Counting Sort (Non Comparison Sort)

Counting sort is an efficient, linear-time sorting algorithm that works by counting how many times each element of a set occurs to determine how the set should be ordered.

By avoiding the comparisons that have been a part of the sorting methods e.g. Insertion Sort, Merge Sort, Quick Sort etc., counting sort improves on the O(n lg n) runtime bound of comparison sorts.

Aside from being fast, an important virtue of counting sort is that it is stable. Stable sorts leave elements that have equal values in the same order as they appear in the original set.

`{10, 7, 12, 4, 9, 13}`

Min = 4, Max = 13, Range = Max – Min + 1 = 13 – 4 +1 = 10

10	7	12	4	9	13

Count Array – Stores the Count of number of occurrences of an element

```
int range = max - min + 1;
int count[] = new int[range];
```

10 – 4 = 6

7 – 4 = 3

12 – 4 = 8

4 – 4 = 0

9 – 4 = 5

13 – 4 = 9

C4	C5	C6	C7	C8	C9	C10	C11	C12	C13
0	1	2	3	4	5	6	7	8	9

(a)

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

(b)

`count[arr[i] - min] = count[arr[i] - min] + 1; (Meat of counting sort)`

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

(c) - Count Array

```
count[i] = count[i] + count[i - 1];
```

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

(d) - Count Array - Summing the counts

(e) – Output Array

```
output[count[arr[i] - min] - 1] = arr[i];
count[arr[i] - min] = count[arr[i] - min] - 1;
```

output[count[6] – 1] = output [4 – 1] = **output [3] = arr[0] = 10**

output[count[3] – 1] = output [2 – 1] = **output [1] = arr[1] = 7**

output[count[8] – 1] = output [5 – 1] = **output [4] = arr[2] = 12**

output[count[0] – 1] = output [1 – 1] = **output [0] = arr[3] = 4**

output[count[5] – 1] = output [3 – 1] = **output [2] = arr[4] = 9**

output[count[9] – 1] = output [6 – 1] = **output [5] = arr[5] = 13**

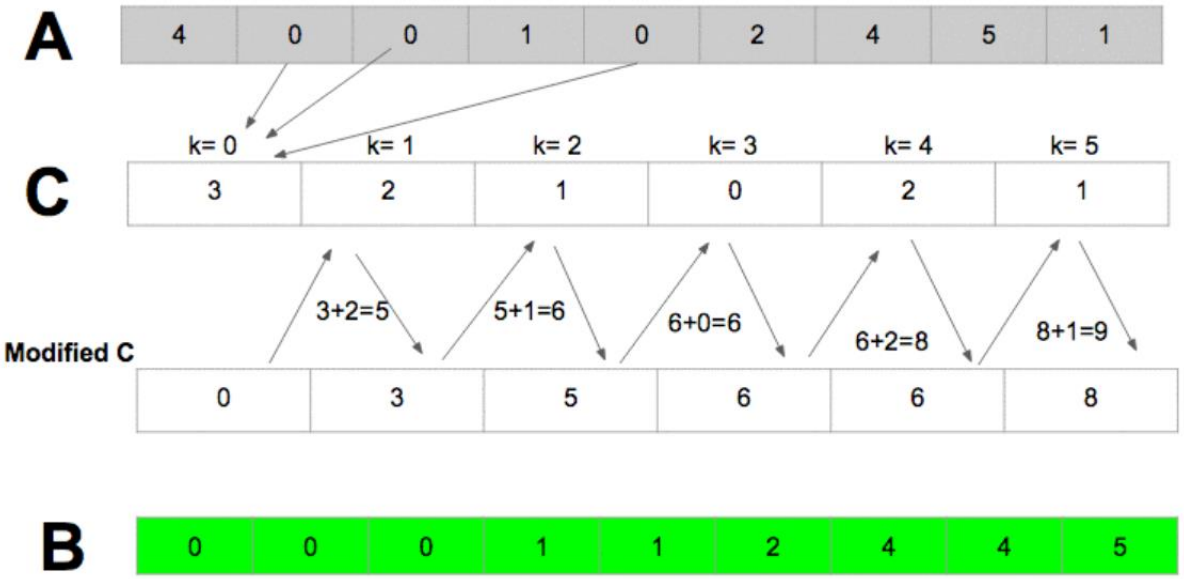
4	**7**	**9**	**10**	**12**	**13**

(f) – Output Array

Note :

Output array's size will be equal to size of input array, n

Count array's size will be equal to range of input/keys k



Key Points :

1. Counting sort is only efficient if the range of input data, k , is not significantly greater than the number of elements to be sorted, n

   Counting sort will not be a good choice for sorting below input :

   A = {10, 5, 10000, 5000}

2. The first loop goes through input array, which has n elements. So first loop has O(n) running time. The second loop iterates over count array with k elements, so this step has a running time of O(k). The third loop iterates through input array again, so third step has a running time of O(n). That is why Counting sort has O(k+n) running time.

3. Counting Sort works only with integers or data that can be expressed in some integer form.