Counting Sort

Comparison based Sorting Algorithms

Uptil now, all the algorithms that we have studied, do the comparisons to sort the array.

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort

Comparison Based Sorting Algorithms

Comparison Sorting Algorithm	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Bubble Sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
Selection Sort	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(1)$
Insertion Sort	$O(N)$	$O(N^2)$	$O(N^2)$	$O(1)$
Merge Sort	$O(N*\log_2 N)$	$O(N*\log_2 N)$	$O(N*\log_2 N)$	$O(N)$
Quick Sort	$O(N*\log_2 N)$	$O(N*\log_2 N)$	$O(N^2)$	$O(N)$
Heap Sort	$O(N*\log_2 N)$	$O(N*\log_2 N)$	$O(N*\log_2 N)$	$O(1)$

Comparison based Sorting Algorithms

Average Time Complexity of any comparison based sorting algorithm can not be better than the $O(n*\log_2(n))$.

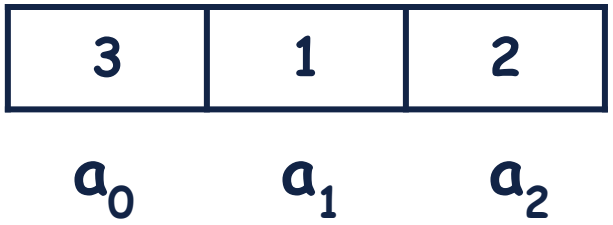
Comparison based Sorting Algorithms

Average Time Complexity of any comparison based sorting algorithm can not be better than the $O(n*\log_2(n))$.

Why?

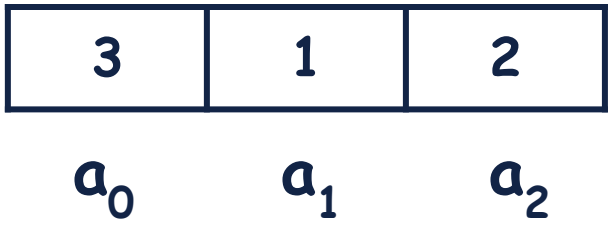
Comparison Sorting Algorithms

Let's say we have data with 3 elements and we want to sort them.



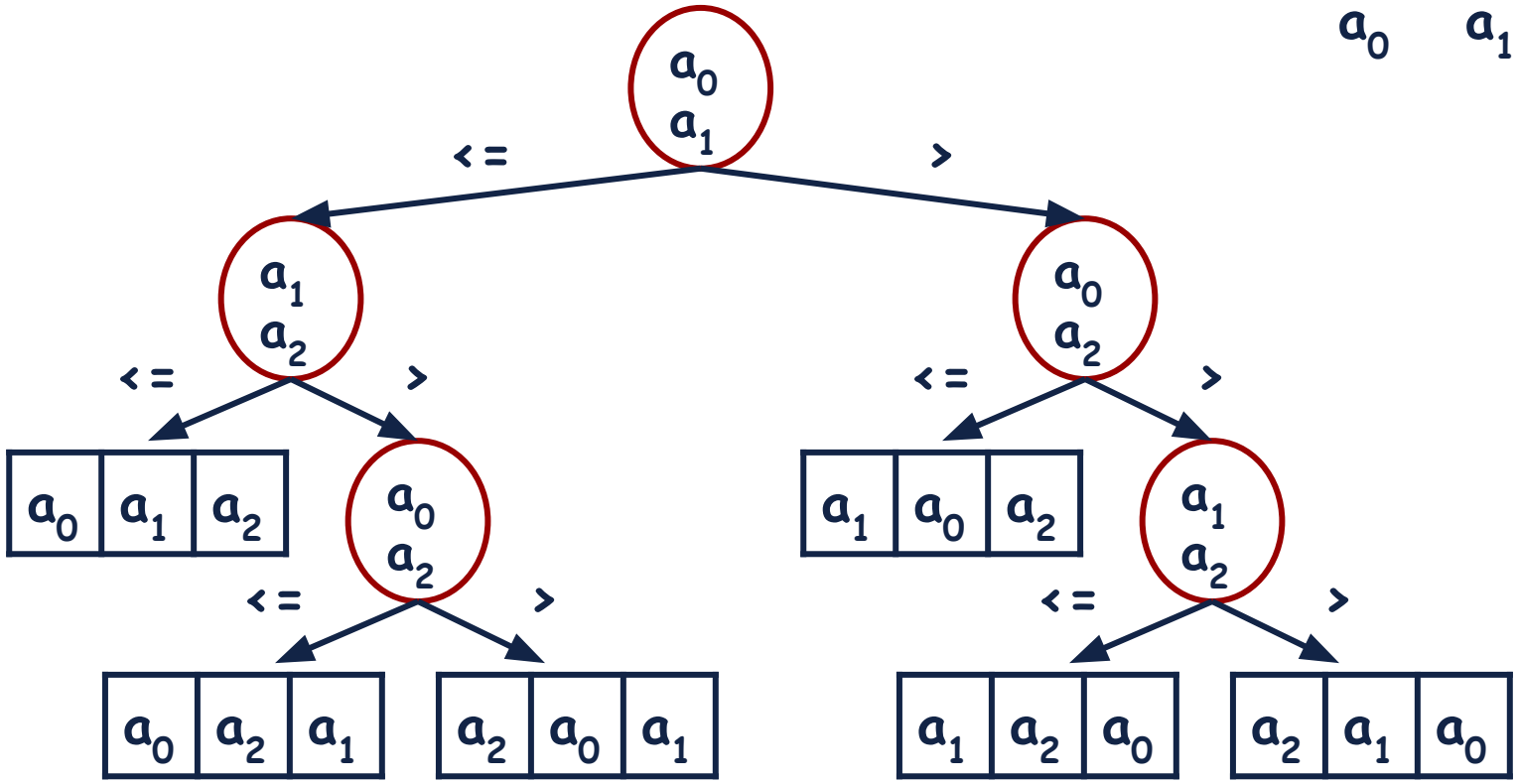
Comparison Sorting Algorithms

Let's say we have data with 3 elements and we want to sort them.

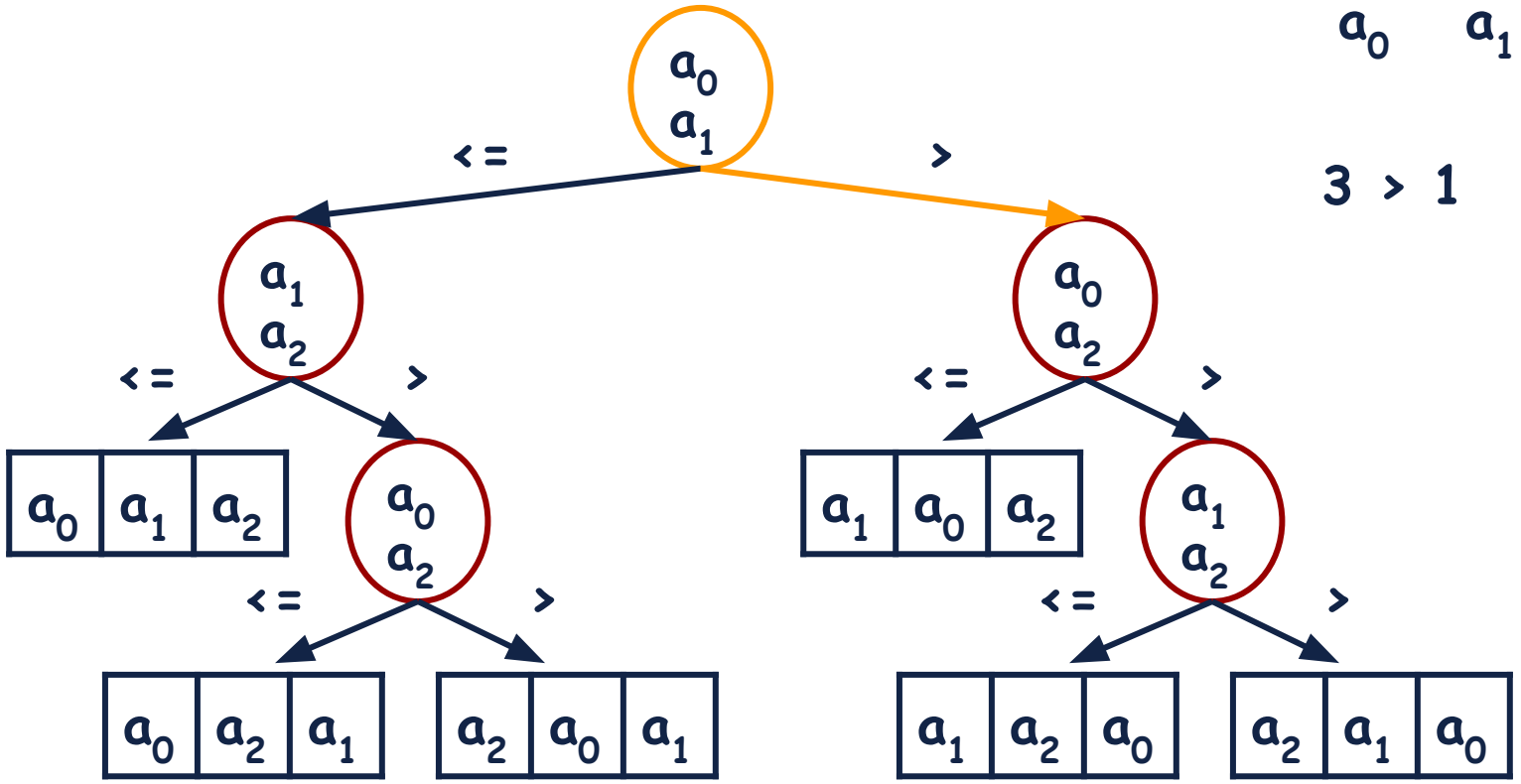


We can perform 2 comparisons at a time. Let's make a **Decision Tree** to understand it clearly.

