<u>For Example :</u>

Consider an array having elements **: {23, 1, 10, 5, 2}**



First Pass:

- Current element is **23**
- The first element in the array is assumed to be sorted.
- The sorted part until **0th** index is : **[23]**

Second Pass:

- Compare **1** with **23** (current element with the sorted part).

- Since **1** is smaller, insert **1** before **23** .
- The sorted part until **1st** index is: **[1, 23]**

Third Pass:

- Compare **10** with **1** and **23** (current element with the sorted part).
- Since **10** is greater than **1** and smaller than **23** , insert **10** between **1** and **23** .
- The sorted part until **2nd** index is: **[1, 10, 23]**

Fourth Pass:

- Compare **5** with **1** , **10** , and **23** (current element with the sorted part).
- Since **5** is greater than **1** and smaller than **10** , insert **5** between **1** and **10** .
- The sorted part until **3rd** index is **: [1, 5, 10, 23]**

Fifth Pass:

- Compare **2** with **1, 5, 10** , and **23** (current element with the sorted part).
- Since **2** is greater than **1** and smaller than **5** insert **2** between **1** and **5** .
- The sorted part until **4th** index is: **[1, 2, 5, 10, 23]**

Final Array:

- The sorted array is: **[1, 2, 5, 10, 23]**

Algorithm:

Programming Language:

Code:

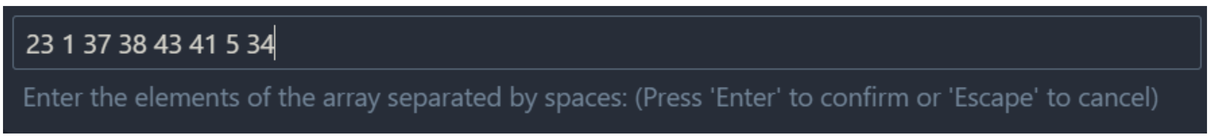
```
def insertionSort(arr):
    n = len(arr)

    if n <= 1:
        return

    for i in range(1, n):
        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        key = arr[i]
        j = i - 1
        while j >= 0 and key < arr[j]:
            arr[j + 1] = arr[j]
            j -= 1
        arr[j + 1] = key

# User input for the array
arr = list(map(int, input("Enter the elements of the array separated by spaces:
").split()))
insertionSort(arr)
print("Sorted array:", arr)
```

Output:



```
[2]    ✓  46.8s

...   Sorted array: [1, 5, 23, 34, 37, 38, 41, 43]
```

Space complexity: __________

Justification:____
____

Time complexity:

Best case time complexity: __________

Justification:____
____

Worst case time complexity: __________

Justification:____
____

Theory:

2) Bubble Sort

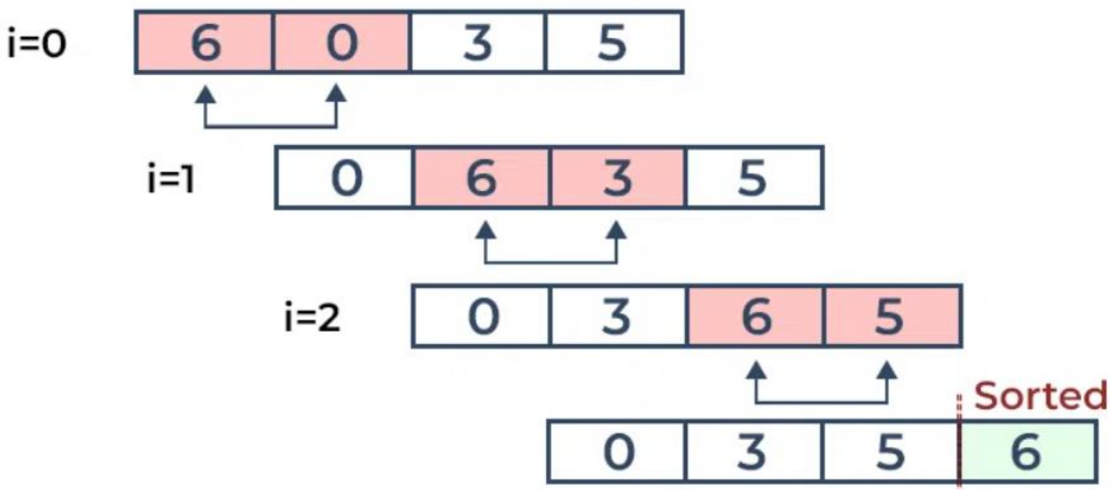
Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in the wrong order. This algorithm is not suitable for large data sets as its average and worst-case time complexity is quite high.

For example:

Input: arr[] = {6, 0, 3, 5}

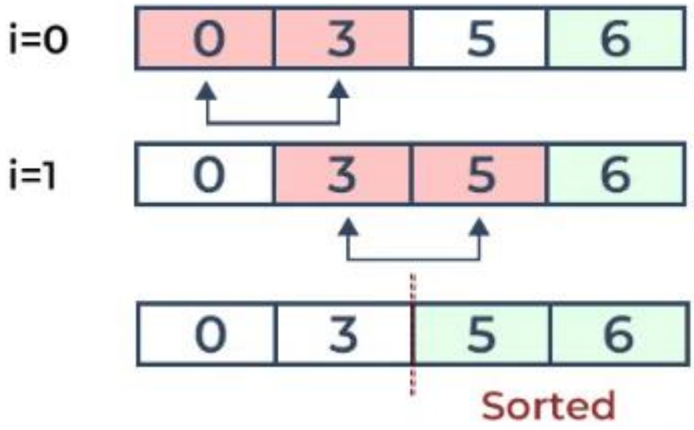
First Pass:

The largest element is placed in its correct position, i.e., the end of the array.