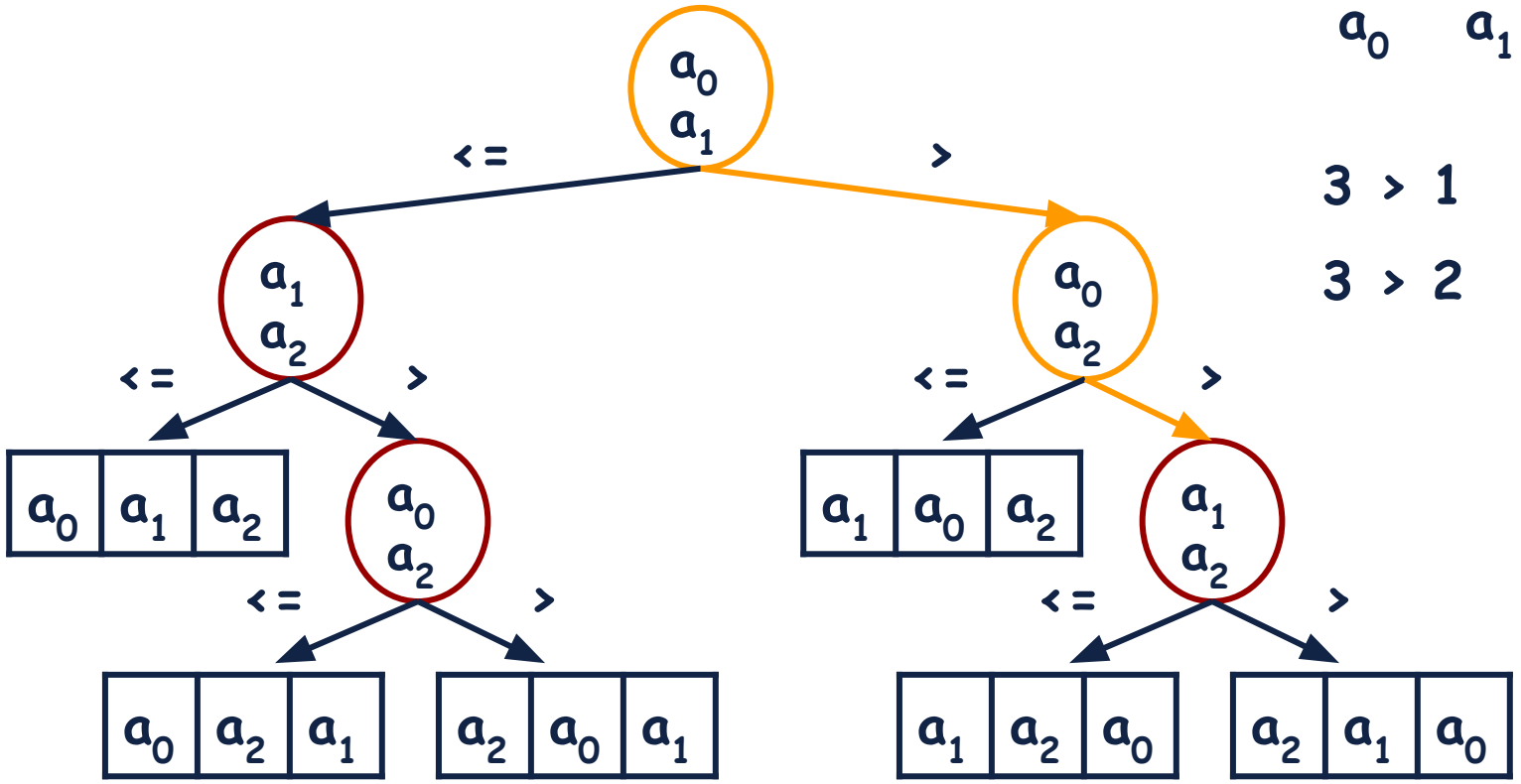
Comparison Sorting Algorithms



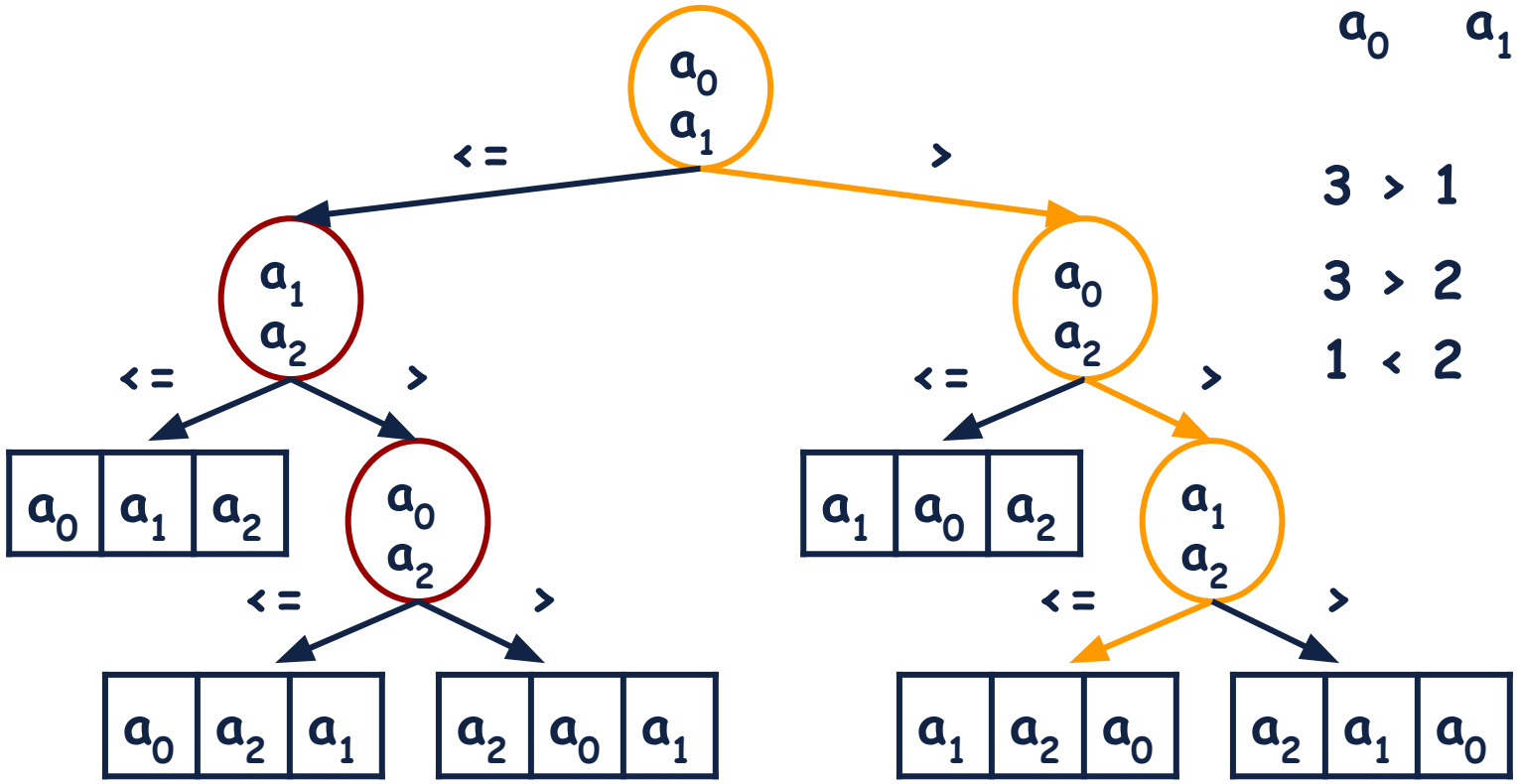
Comparison Sorting Algorithms



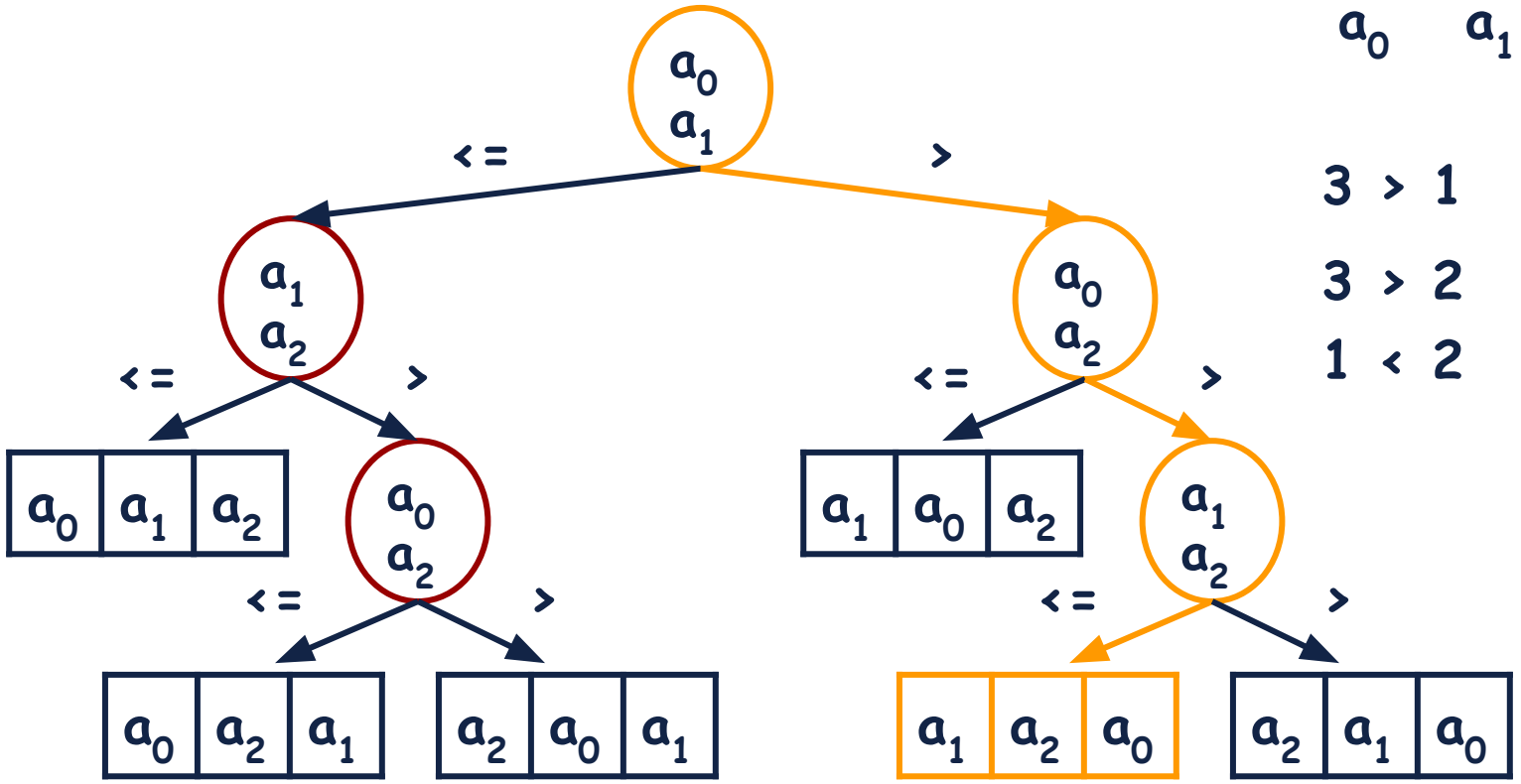
Comparison Sorting Algorithms



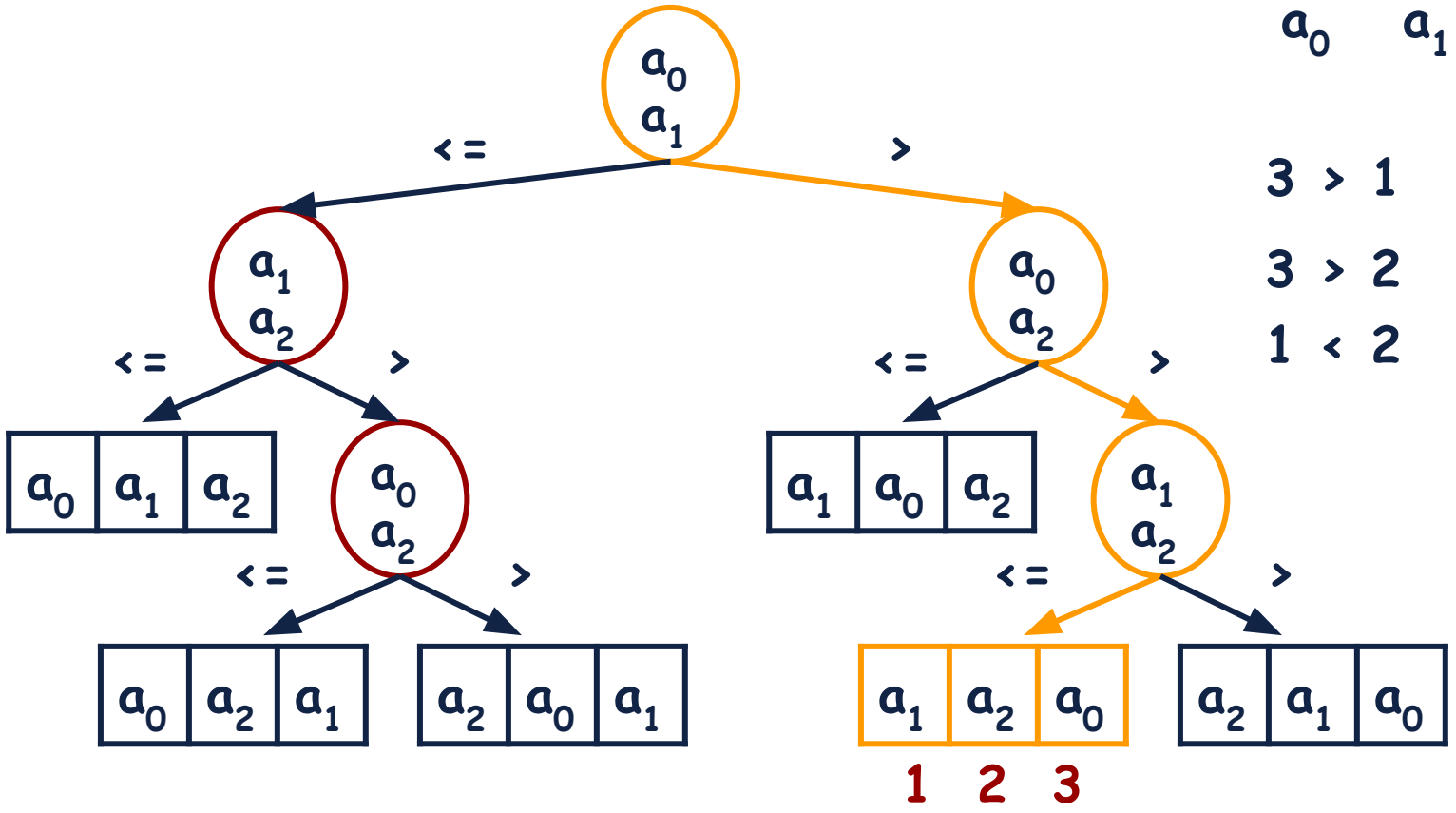
Comparison Sorting Algorithms



Comparison Sorting Algorithms



Comparison Sorting Algorithms



Non-Comparison Sorting Algorithms

There are also some sorting algorithms that does not do comparisons to sort the elements and their Average time complexity is better than $O(n*\log_2(n))$.

Non-Comparison Sorting Algorithms

Let's see those non-comparison based sorting algorithms now.

Counting Sort

Counting Sort Algorithm

As the name suggests, this algorithm has to do something with counting.

Counting Sort

Suppose, we want to sort this array.

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

Counting Sort: Working

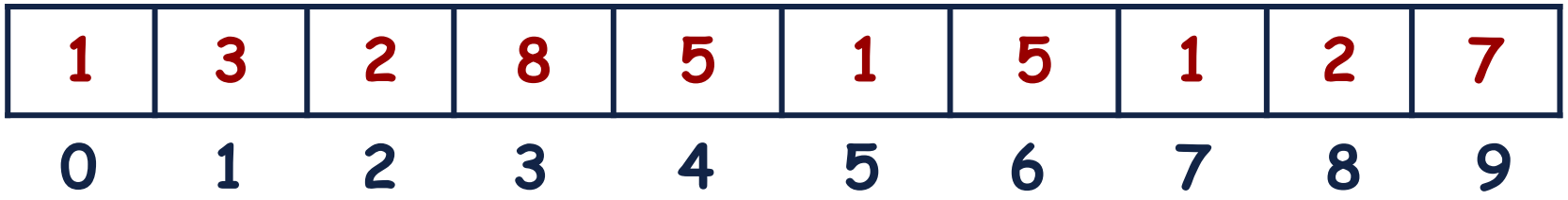
Step 1: Find out the maximum element from the given array.

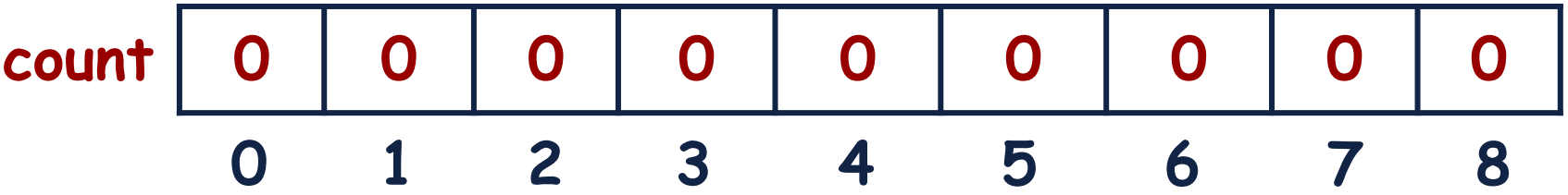
1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

Although, we are doing comparisons to find the max element. So, the joke is on them

Counting Sort: Working

Step 2: Initialize another array of length max+1 with all elements as 0. This array will be used for storing the count of the elements in the array.





Auxiliary array/data structure is a fancy way of saying helper array/data structure

Counting Sort: Working

Step 3: Store the count of each element at their respective index in the auxiliary count array

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

count

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

Auxiliary array/data structure is a fancy way of saying helper array/data structure

Counting Sort: Working

Step 3: Store the count of each element at their respective index in the auxiliary count array

For example: the count of element 1 is 3 therefore, 3 is stored on the 1st index of count array.

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

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

Auxiliary array/data structure is a fancy way of saying helper array/data structure

Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

$$count[i] = \sum count[x];\quad 0 <= x <= i$$

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

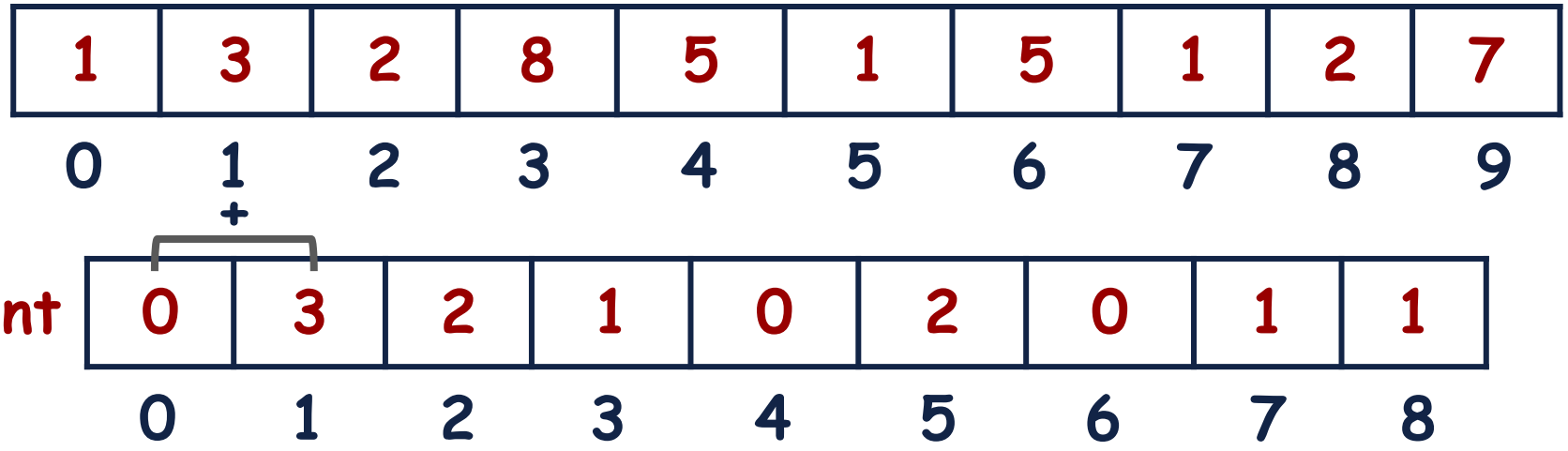
count

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

Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

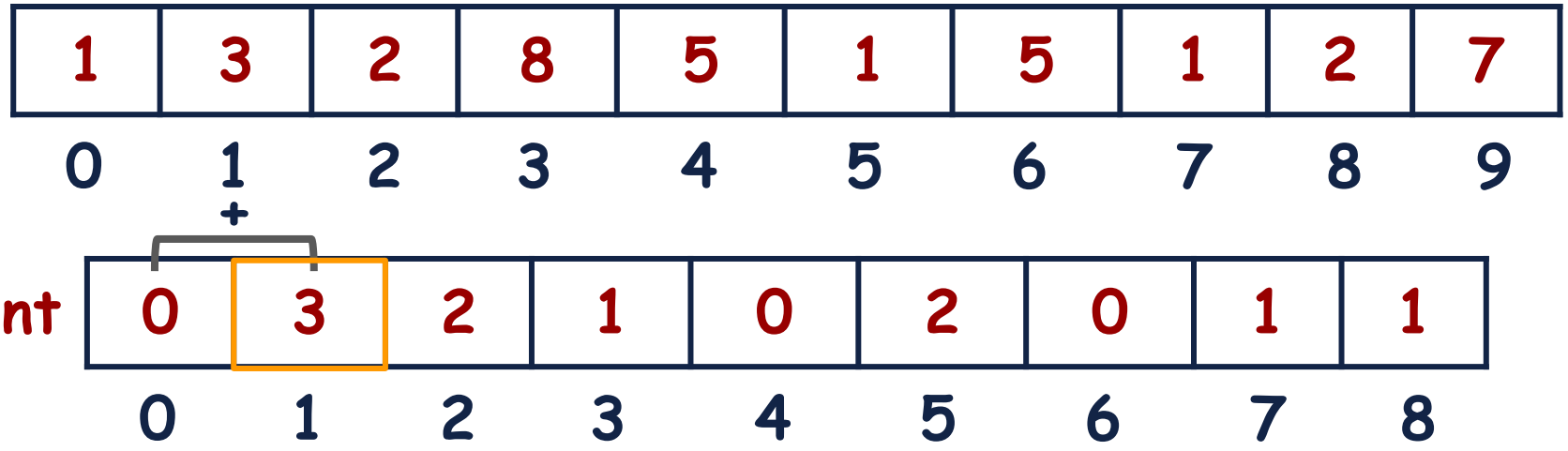
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

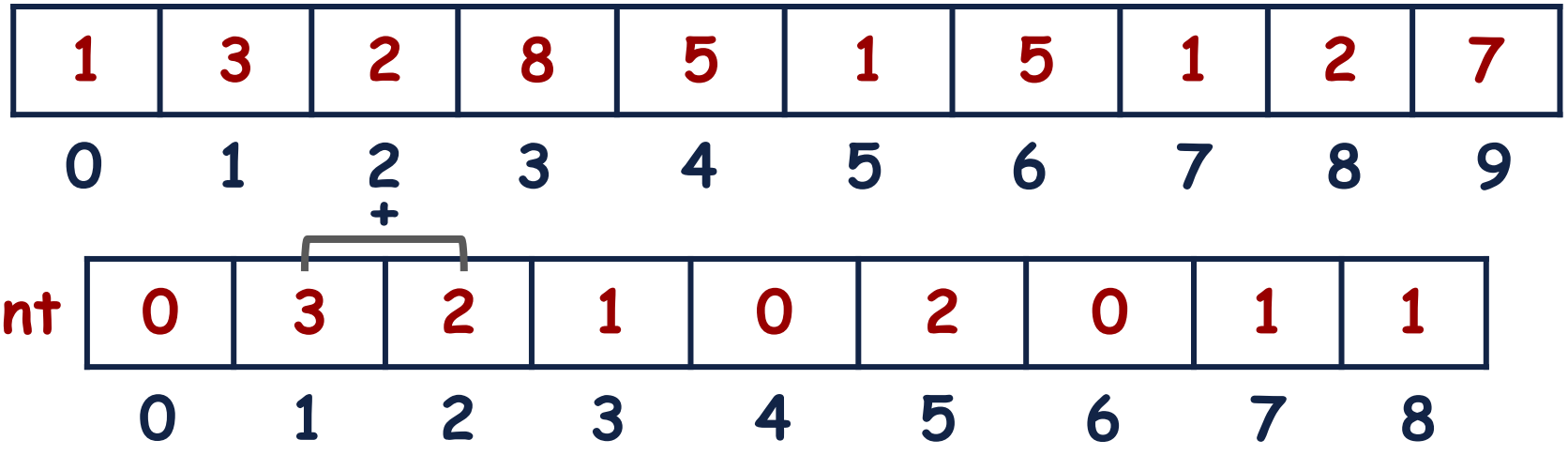
$$count[i] = \sum count[x]; \quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

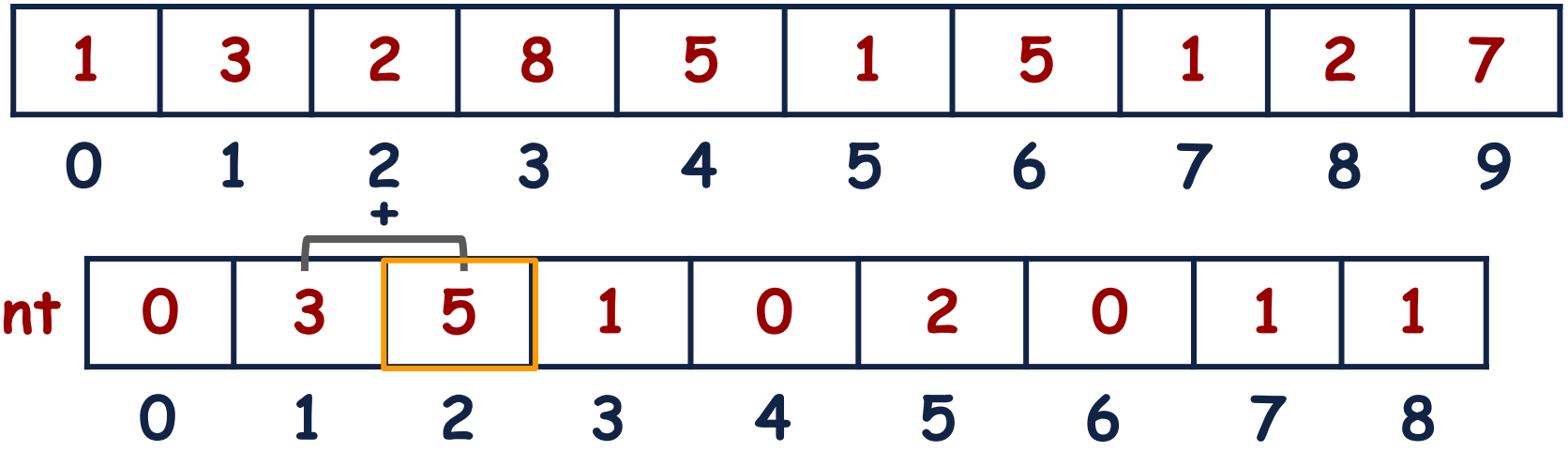
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

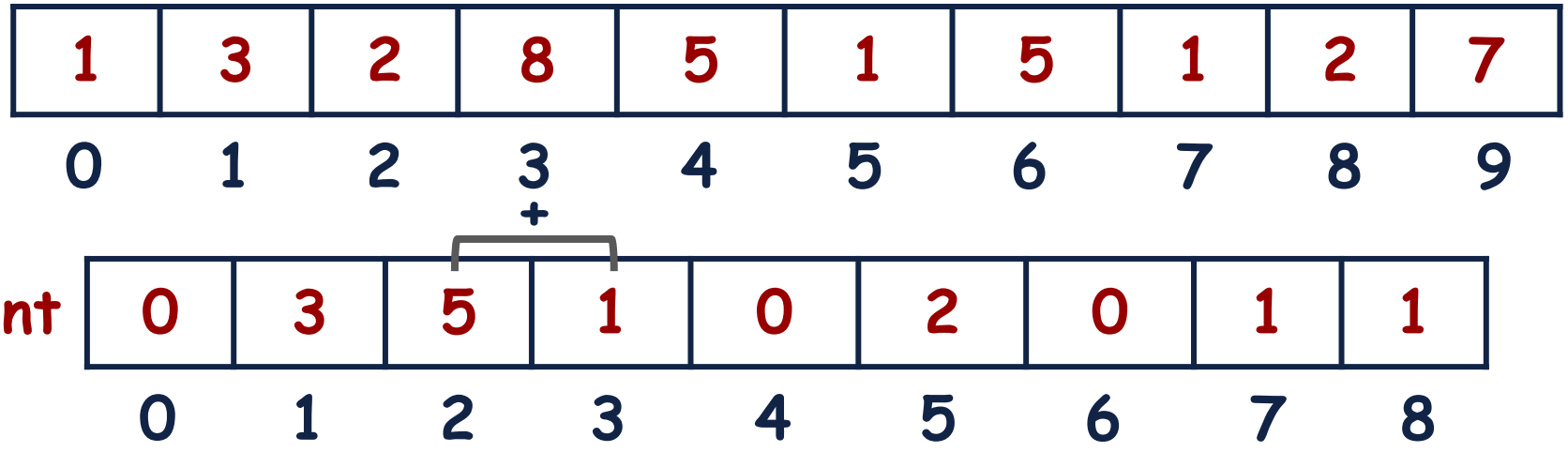
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

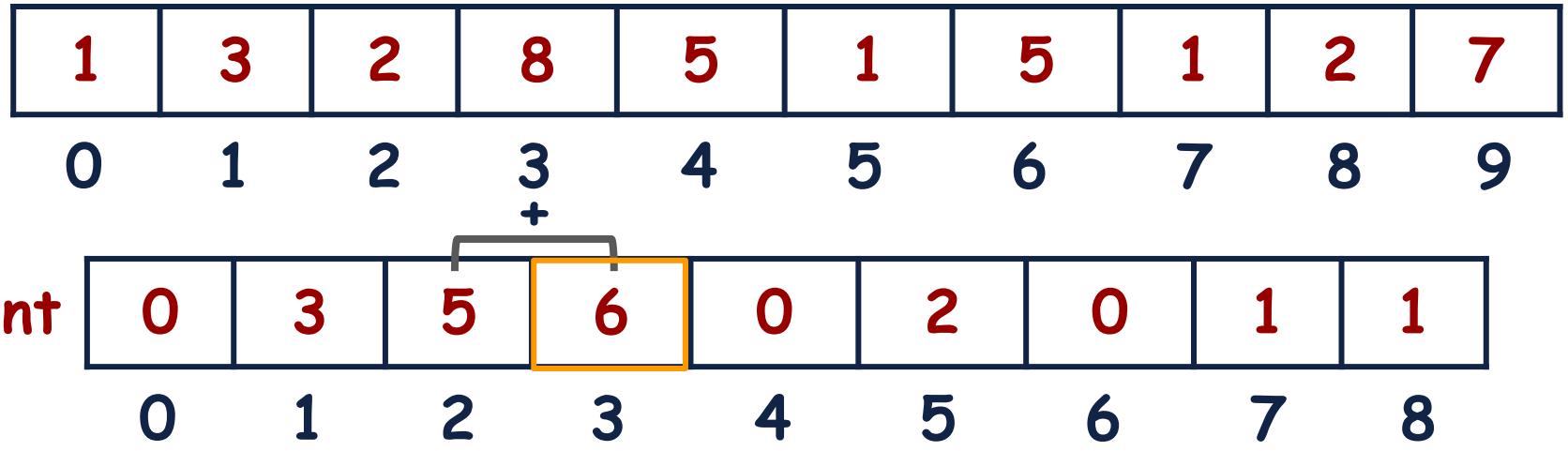
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

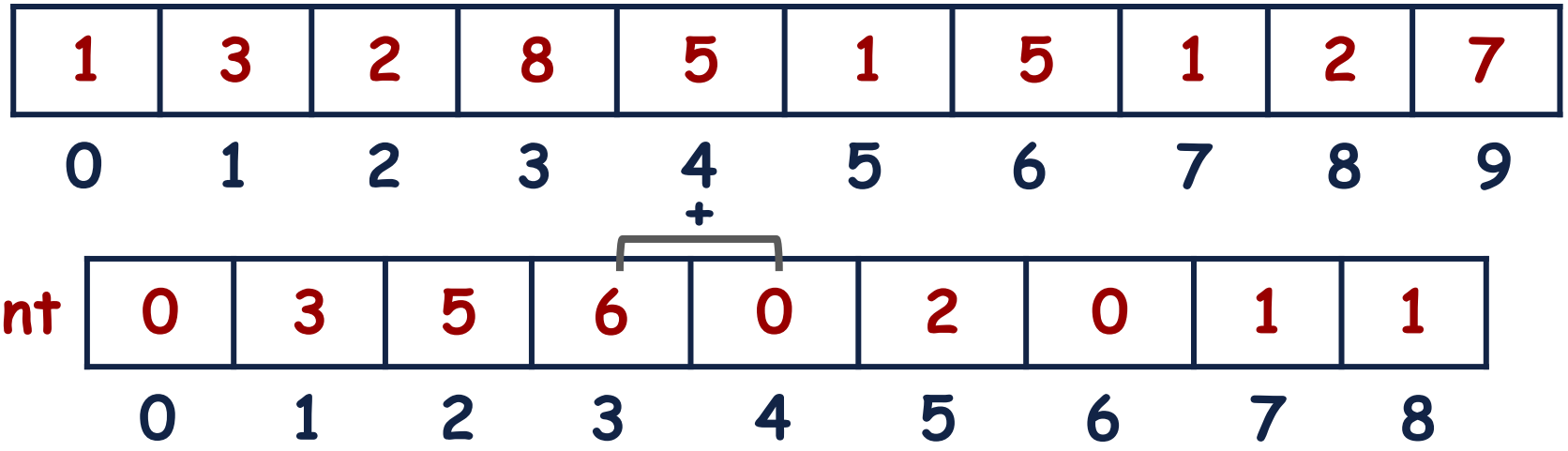
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

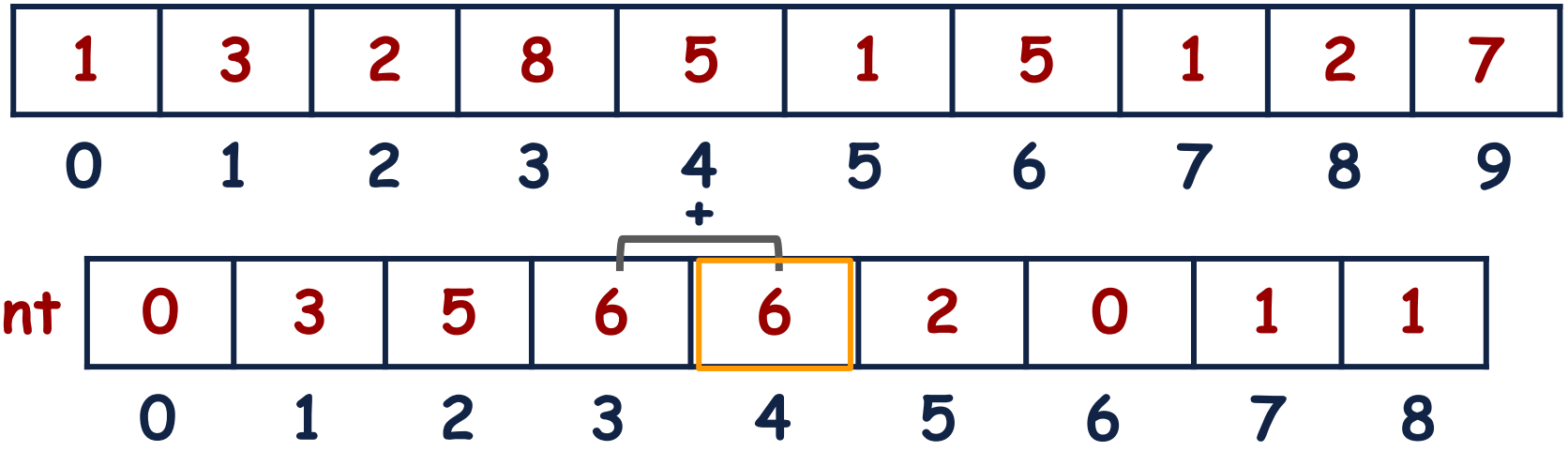
$$count[i] = \sum count[x]; \quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

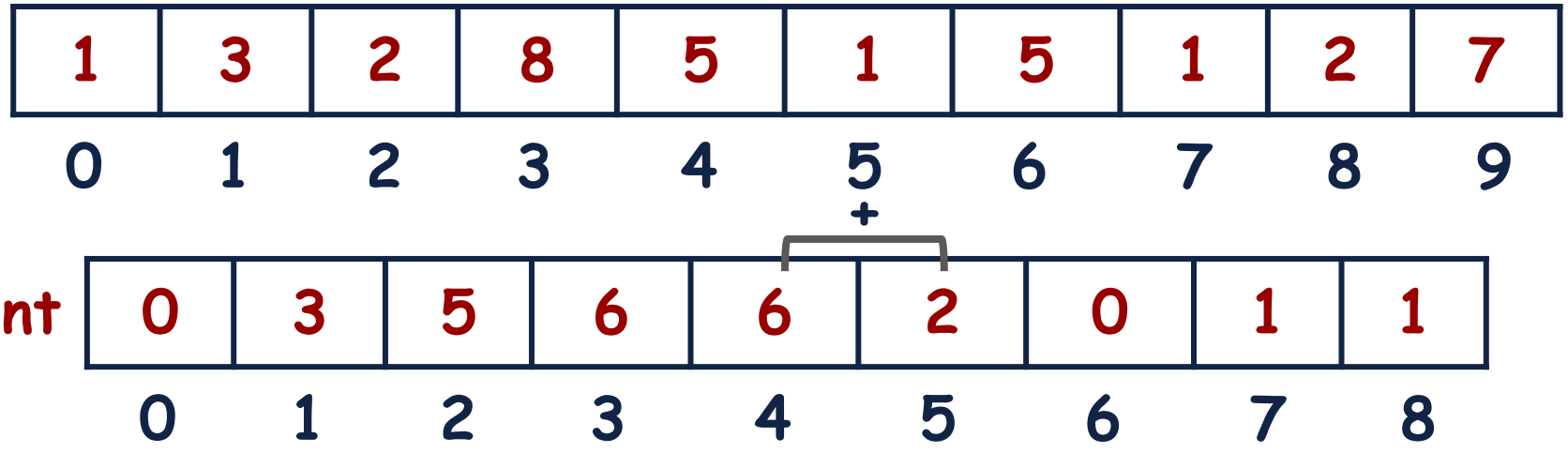
$$count[i] = \sum count[x]; \quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

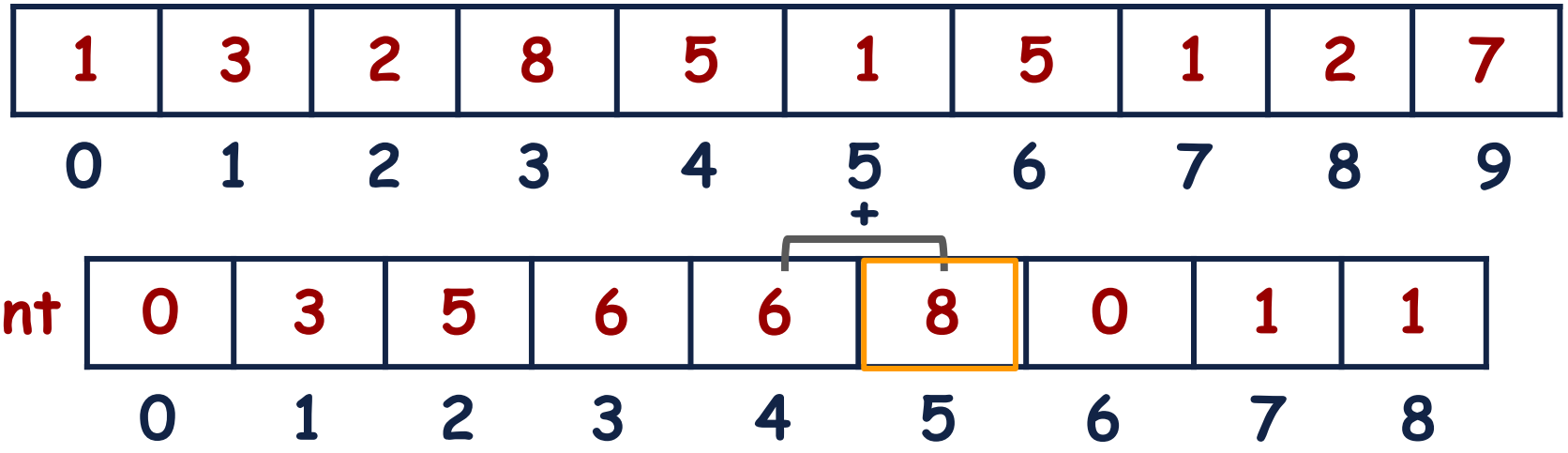
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

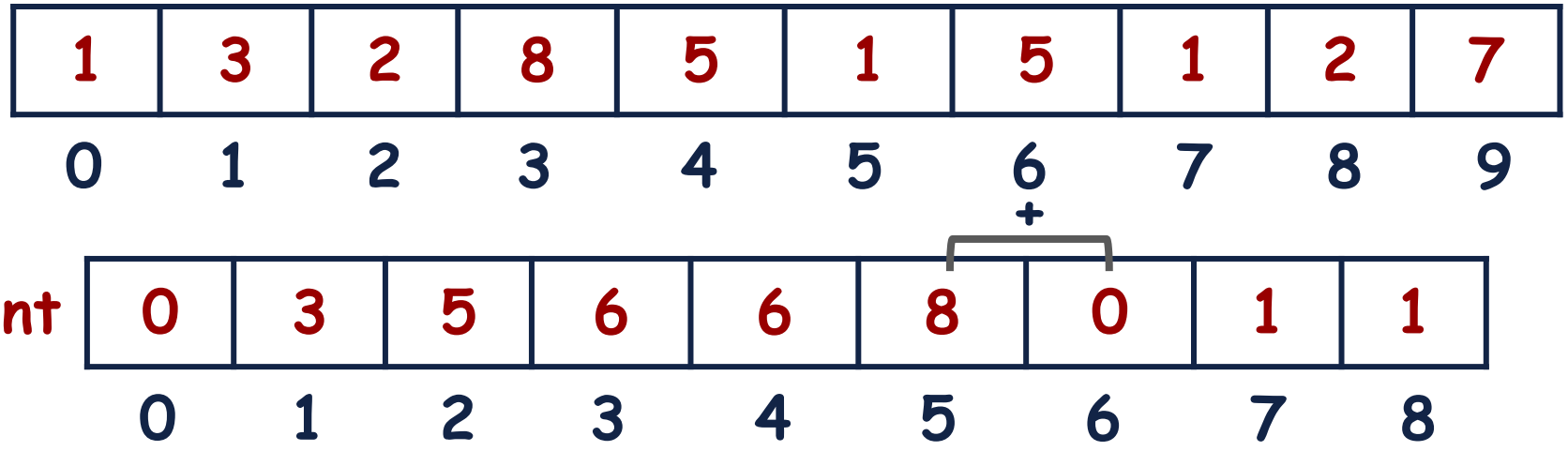
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

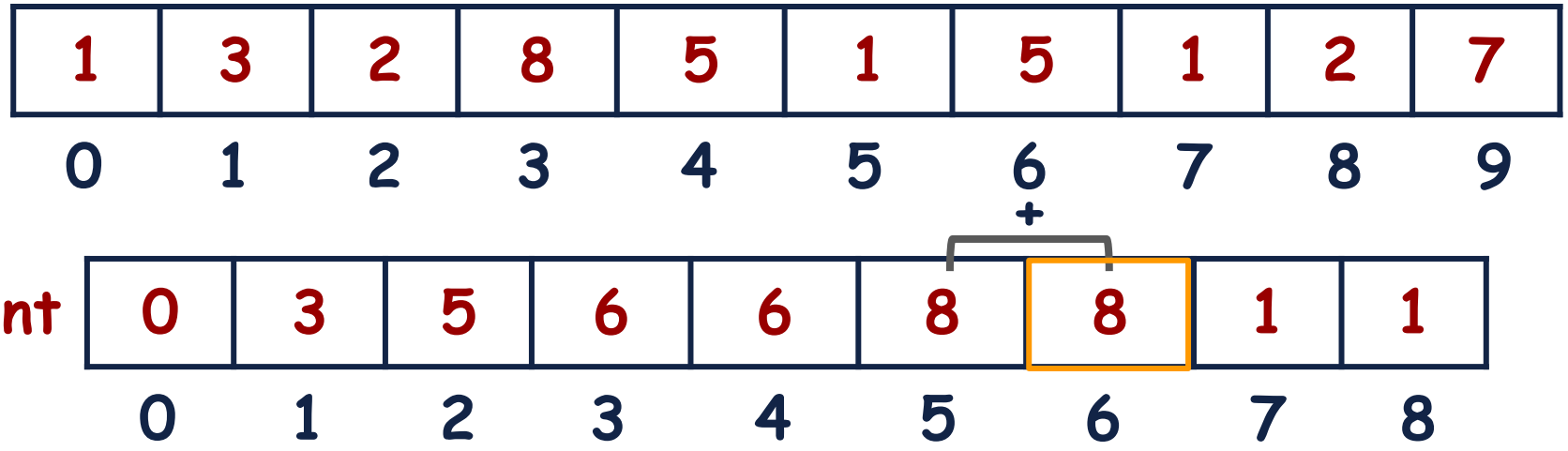
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

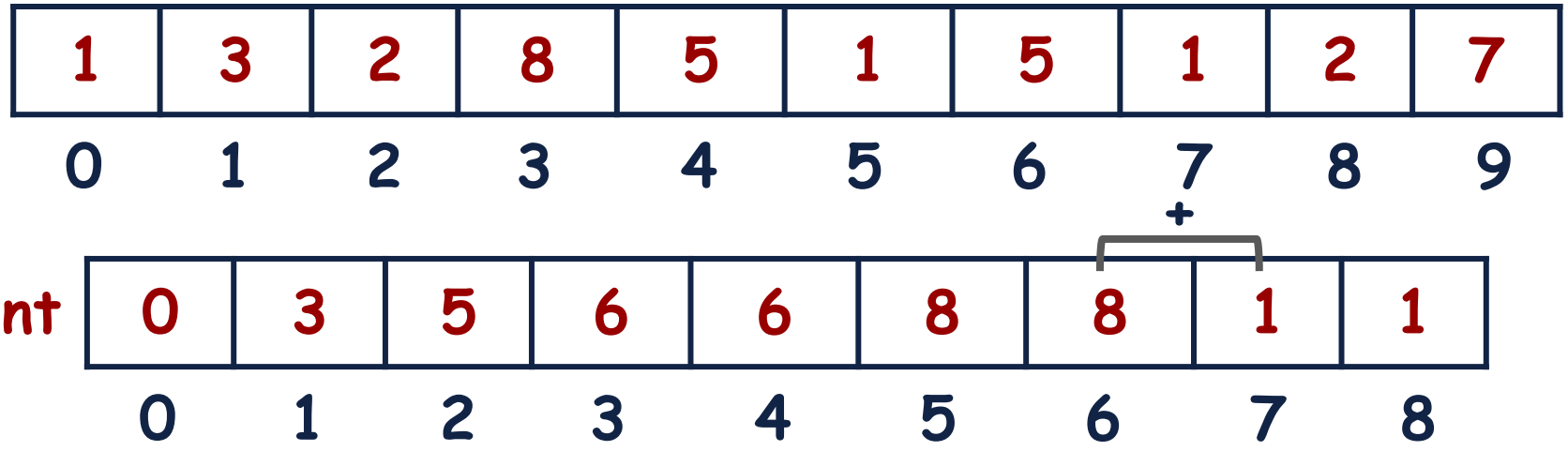
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