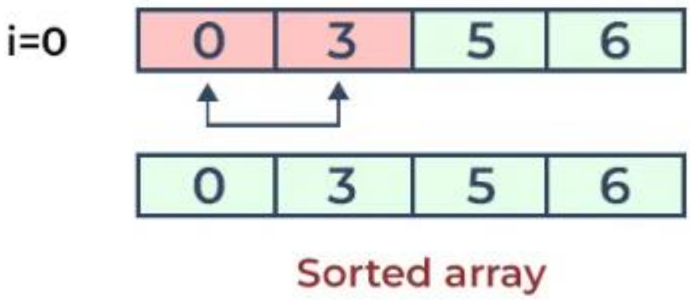
Second Pass:

Place the second largest element at correct position



Third Pass:

Place the remaining two elements at their correct positions.



Algorithm:

Programming Language:

Code:

```
def bubblesort(elements):
    size = len(elements)

    for i in range(size-1):
        swapped = False
        for j in range(size-1-i):
            if elements[j] > elements[j+1]:
```

```
        tmp = elements[j]
        elements[j] = elements[j+1]
        elements[j+1] = tmp
        swapped = True
    if not swapped:
        break

arr = list(map(int, input("Enter the elements of the array separated by spaces: ").split()))
bubblesort(arr)
print("Sorted array:", arr)
```

```
22 5 12 25 76
Enter the elements of the array separated by spaces: (Press 'Enter' to confirm or 'Escape' to cancel)
```

```
✓ 16.3s

Sorted array: [5, 12, 22, 25, 76]
```

Space complexity: ___________

Justification:____

____

Time complexity:

Best case time complexity: __________

Justification:____
____

Worst case time complexity: __________

Justification:____
____

Theory:

3) Selection Sort

Selection sort is a simple and efficient sorting algorithm that works by repeatedly selecting the smallest (or largest) element from the unsorted portion of the list and moving it to the sorted portion of the list.

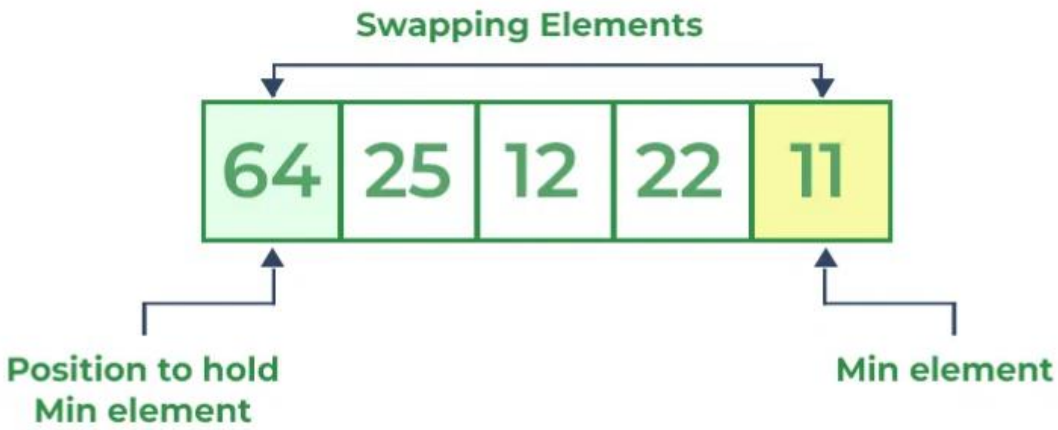
For example :

Lets consider the following array as an example: arr[] = {64, 25, 12, 22, 11}

First pass:

For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where 64 is stored presently, after traversing whole array it is clear that 11 is the lowest value.

Thus, replace 64 with 11. After one iteration 11, which happens to be the least value in the array, tends to appear in the first position of the sorted list.



Second Pass:

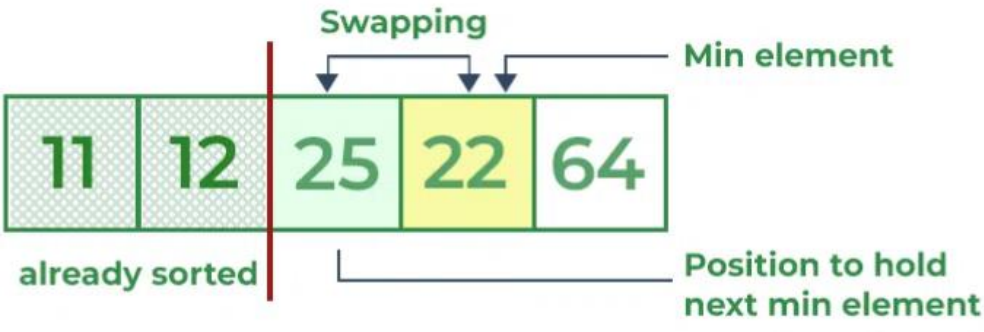
For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.

After traversing, we found that 12 is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.

Third Pass:

Now, for third place, where 25 is present again traverse the rest of the array and find the third least value present in the array.

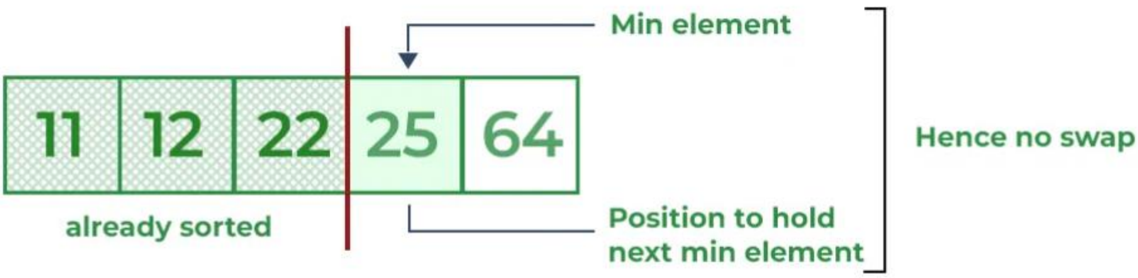
While traversing, 22 came out to be the third least value and it should appear at the third place in the array, thus swap 22 with element present at third position.