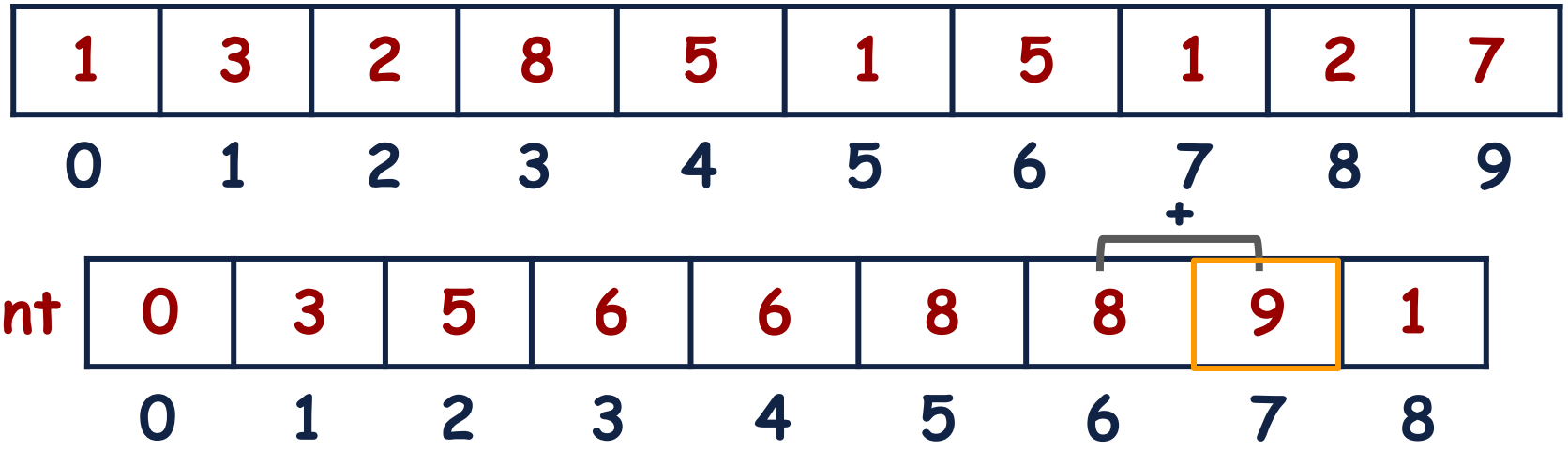
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

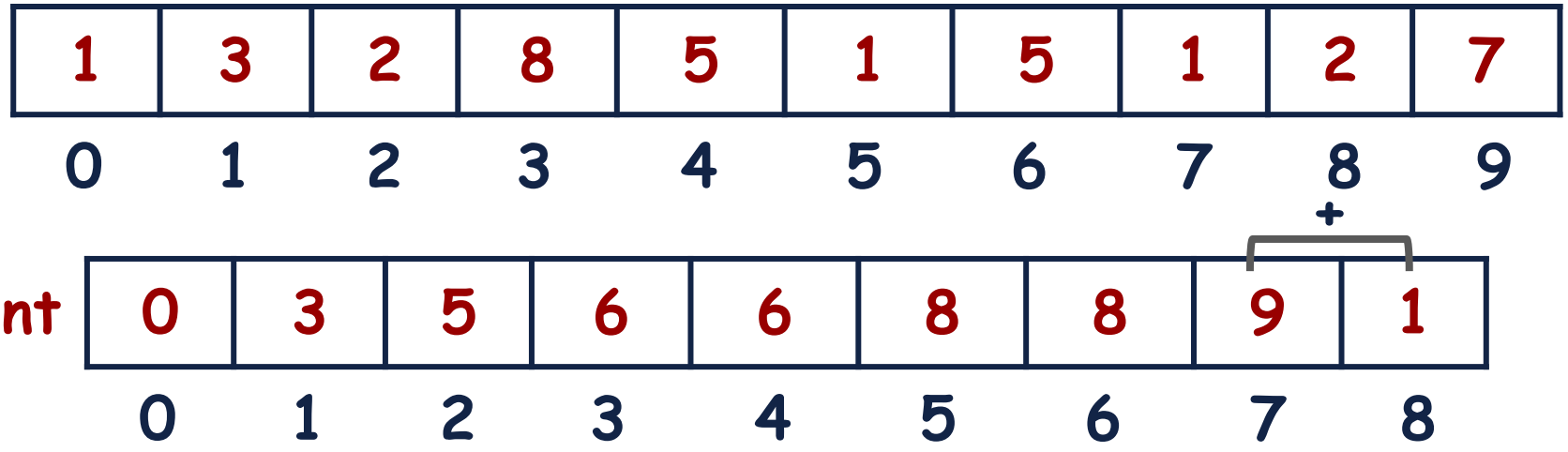
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 4: Store the cumulative sum of the elements of the count array.

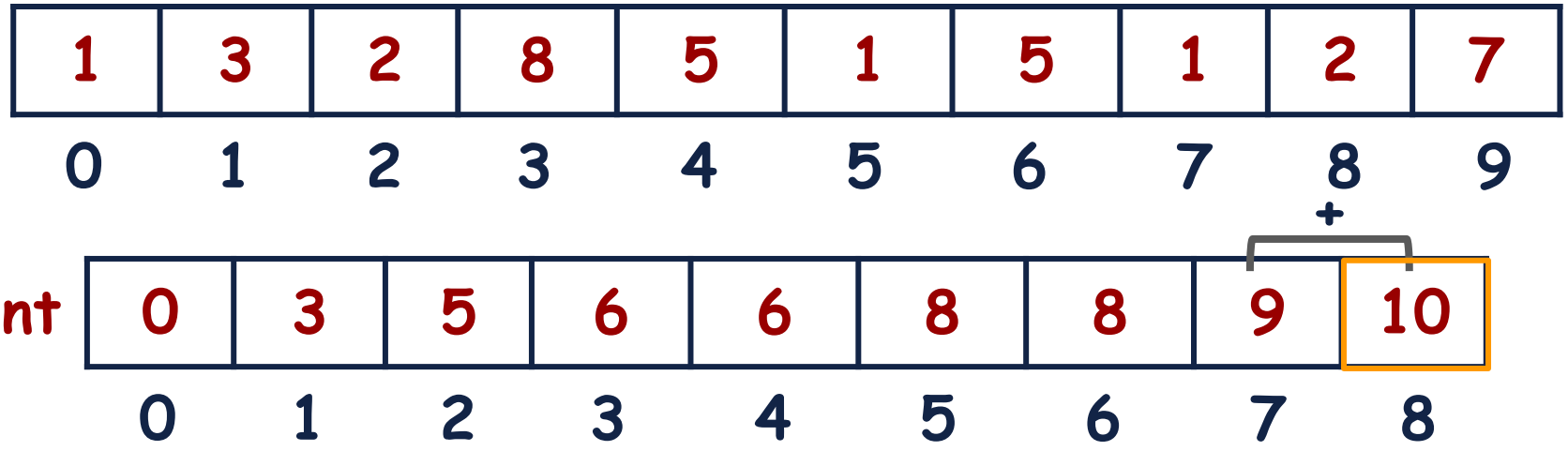
$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

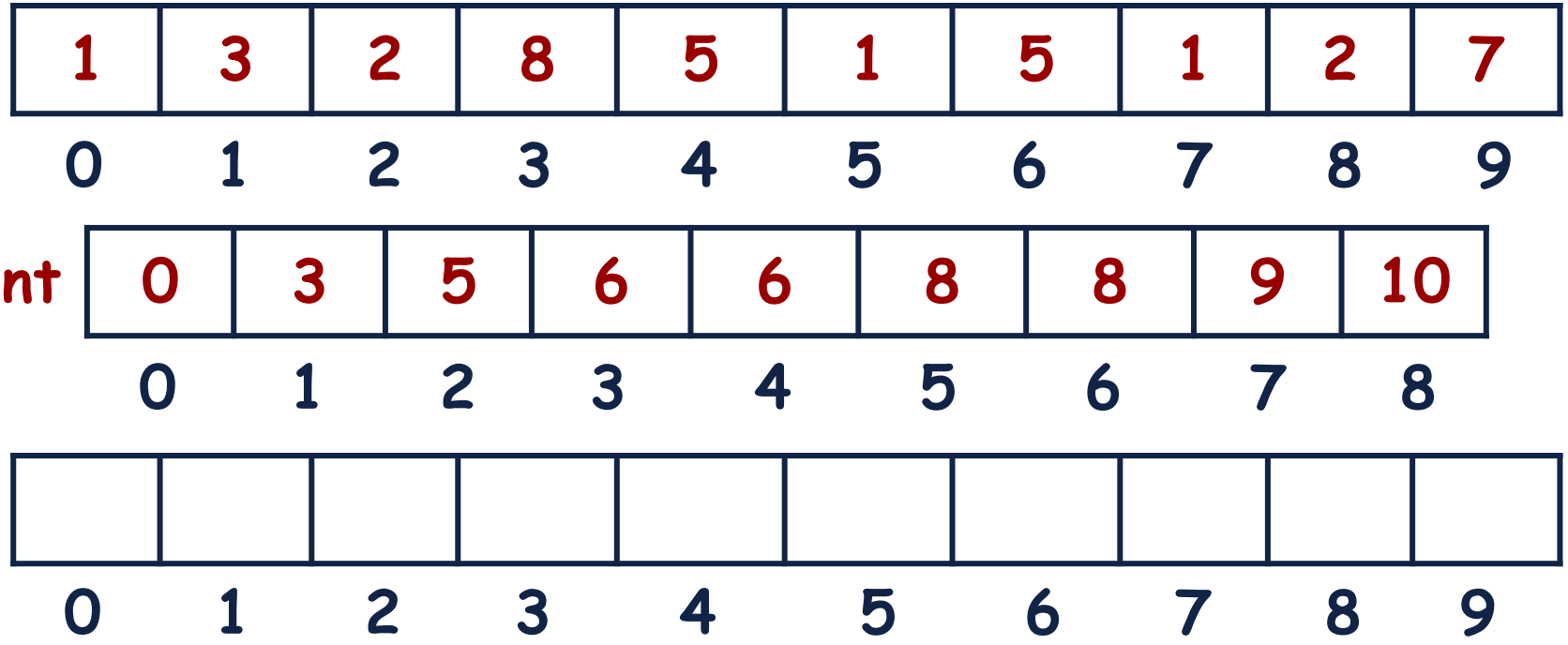
Step 4: Store the cumulative sum of the elements of the count array.

$$count[i] = \sum count[x];\quad 0 <= x <= i$$



Counting Sort: Working

Step 5: Declare another Output Array.



Counting Sort: Working

Step 6: Find the index of each element of the original array in the count array. Place the element at the index in output array by subtracting -1

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

count

0	3	5	6	6	8	8	9	10
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8	9

Counting Sort: Working

Step 6: Find the index of each element of the original array in the count array. Place the element at the index in output array by subtracting -1

1	3	2	8	5	1	5	1	2	7
0	1	2	3	4	5	6	7	8	9

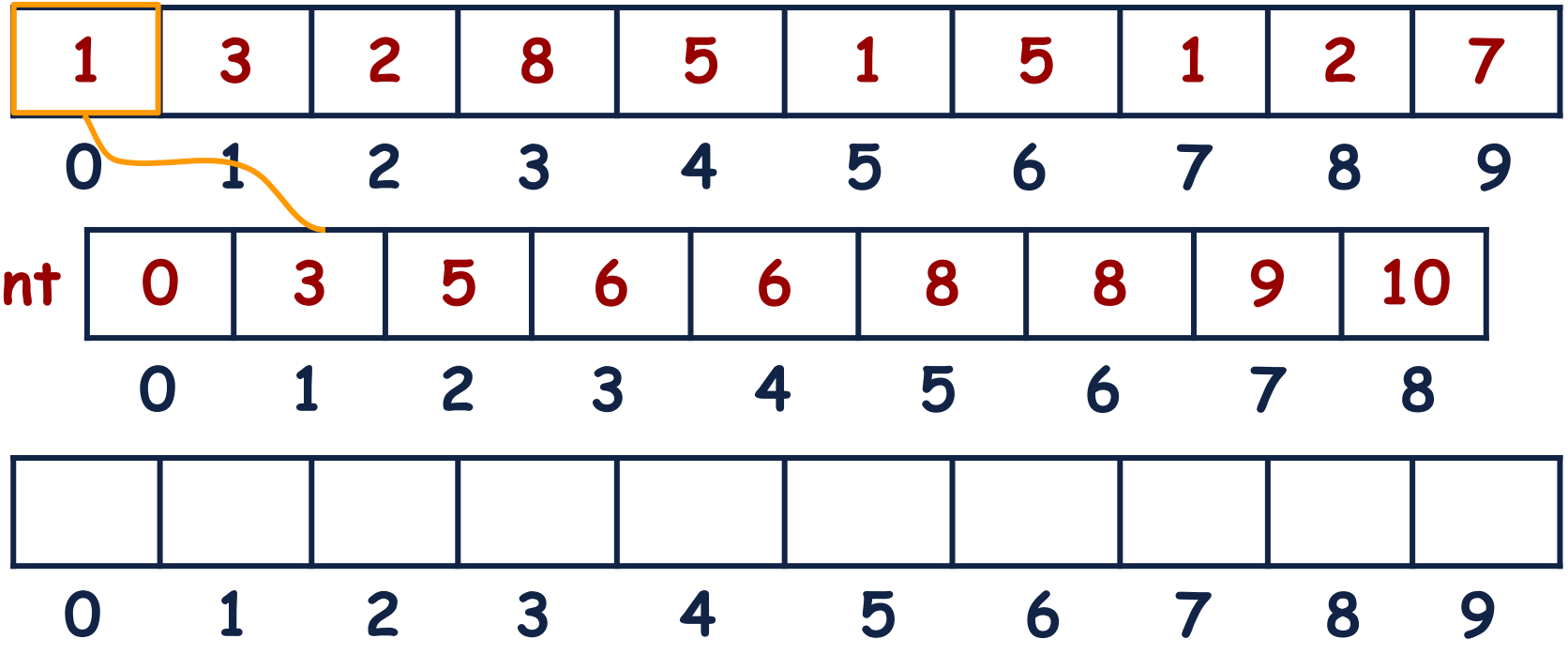
count

0	3	5	6	6	8	8	9	10
0	1	2	3	4	5	6	7	8

0	1	2	3	4	5	6	7	8	9

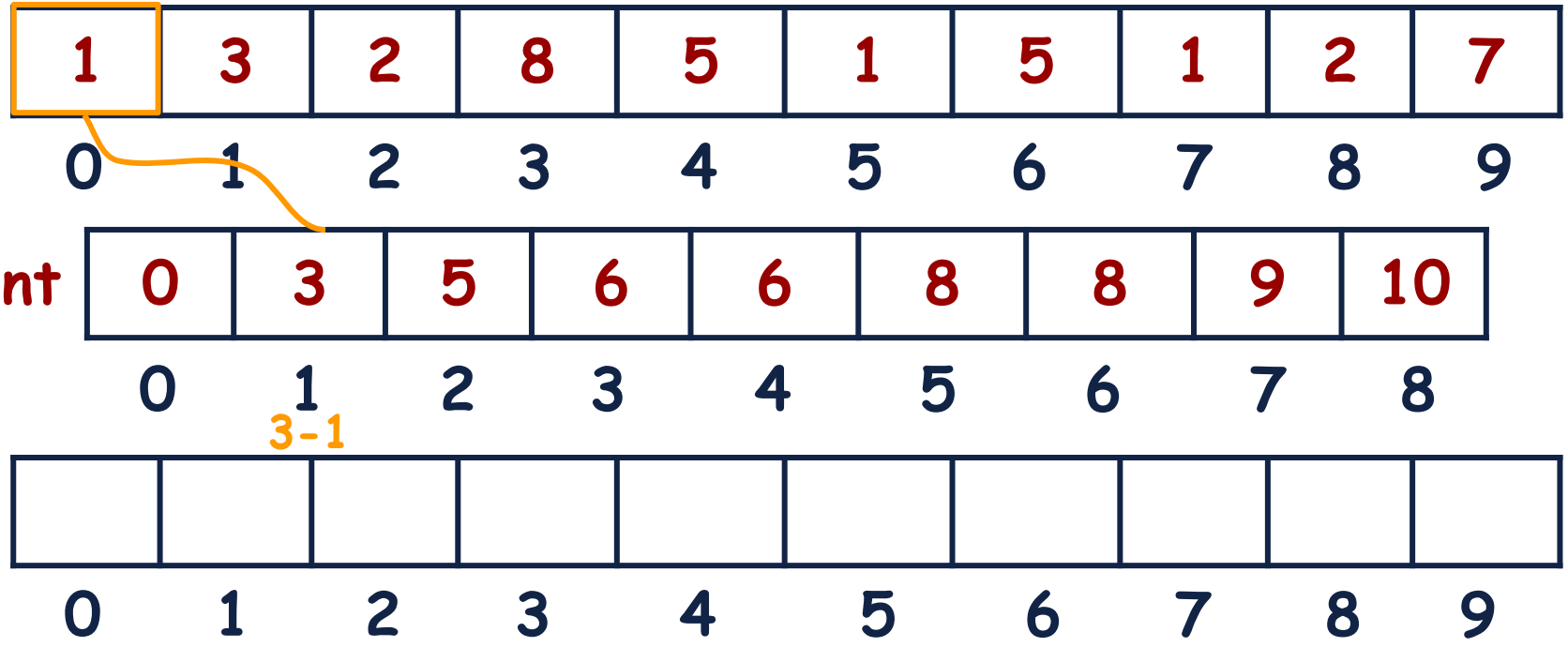
Counting Sort: Working

Step 6: Find the index of each element of the original array in the count array. Place the element at the index in output array by subtracting -1



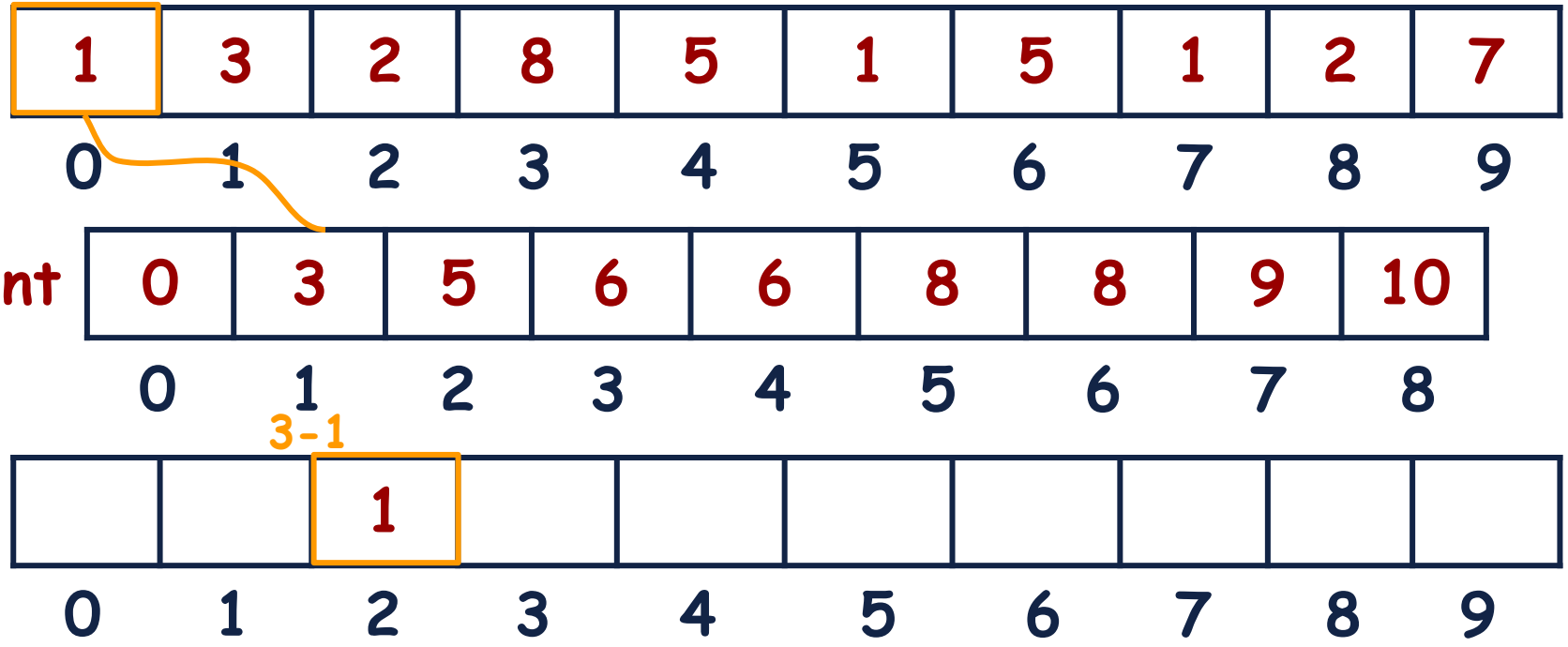
Counting Sort: Working

Step 6: Find the index of each element of the original array in the count array. Place the element at the index in output array by subtracting -1



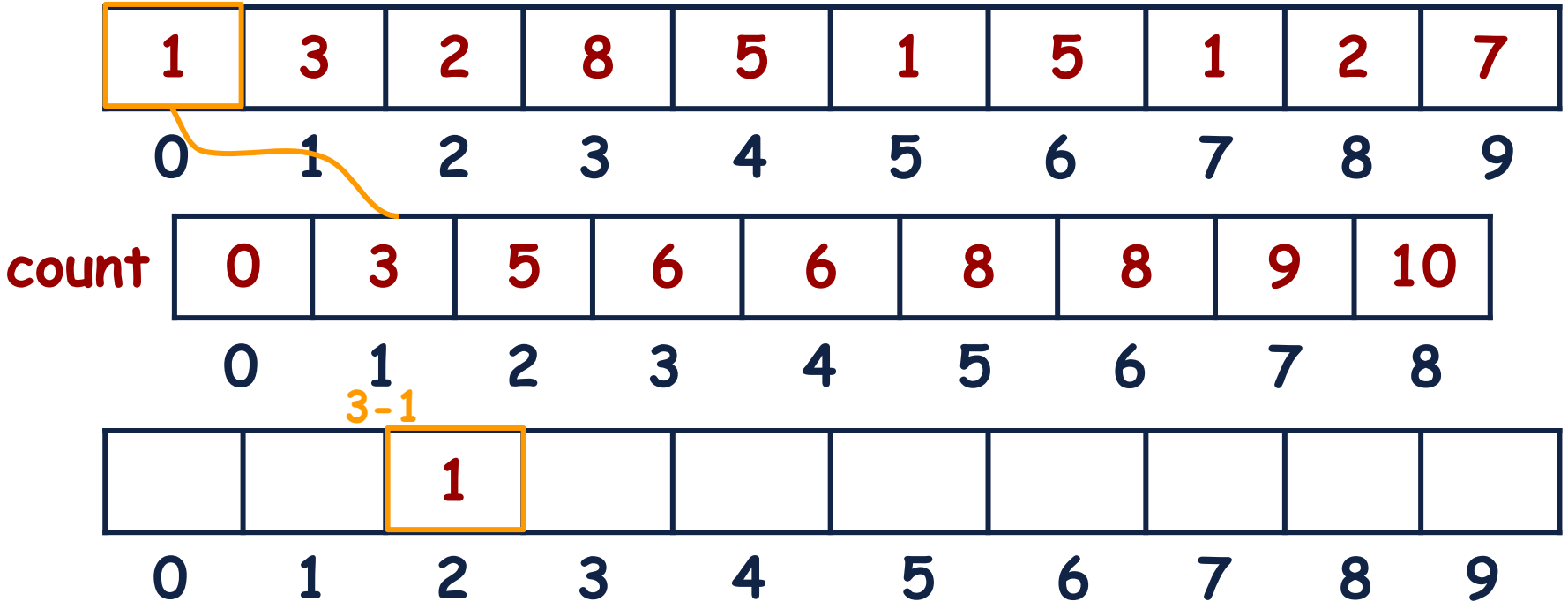
Counting Sort: Working

Step 6: Find the index of each element of the original array in the count array. Place the element at the index in output array by subtracting -1



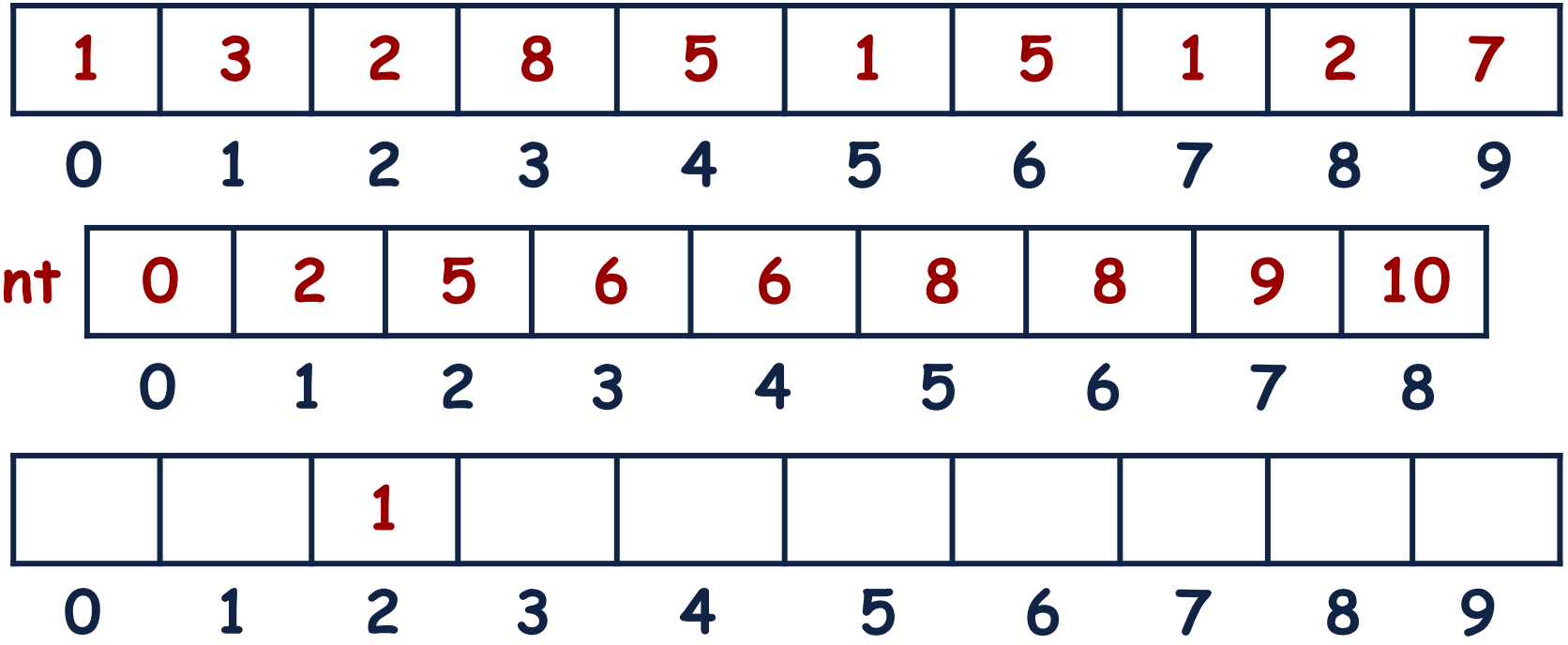
Counting Sort: Working

Step 7: After placing each element at its correct position in output array, decrease its count by one in count array.



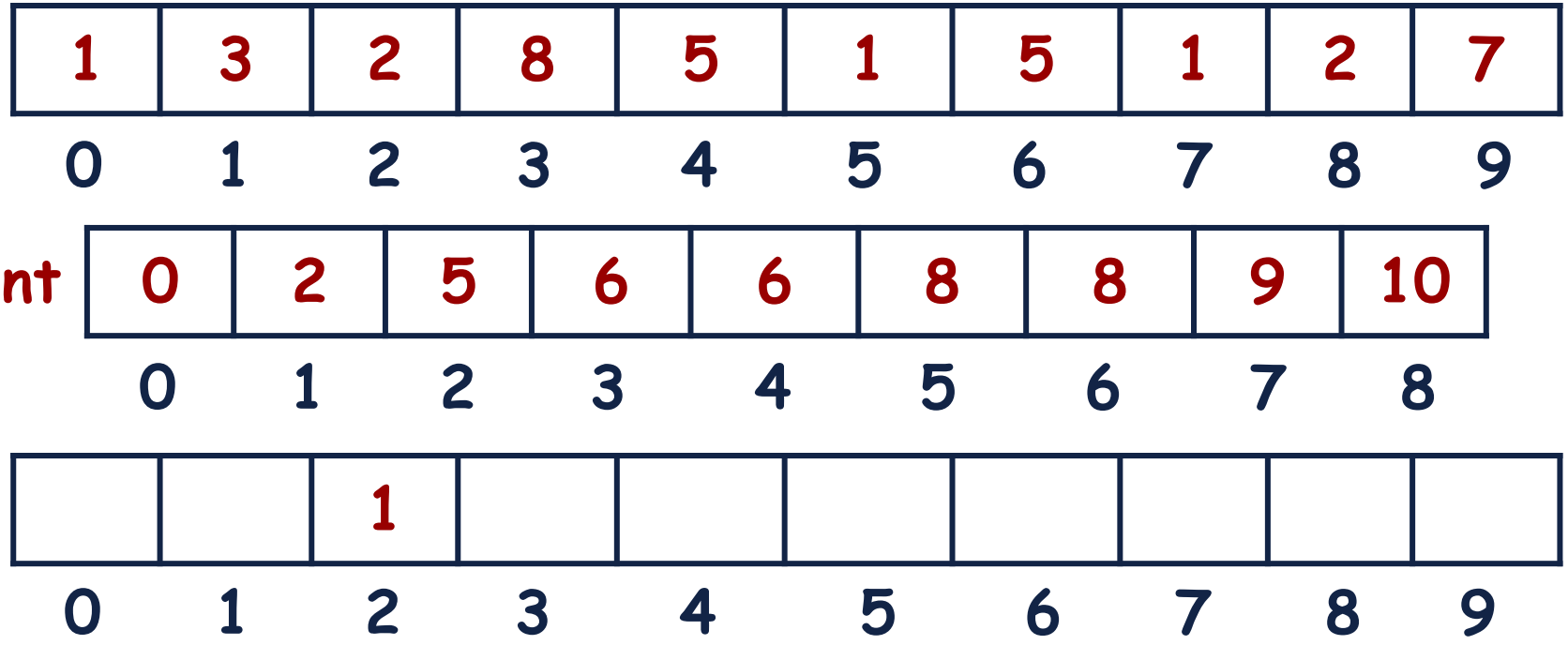
Counting Sort: Working

Step 7: After placing each element at its correct position in output array, decrease its count by one in count array.



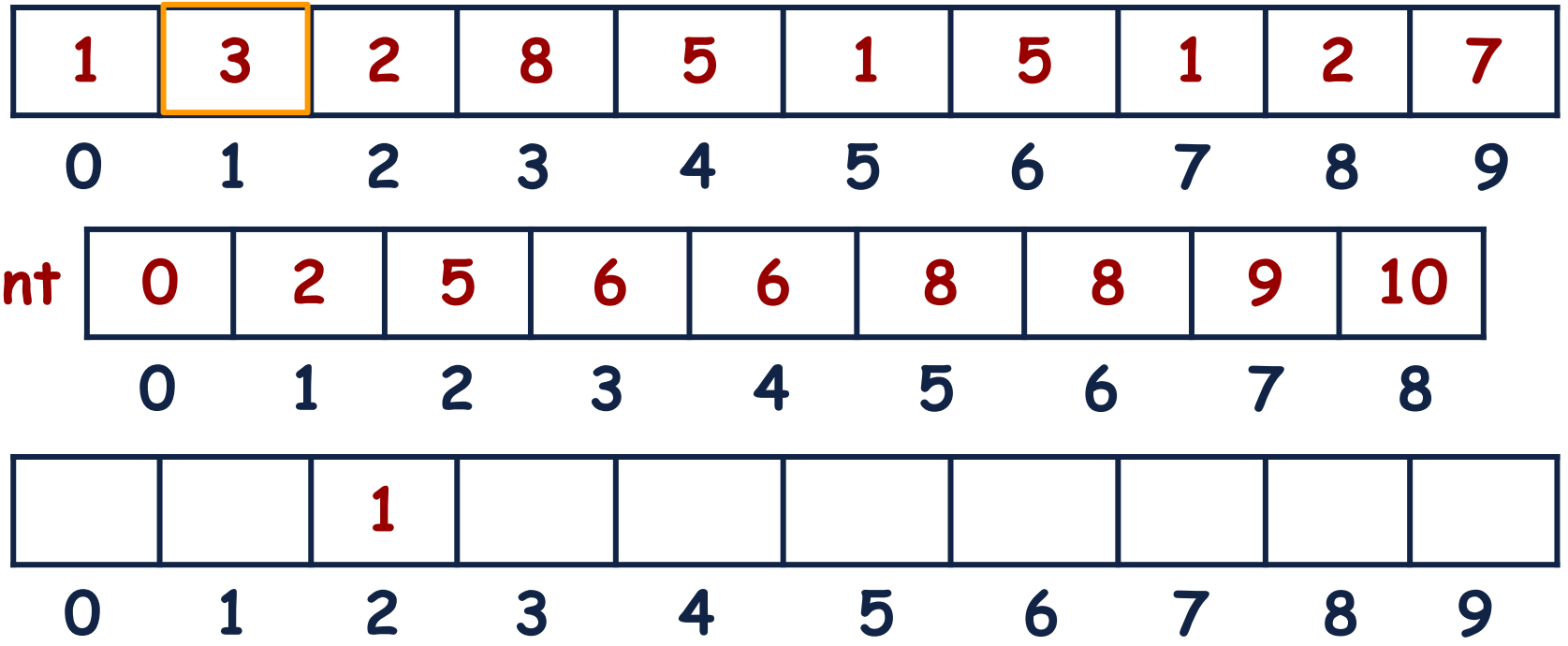
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



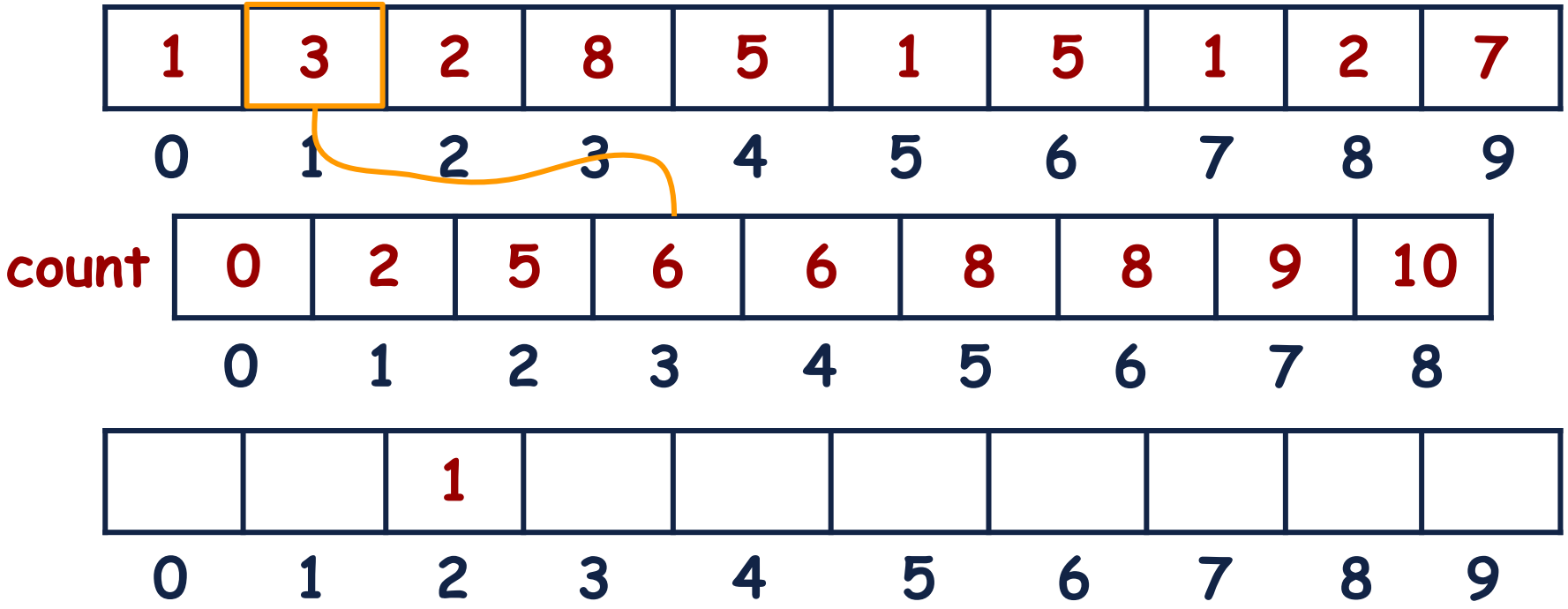
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



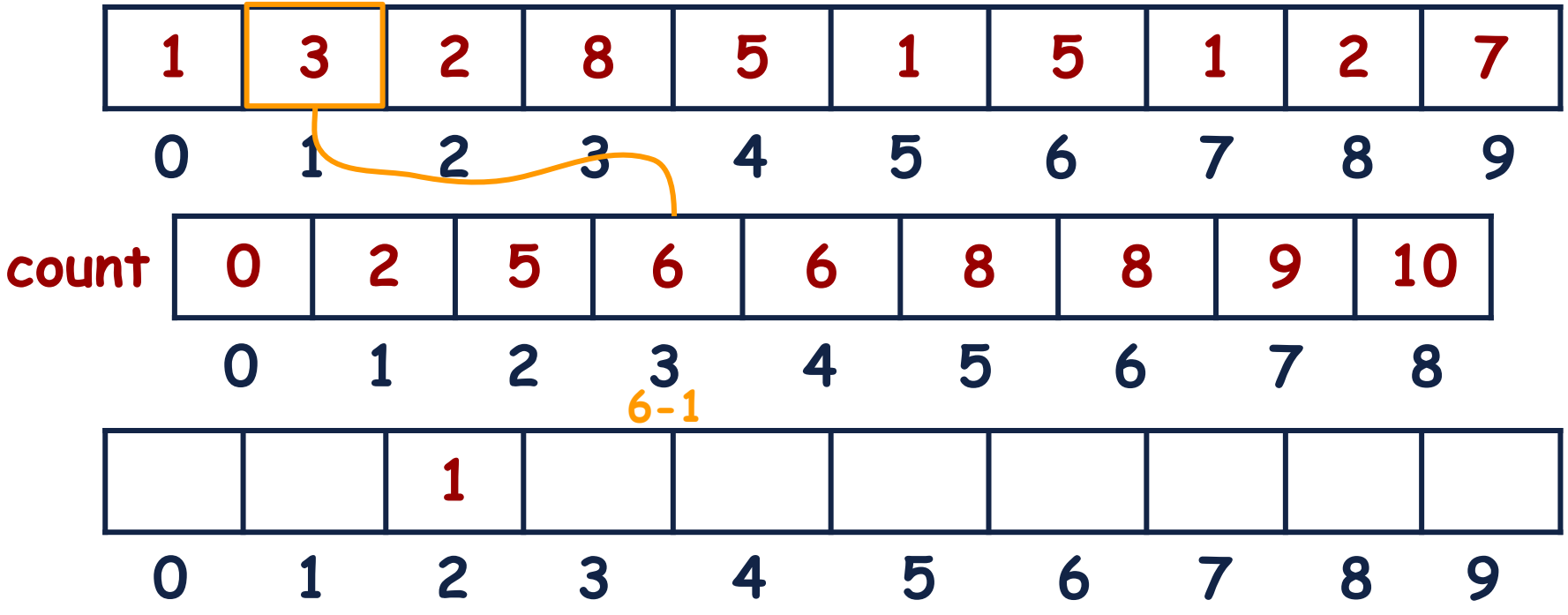
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