Fourth pass:

Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array

As 25 is the 4th lowest value hence, it will place at the fourth position.



Fifth Pass:

At last the largest value present in the array automatically get placed at the last position in the array

The resulted array is the sorted array.

Sorted array

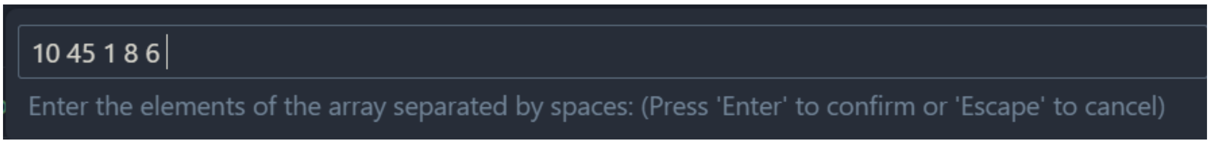
Algorithm:

Programming Language:

Code:

```
def selectionsort(arr):
    size = len(arr)
    for i in range(size-1):
        min_index = i
        for j in range(min_index+1,size):
            if arr[j] < arr[min_index]:
                min_index = j
        if i != min_index:
            arr[i],arr[min_index] = arr[min_index],arr[i]


elements = list(map(int, input("Enter the elements of the array separated by spaces: ").split()))
selectionsort(elements)
print("Sorted array:", elements)
```



```
[7]   ✓  31.6s

...   Sorted array: [1, 6, 8, 10, 45]
```

Space complexity: __________

Justification:____

____

Time complexity:

Best case time complexity: __________

Justification:____

____

Worst case time complexity: __________

Justification:____

____

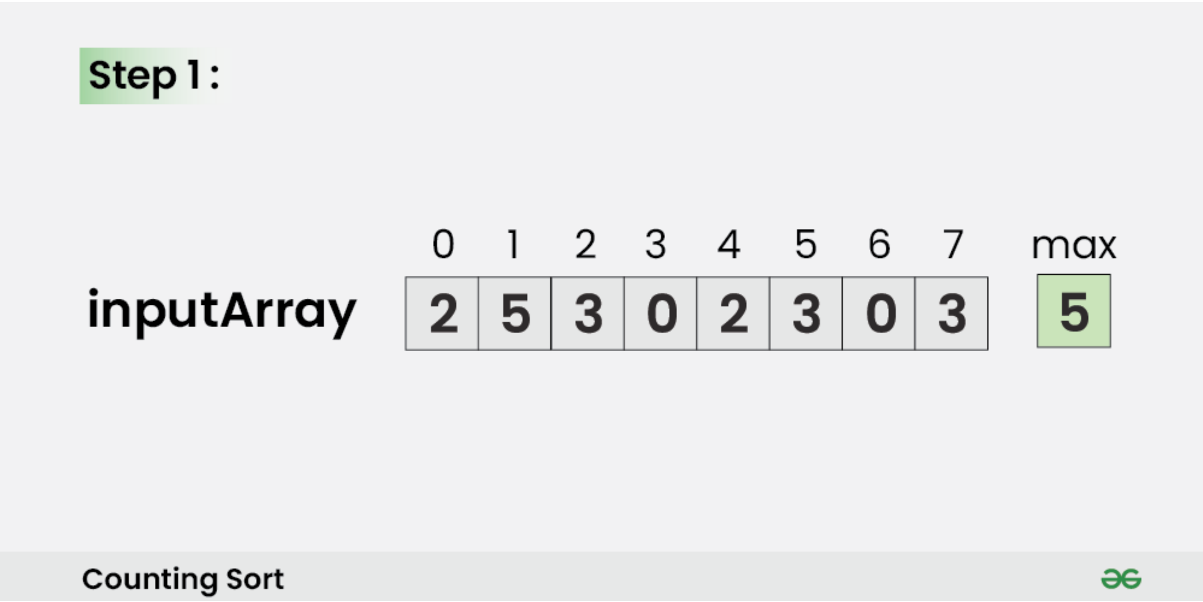
Theory:

4) Counting Sort

Counting Sort is a non-comparison-based sorting algorithm that works well when there is limited range of input values. It is particularly efficient when the range of input values is small compared to the number of elements to be sorted. The basic idea behind Counting Sort is to count the frequency of each distinct element in the input array and use that information to place the elements in their correct sorted positions.