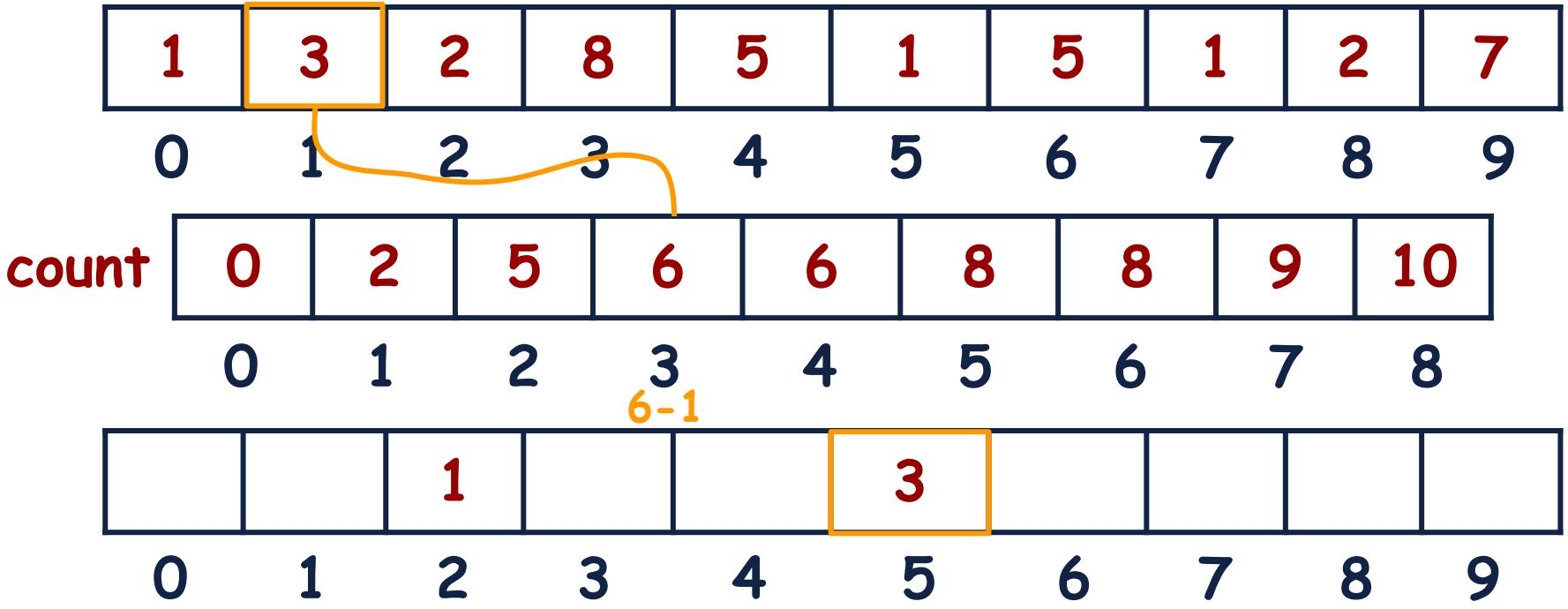
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



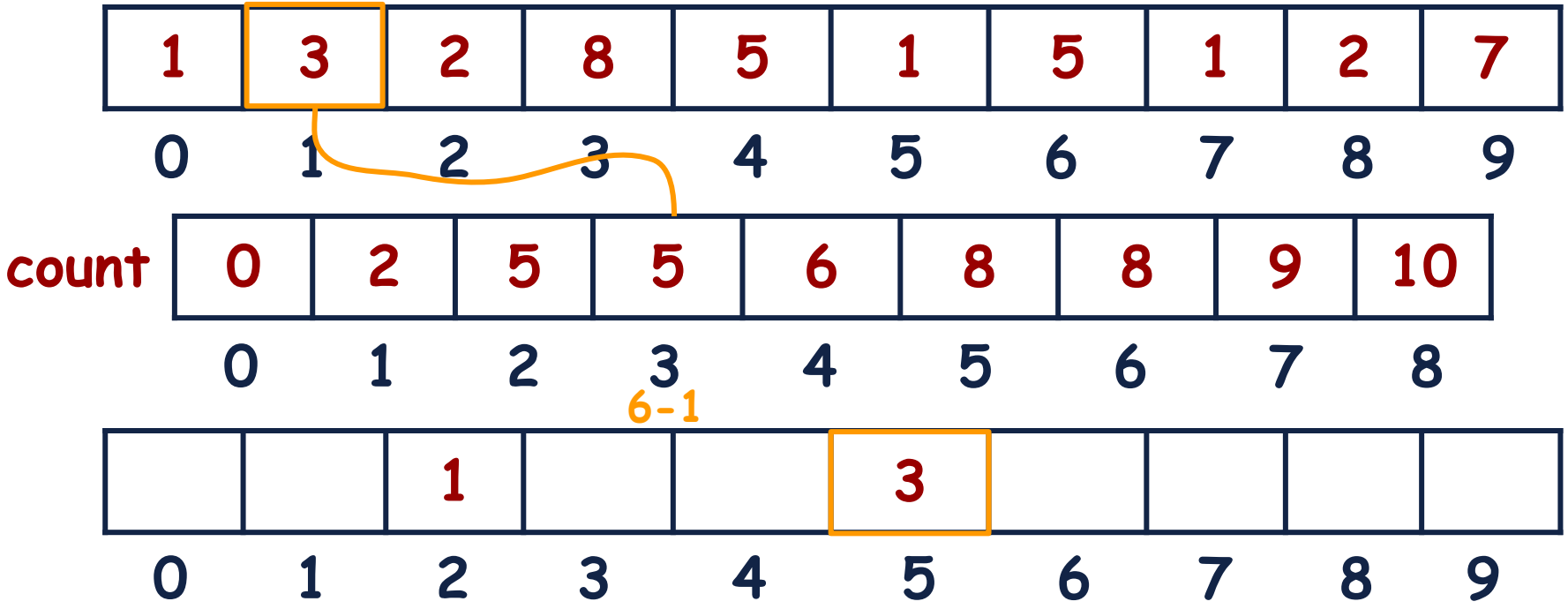
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



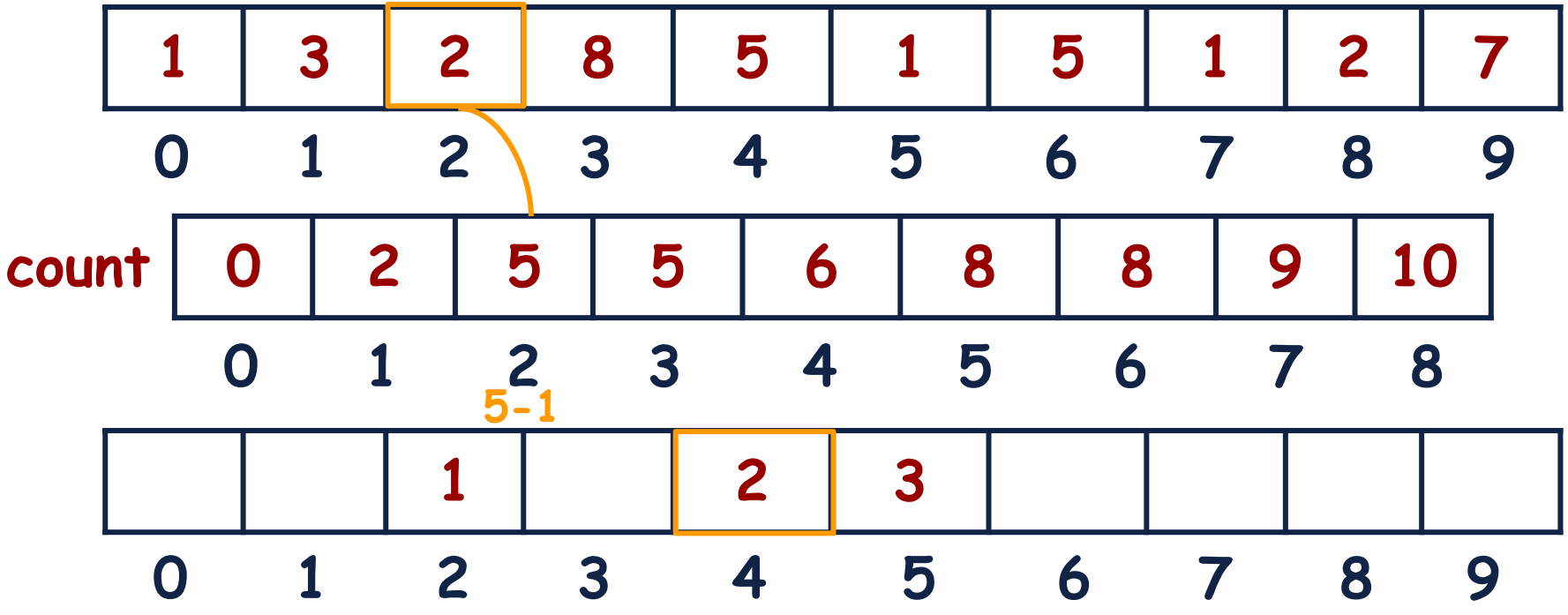
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



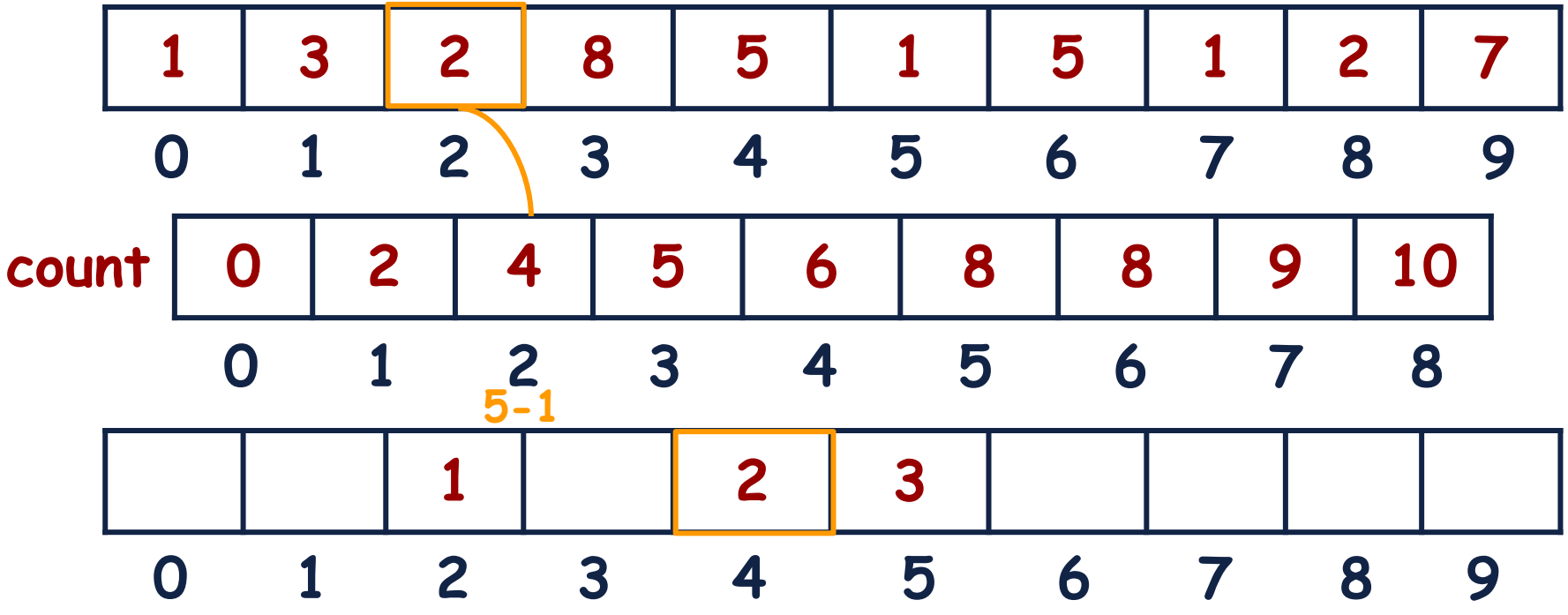
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



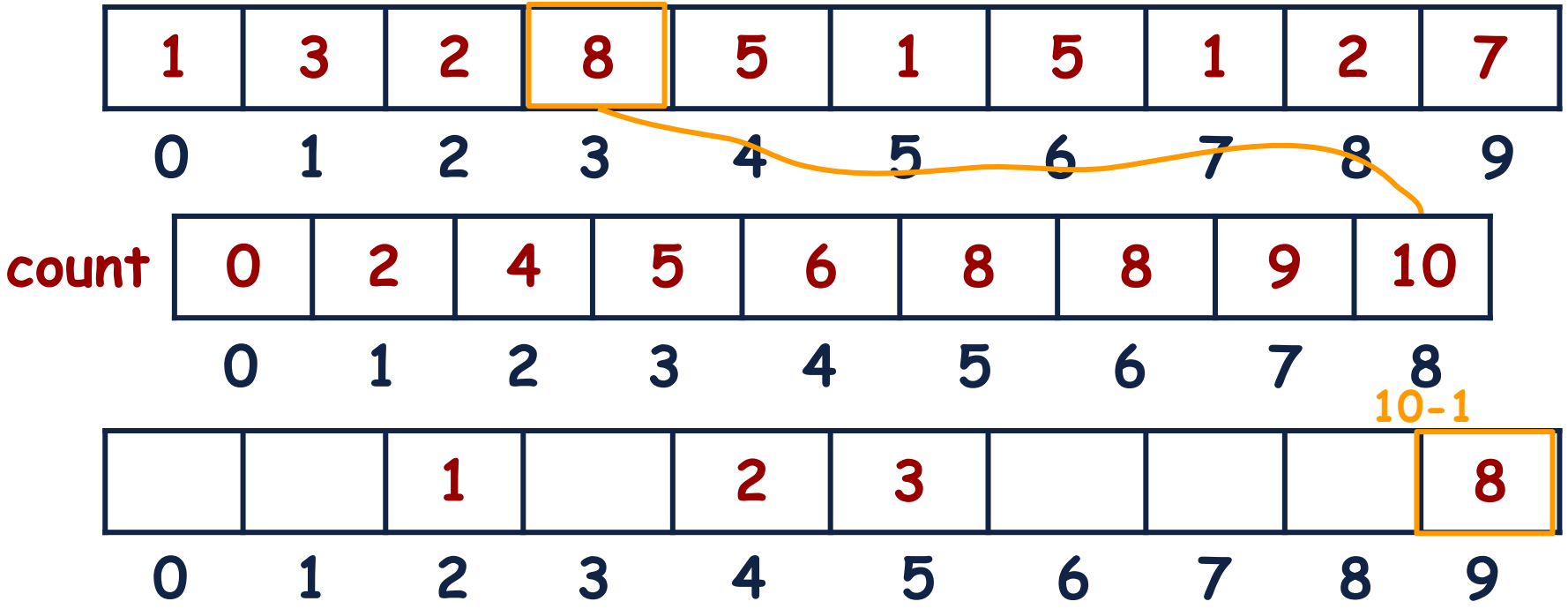
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



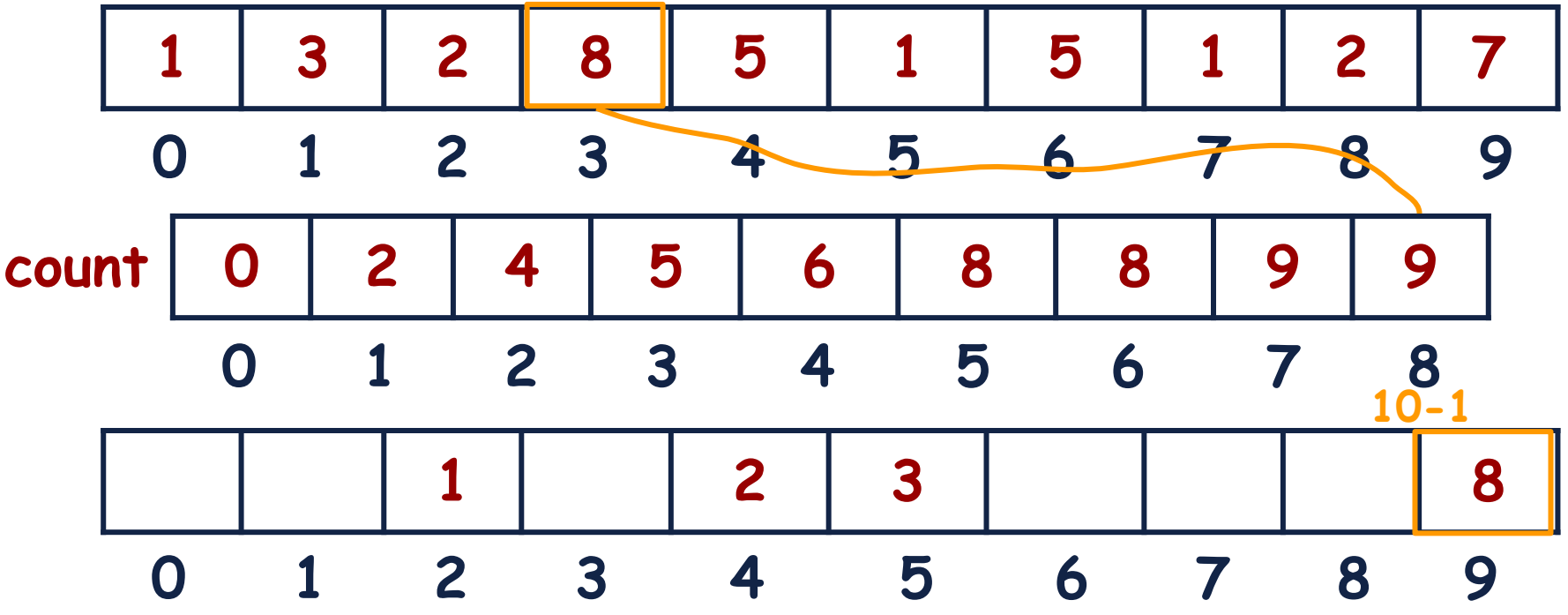
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



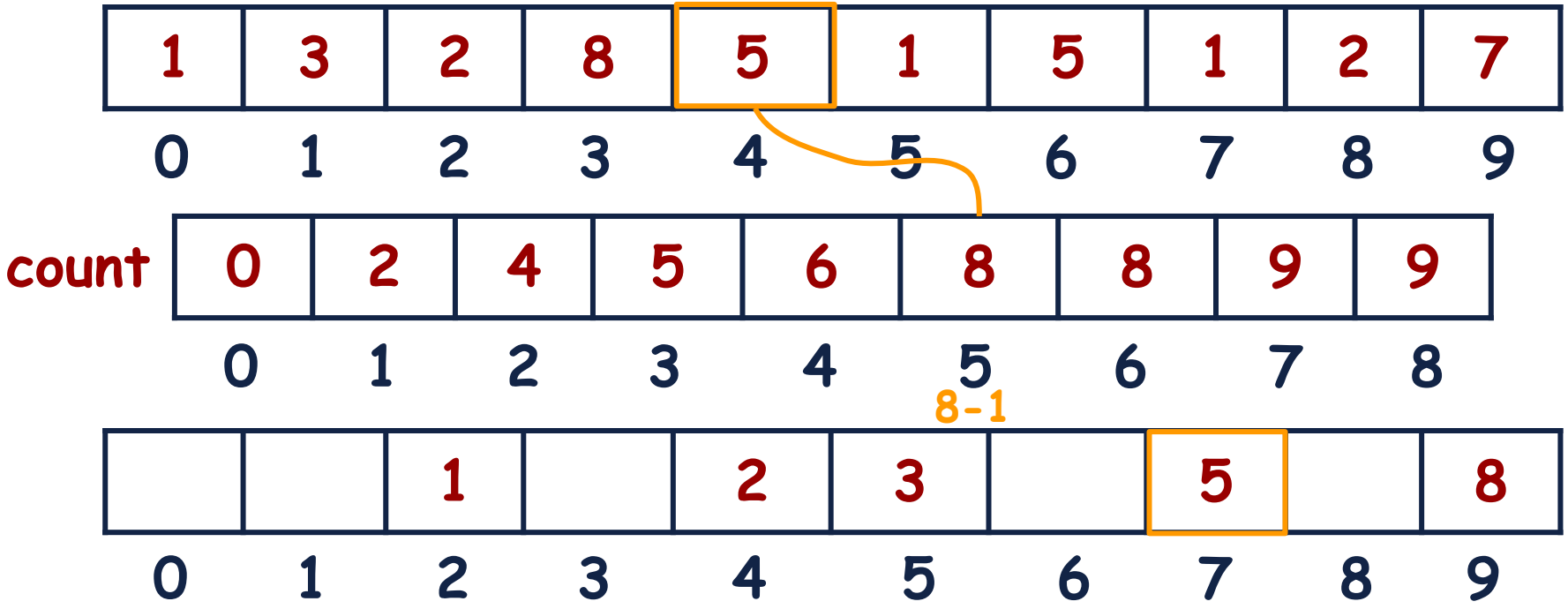
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



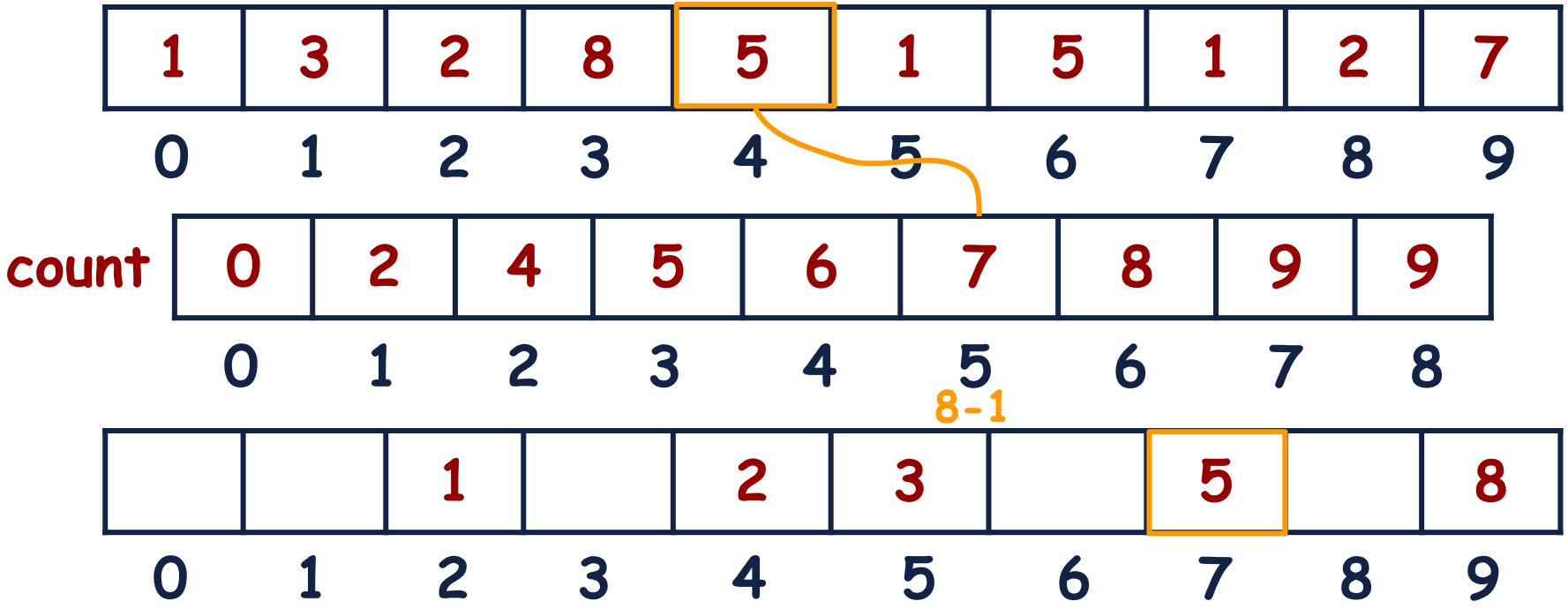
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



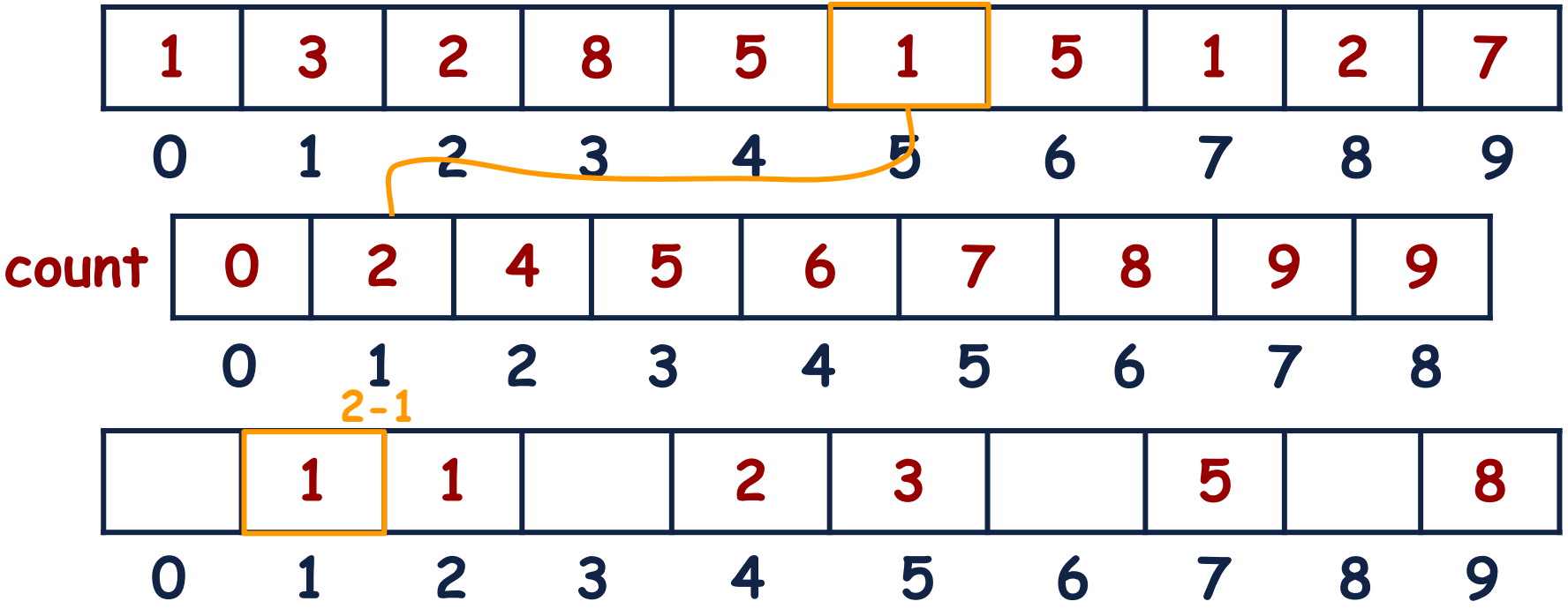
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



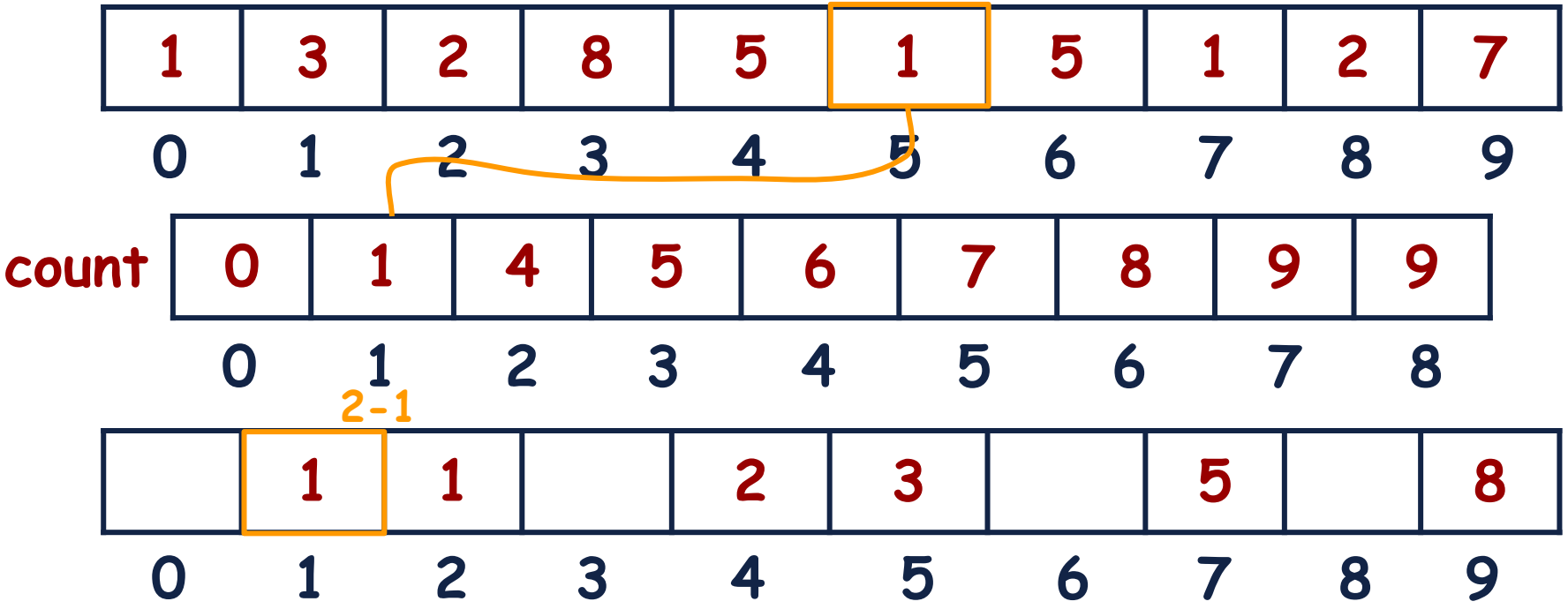
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



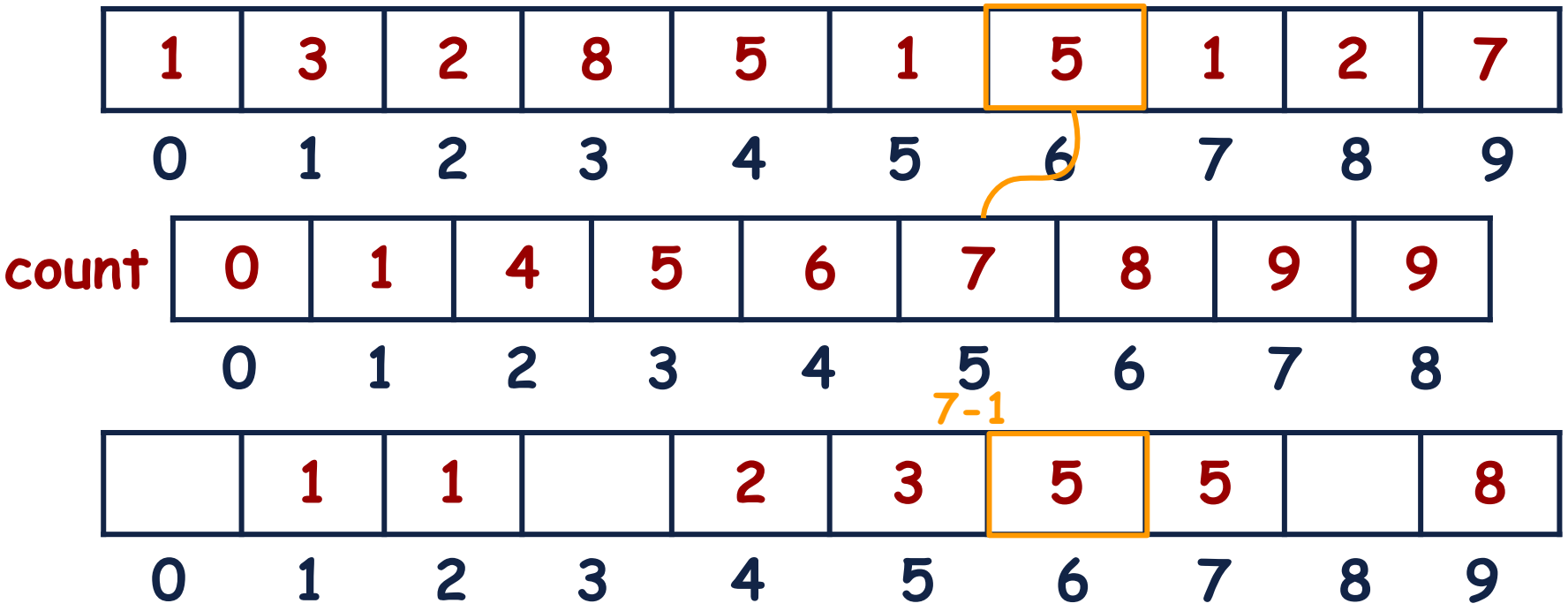
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



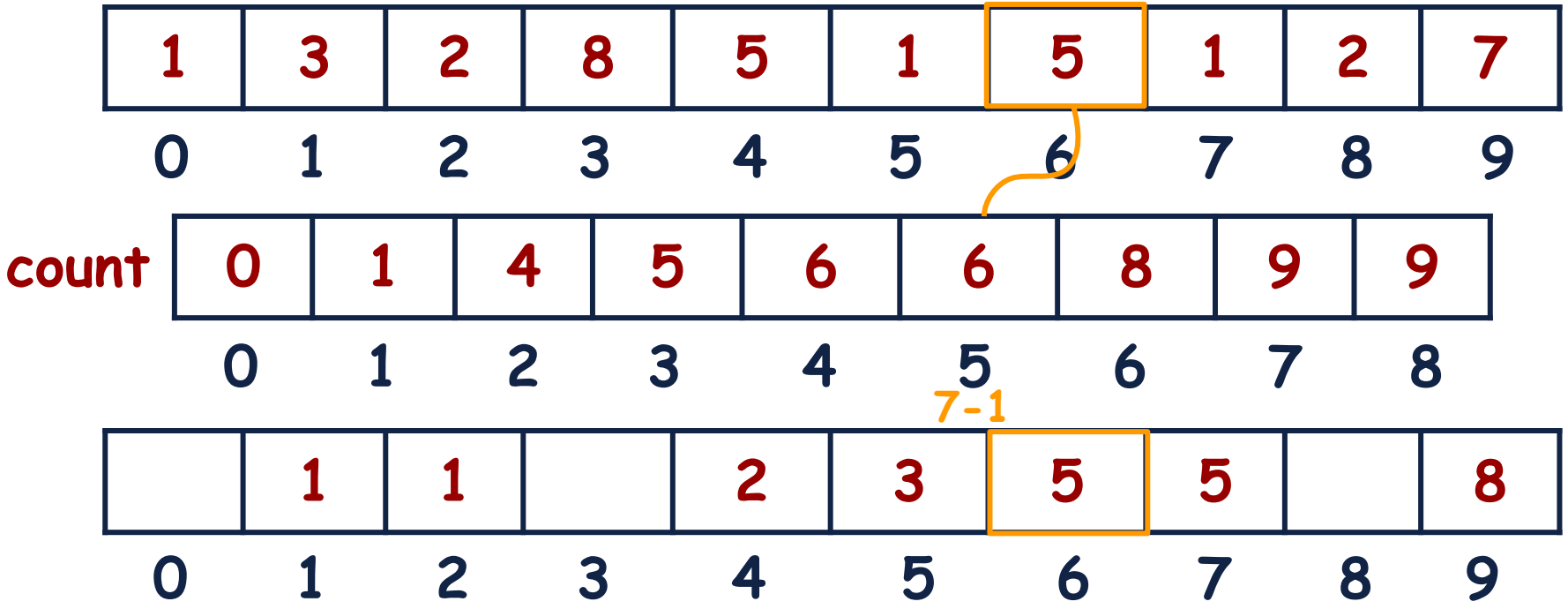
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



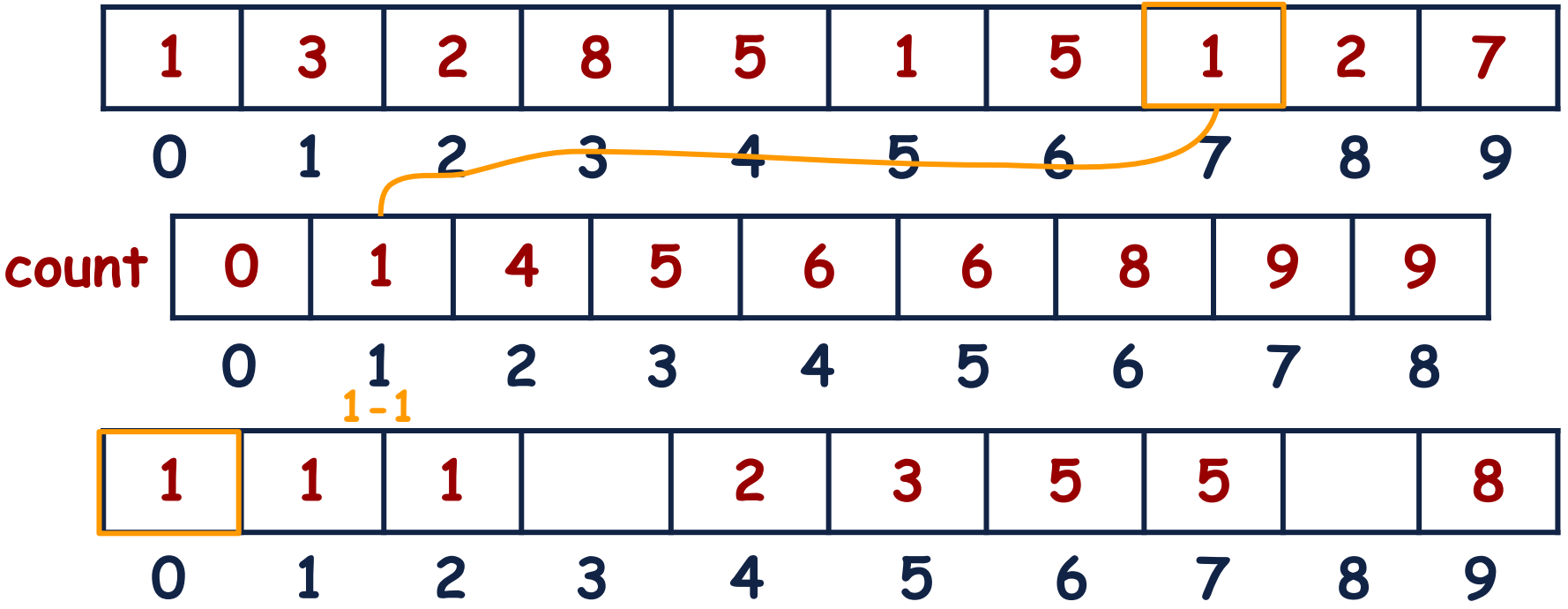
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