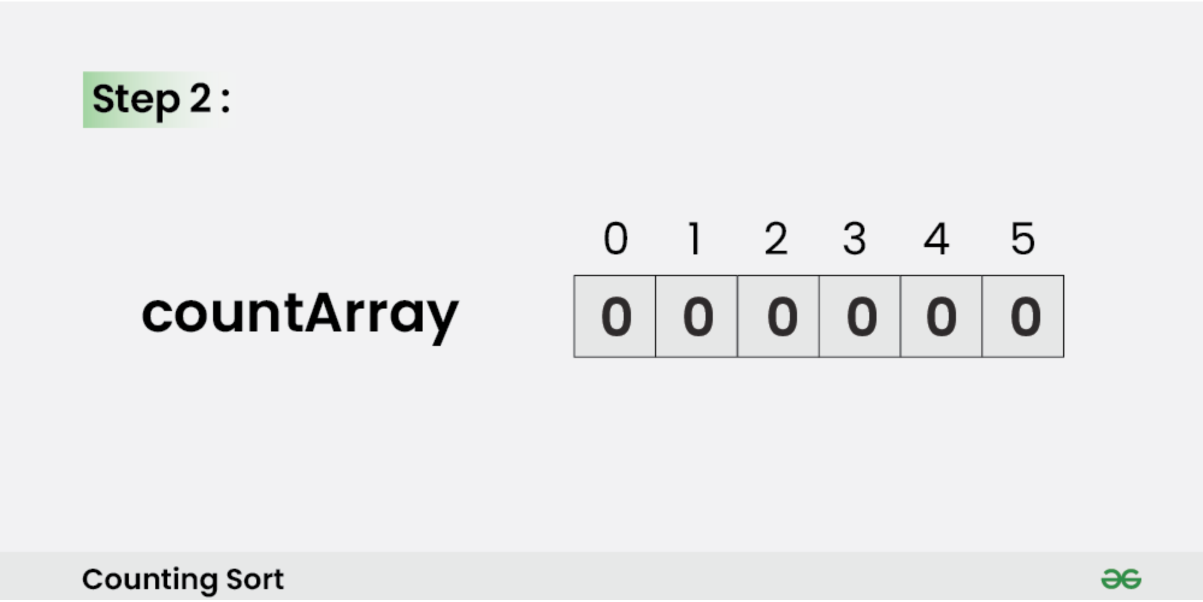
Step1 :

- Find out the **maximum** element from the given array.



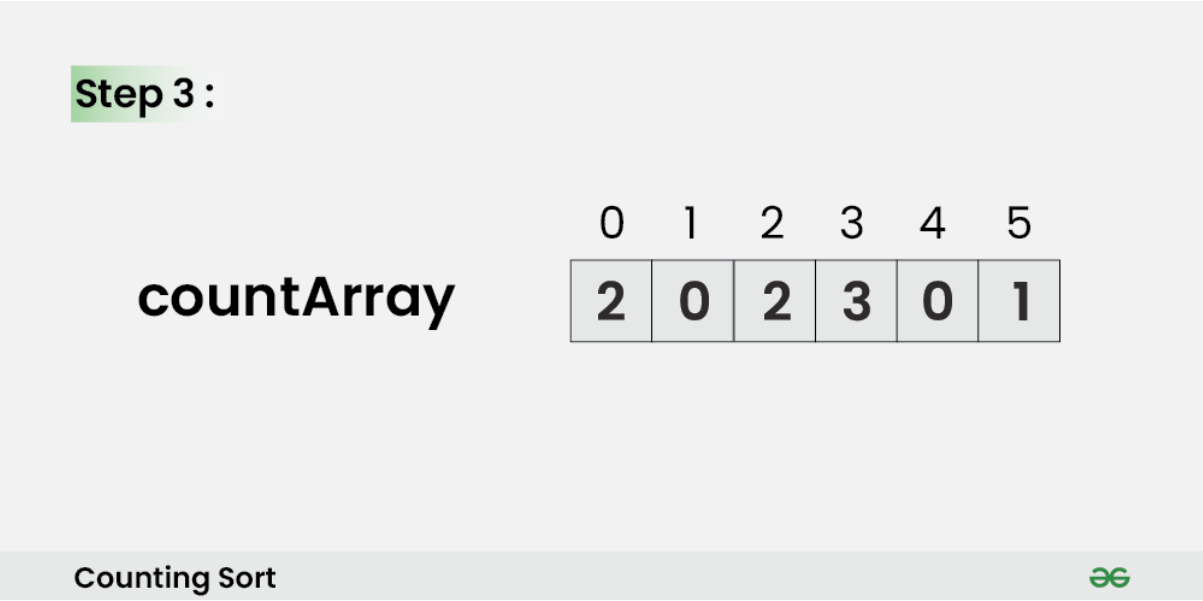
Step 2:

- Initialize a **countArray[]** of length **max+1** with all elements as **0**. This array will be used for storing the occurrences of the elements of the input array.



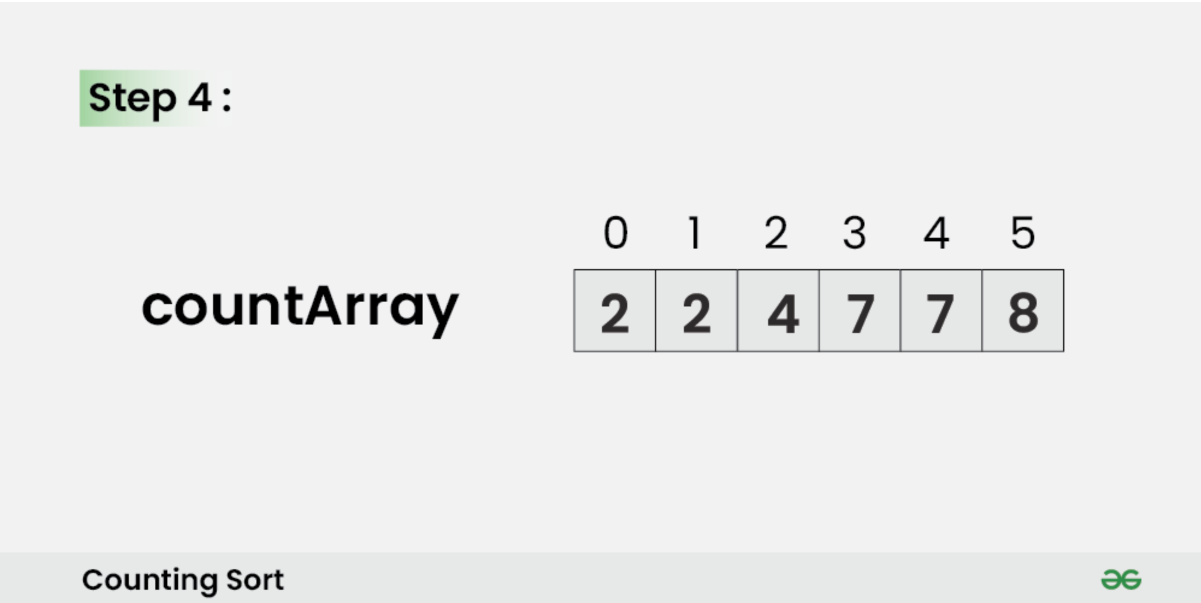
Step 3:

- In the **countArray[]**, store the count of each unique element of the input array at their respective indices.
- **For Example:** The count of element **2** in the input array is **2.** So, store **2** at index **2** in the **countArray[]**. Similarly, the count of element **5** in the input array is **1**, hence store **1** at index **5** in the **countArray[]**.



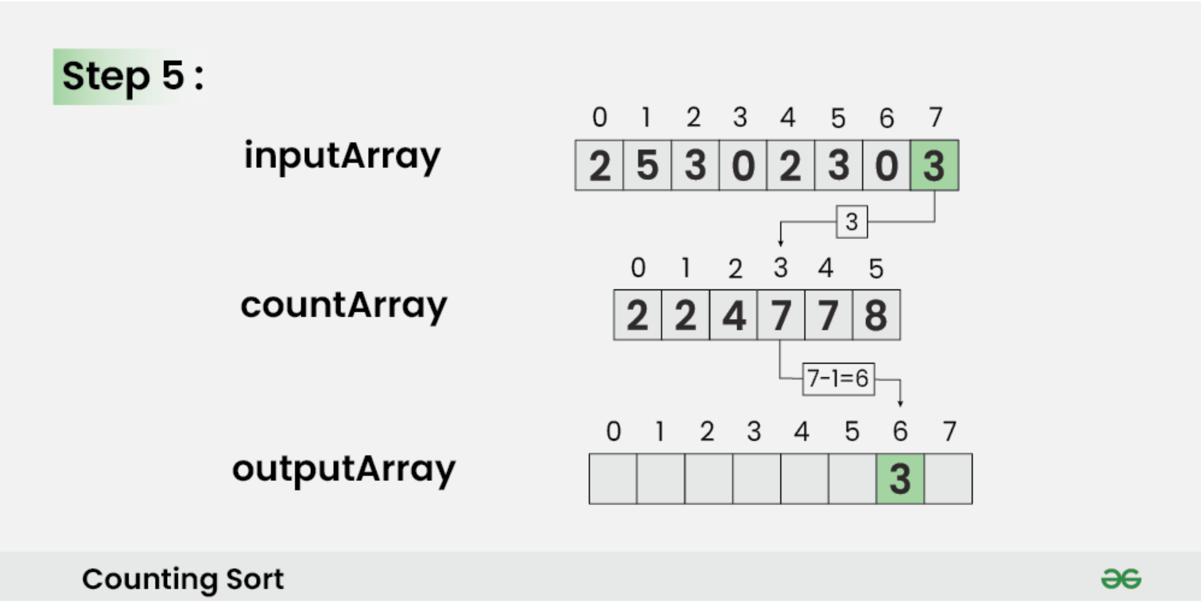
Step 4:

- Store the **cumulative sum** or **prefix sum** of the elements of the **countArray[]** by doing **countArray[i] = countArray[i – 1] + countArray[i].** This will help in placing the elements of the input array at the correct index in the output array.



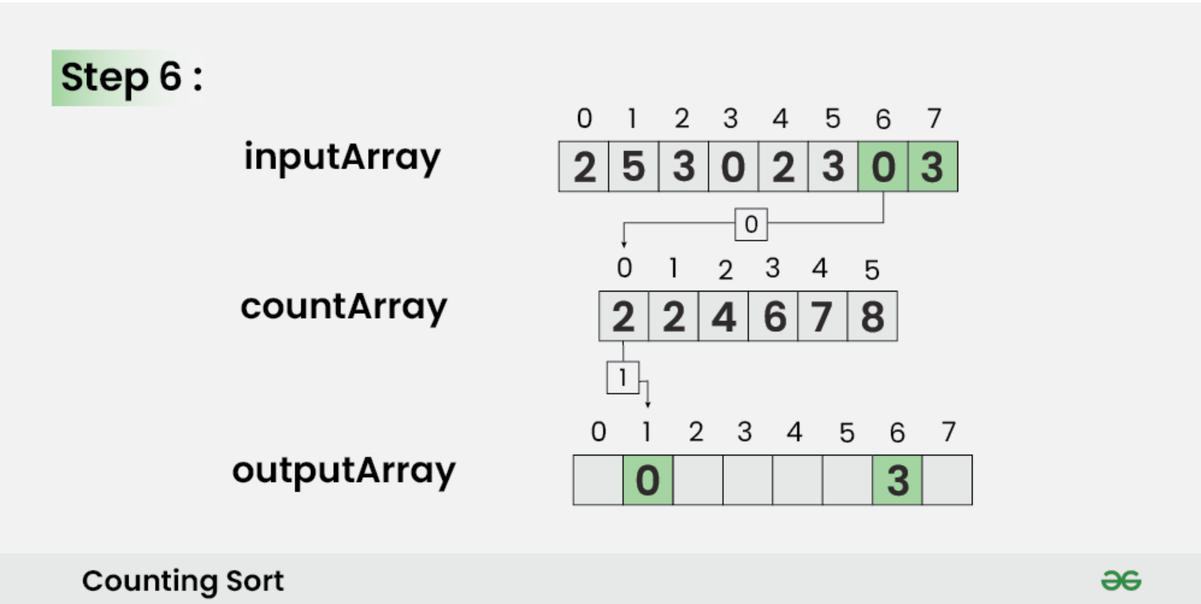
Step 5:

- Iterate from end of the input array and because traversing input array from end preserves the order of equal elements, which eventually makes this sorting algorithm **stable**.
- Update **outputArray[countArray[inputArray[i]] – 1] = inputArray[i]**.
- Also, update **countArray[inputArray[i]] = countArray[inputArray[i]]**– -.



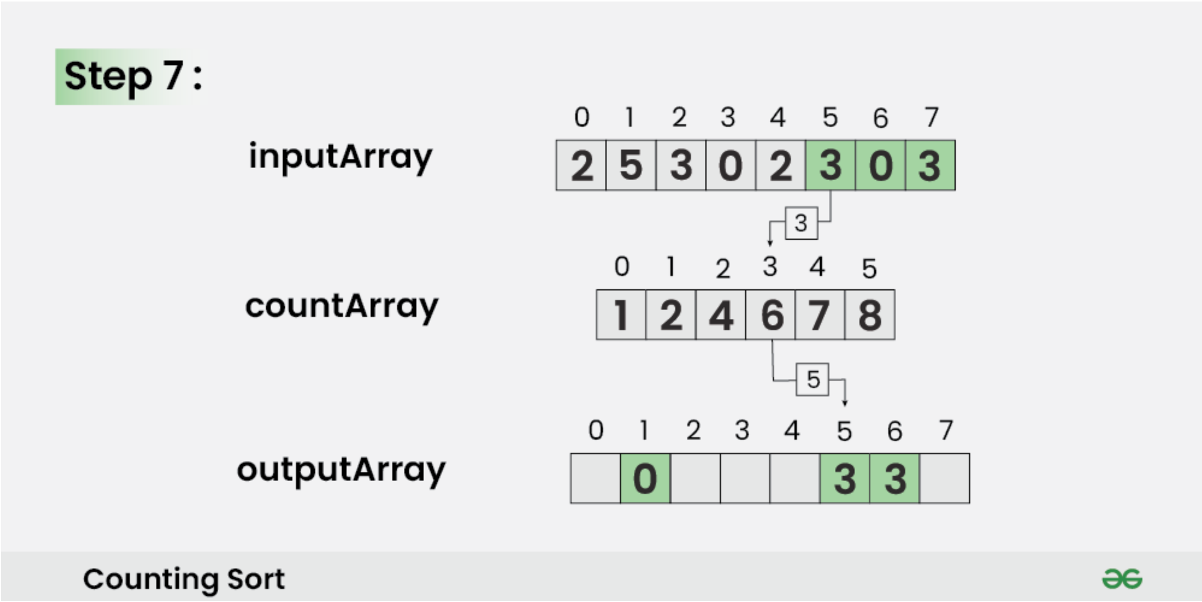
Step 6: **For i = 6**,

Update **outputArray[countArray[inputArray[6]] – 1] = inputArray[6]**
Also, update **countArray[inputArray[6]] = countArray[inputArray[6]]- –**



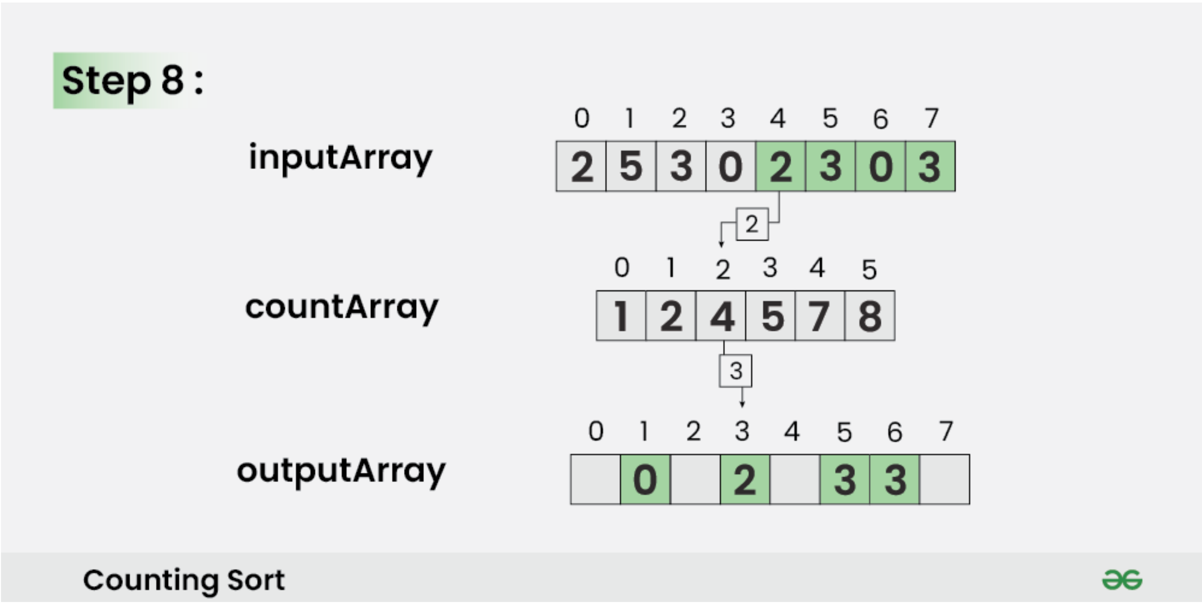
Step 7: **For i = 5**,

Update **outputArray[countArray[inputArray[5]] – 1] = inputArray[5]**
Also, update **countArray[inputArray[5]] = countArray[inputArray[5]]- –**