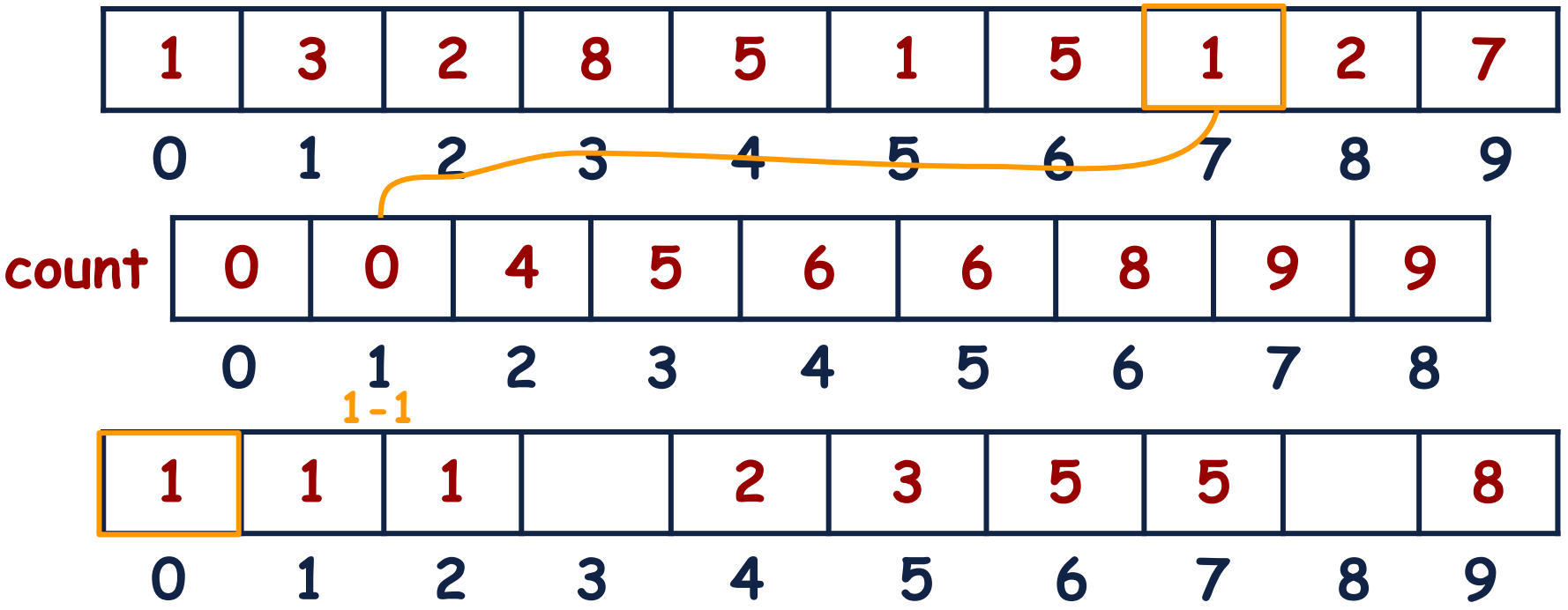
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



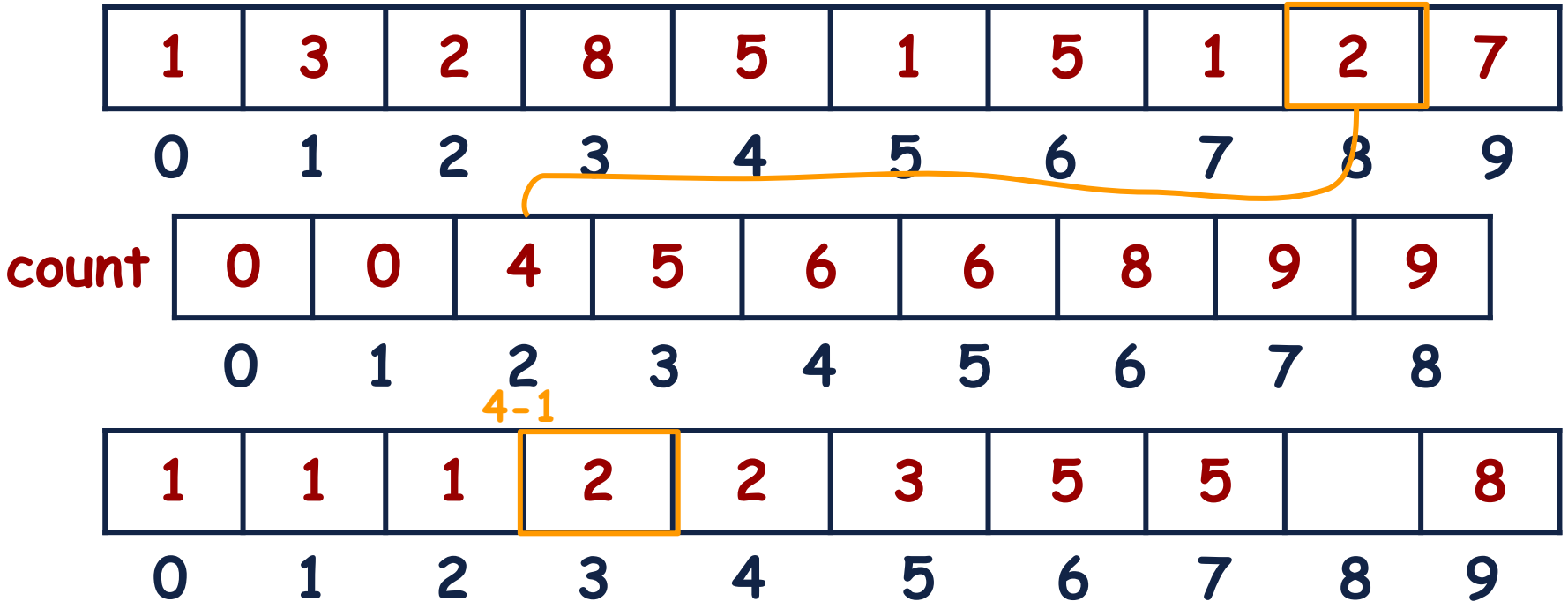
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



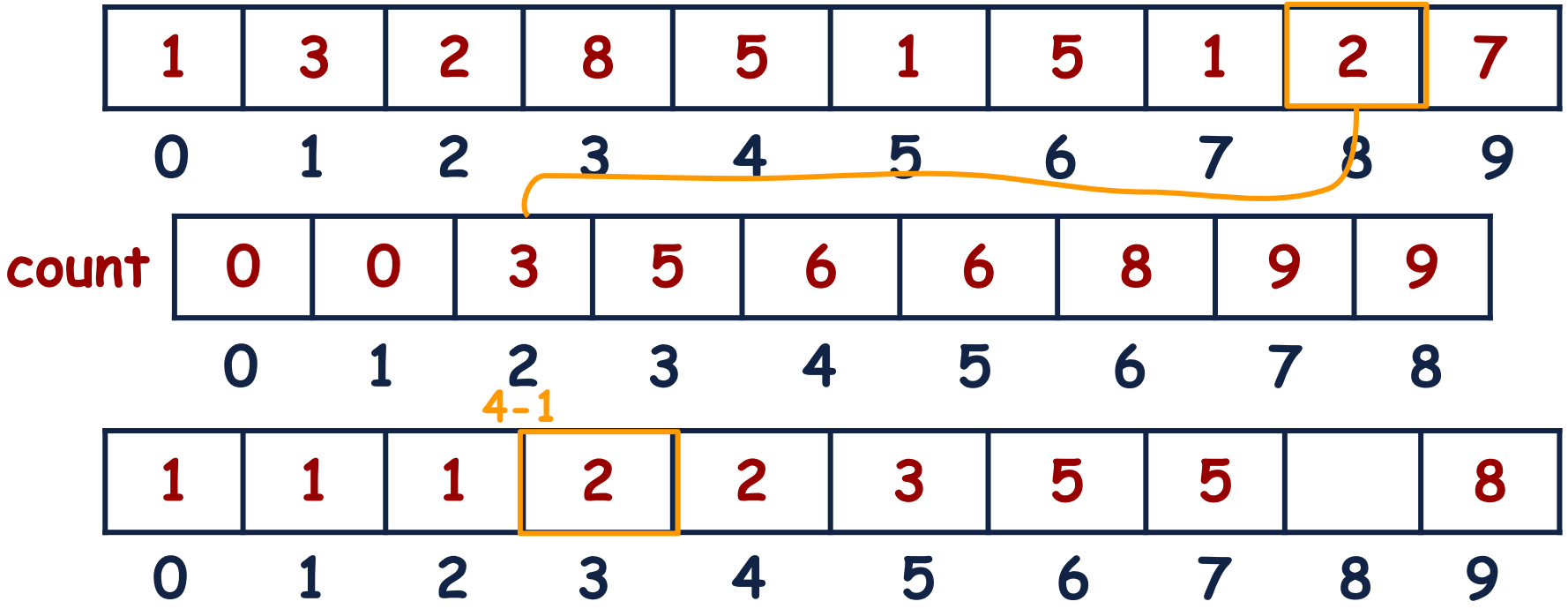
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



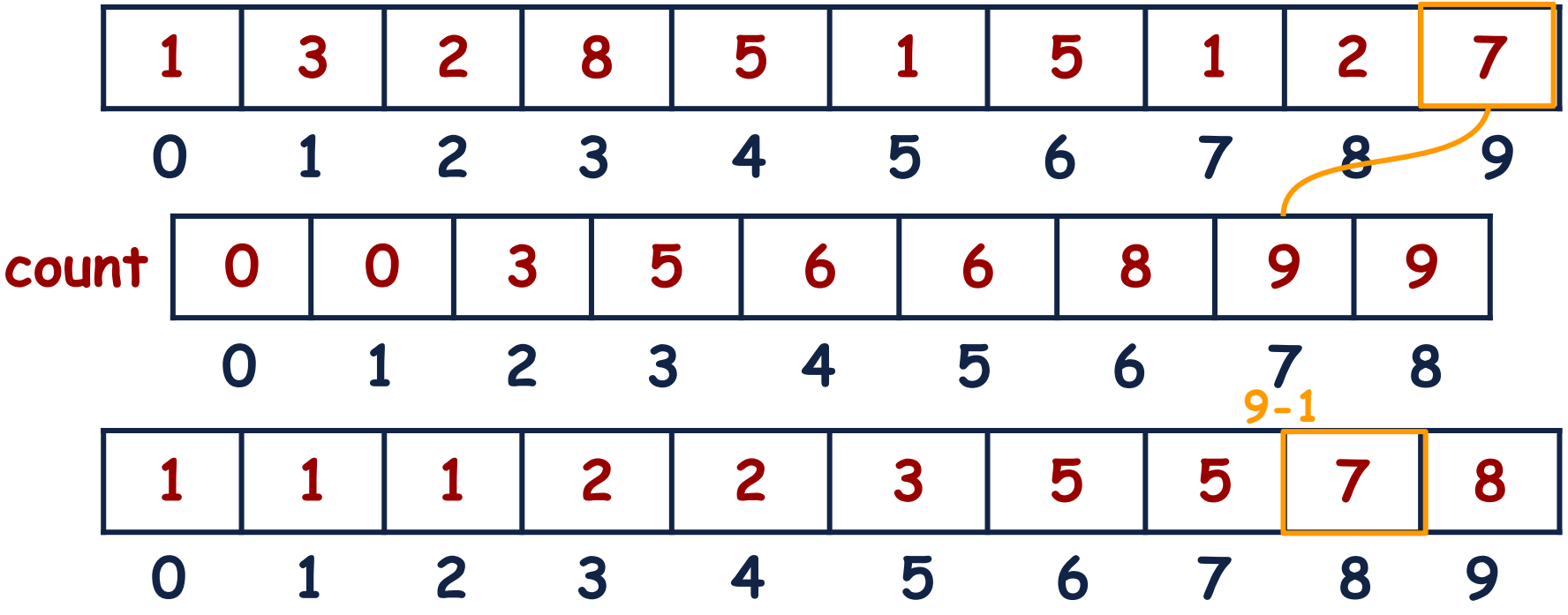
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



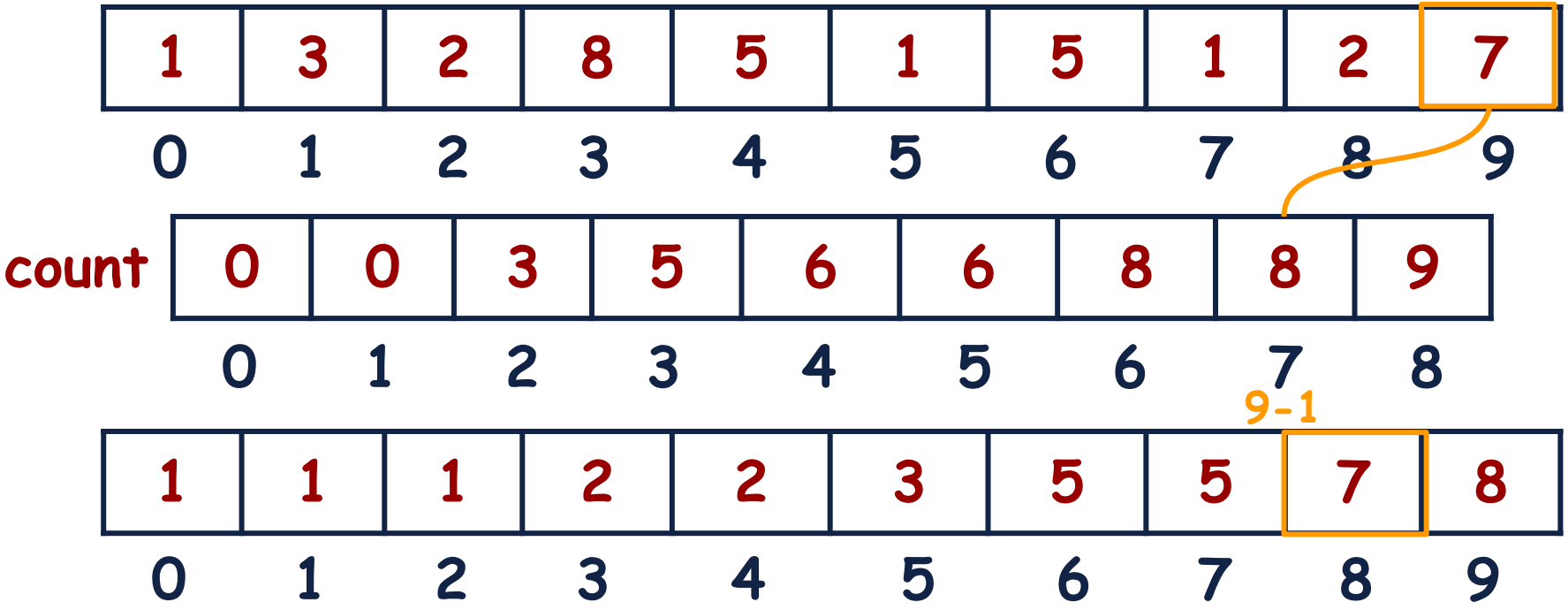
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



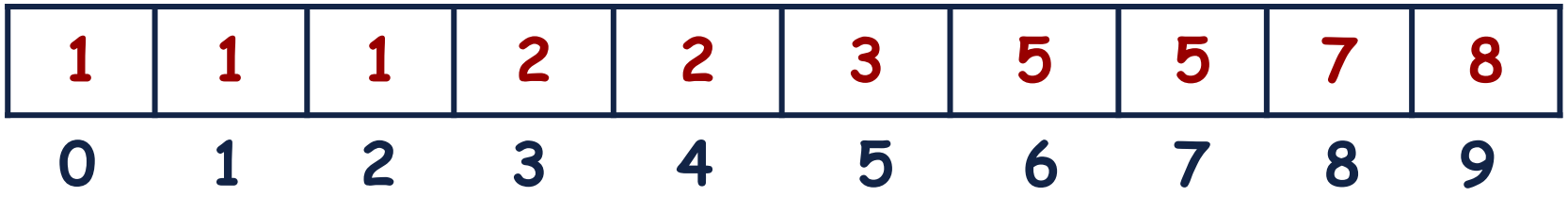
Counting Sort: Working

Step 8: Repeat the same process for all the elements.



Counting Sort: Working

Now, the data in the output array is sorted.



Counting Sort: Stable or Unstable

Now, the data in the output array is sorted.
Was it a Stable Sort or Unstable Sort?

1	1	1	2	2	3	5	5	7	8
0	1	2	3	4	5	6	7	8	9

Counting Sort: Stable or Unstable

Now, the data in the output array is sorted.
Was it a Stable Sort or Unstable Sort?

UNSTABLE

1	1	1	2	2	3	5	5	7	8
0	1	2	3	4	5	6	7	8	9

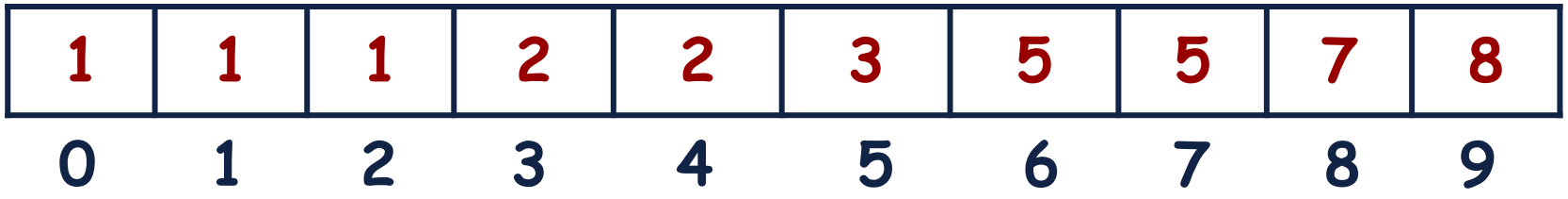
Counting Sort: Stable or Unstable

Now, the data in the output array is sorted.
Can we make it a Stable Sort?

1	1	1	2	2	3	5	5	7	8
0	1	2	3	4	5	6	7	8	9

Counting Sort: Stable

Now, the data in the output array is sorted.
Start iterating from the end of the array.



Counting Sort: Implementation

Now, Let's implement the counting sort Algorithm.

Counting Sort: Implementation

```
main()
{
    vector<int> arr = {1, 3, 2, 8, 5, 1, 5, 1, 2, 7};
    countingSort(arr);
    for(int x = 0; x < arr.size(); x++)
    {
        cout << arr[x] << " ";
    }
}
```

Counting Sort

```
void countingSort(vector<int> &arr)
{
    int max = *max_element(arr.begin(), arr.end());
    vector<int> count(max + 1);
    vector<int> output(arr.size());
    for (int x = 0; x < arr.size(); x++)
    {
        count[arr[x]]++;
    }
    for (int x = 1; x < count.size(); x++)
    {
        count[x] = count[x - 1] + count[x];
    }
    for (int x = arr.size() - 1; x >= 0; x--)
    {
        int index = count[arr[x]] - 1;
        count[arr[x]]--;
        output[index] = arr[x];
    }
    for (int x = 0; x < output.size(); x++)
    {
        arr[x] = output[x];
    }
}
```

Counting Sort

What is the Time Complexity of this Algorithm?

```
void countingSort(vector<int> &arr)
{
    int max = *max_element(arr.begin(), arr.end());
    vector<int> count(max + 1);
    vector<int> output(arr.size());
    for (int x = 0; x < arr.size(); x++)
    {
        count[arr[x]]++;
    }
    for (int x = 1; x < count.size(); x++)
    {
        count[x] = count[x - 1] + count[x];
    }
    for (int x = arr.size() - 1; x >= 0; x--)
    {
        int index = count[arr[x]] - 1;
        count[arr[x]]--;
        output[index] = arr[x];
    }
    for (int x = 0; x < output.size(); x++)
    {
        arr[x] = output[x];
    }
}
```

Non-Comparison Sorting Algorithms

Non-Comparison Sorting Algorithm	Time Complexity			Space Complexity
	Best Case	Average Case	Worst Case	Worst Case
Counting Sort	O(N + K)	O(N + K)	O(N + K)	O(N+K)

https://www.geeksforgeeks.org/counting-sort/

Non-Comparison Sorting Algorithms

Sorting Algorithm	In-Place	Stable
Counting Sort	No	Yes

Learning Objective

Students should be able to **apply** sorting using non-comparison based sorting algorithm.

Self Assessment

1. https://leetcode.com/problems/height-checker/
2. https://leetcode.com/problems/h-index/