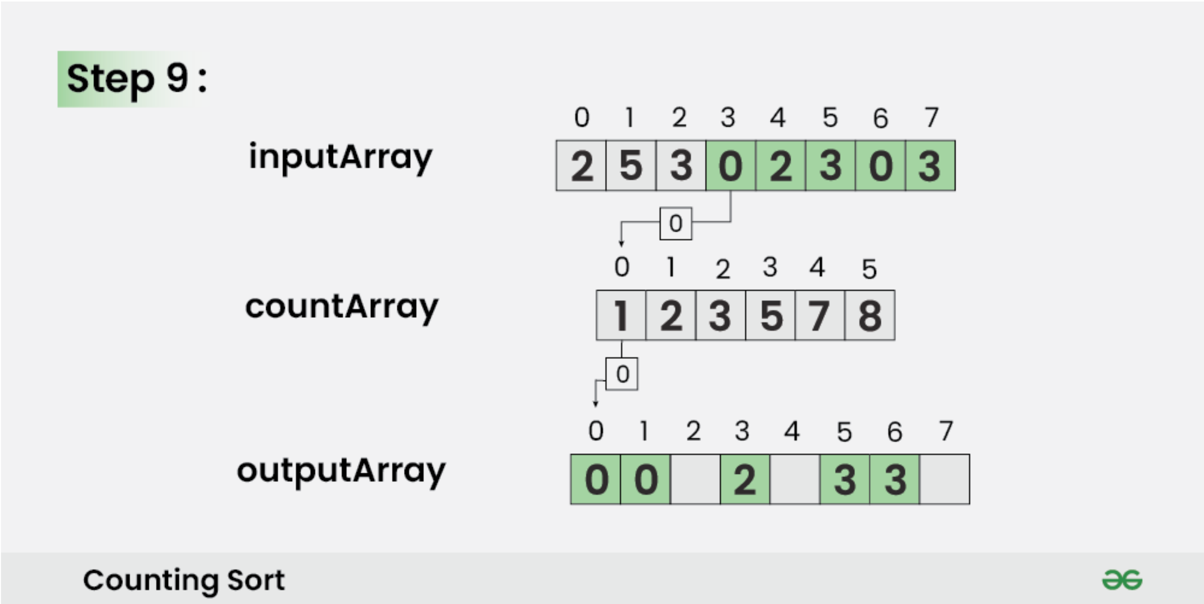
<u>Step 8:</u> For i = 4,

Update **outputArray[countArray[inputArray[4]] – 1] = inputArray[4]**
Also, update **countArray[inputArray[4]] = countArray[inputArray[4]]- –**



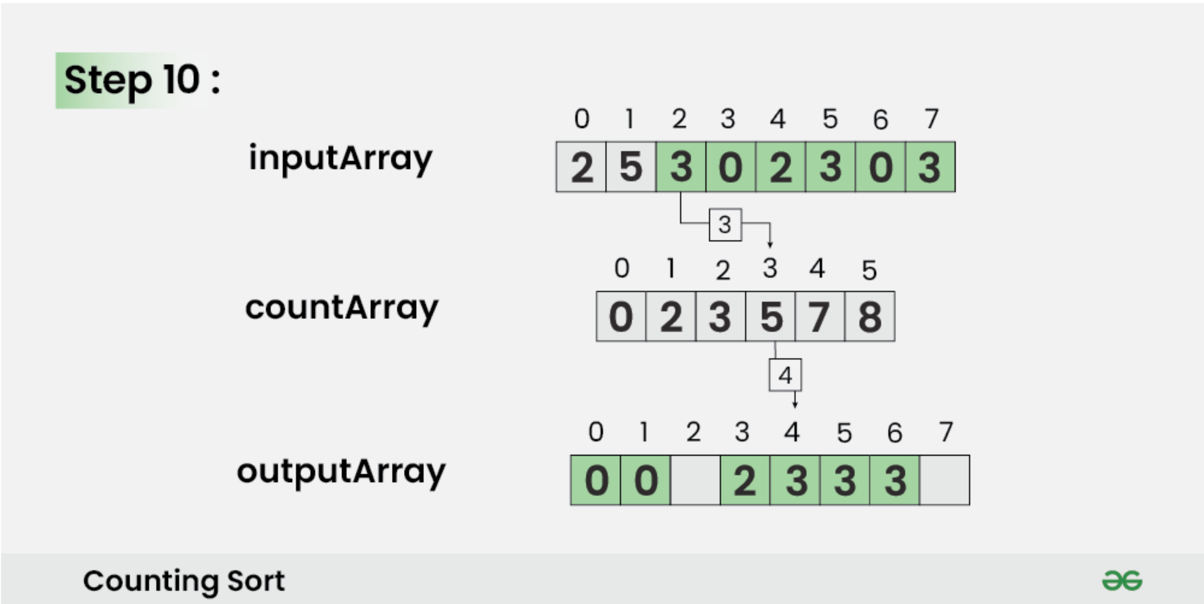
<u>Step 9:</u> For i = 3,

Update **outputArray[countArray[inputArray[3]] – 1] = inputArray[3]**
Also, update **countArray[inputArray[3]] = countArray[inputArray[3]]- –**



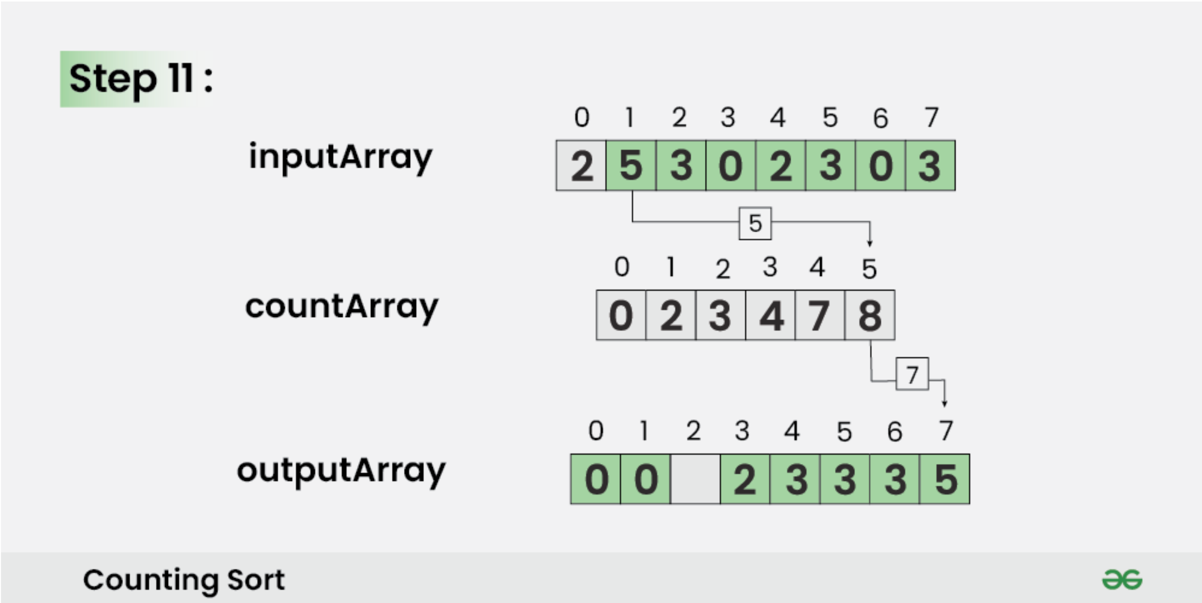
Step 10: **For i = 2**,

Update **outputArray[countArray[inputArray[2]] – 1] = inputArray[2]**
Also, update **countArray[inputArray[2]] = countArray[inputArray[2]]- –**



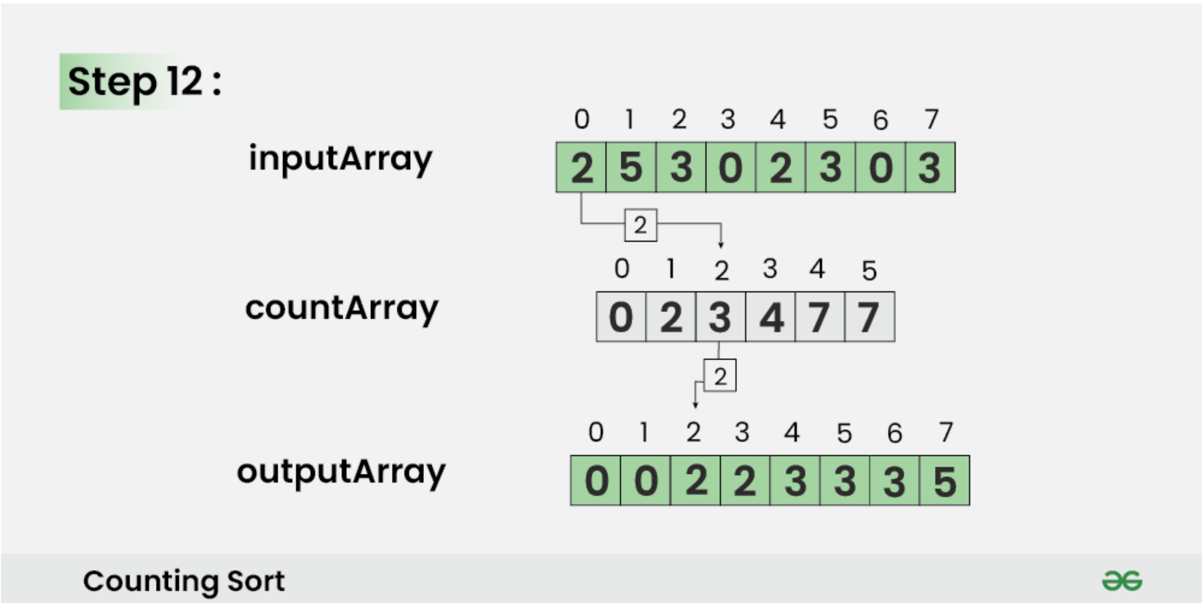
Step 11: **For i = 1**,

Update **outputArray[countArray[inputArray[1]] – 1] = inputArray[1]**
Also, update **countArray[inputArray[1]] = countArray[inputArray[1]]- –**



Step 12: **For i = 0,**

Update **outputArray[countArray[inputArray[0]] – 1] = inputArray[0]**
Also, update **countArray[inputArray[0]] = countArray[inputArray[0]]- –**



Algorithm:

Programming Language:

Code :

```
def count_sort(input_array):
        # Finding the maximum element of input_array.
        M = max(input_array)

        count_array = [0] * (M + 1)

        for num in input_array:
                count_array[num] += 1

        for i in range(1, M + 1):
                count_array[i] += count_array[i - 1]

        output_array = [0] * len(input_array)

        for i in range(len(input_array) - 1, -1, -1):
                output_array[count_array[input_array[i]] - 1] = input_array[i]
```

```
            count_array[input_array[i]] -= 1

        return output_array

if __name__ == "__main__":
        # Input array
        input_array = [4, 3, 12, 1, 5, 5, 3, 9]
        output_array = count_sort(input_array)

        for num in output_array:
                print(num, end=" ")
```

Space complexity: __________

Justification:____

____

Time complexity:

Best case time complexity: __________

Justification:____

____

Worst case time complexity: __________

Justification:____
____