

# Theory: Counting sort in Java

🕒 17 minutes   0 / 3 problems solved

Skip this topic

Start practicing

303 users solved this topic. Latest completion was 1 day ago.

**Counting sort** is not a comparison-based algorithm for sorting small integers. The algorithm counts the number of occurrences of distinct numbers in the input array and then output the values from the minimum to the maximum according to their counts. To store counts, the algorithm uses another array.

The algorithm can be **stable** and **unstable**. We will consider both versions written in Java.

## §1. Implementing unstable counting sort

Given below is an implementation of unstable counting sort in Java:

```
1 public static int[] countingSort(int[] numbers) {
2     int maxVal = 10; // we suppose the maximum is 10
3     int k = maxVal + 1; // the length of the array containing counts
4
5     int[] counts = new int[k]; // it stores 11 zeros with indexes from 0 to k-1
6
7     /* in this loop we count distinct numbers in the input array */
8     for (int i = 0; i < numbers.length; i++) {
9         counts[numbers[i]]++;
10    }
11
12    int pos = 0; // a position in the numbers array
13
14    /* in this loop we modify the input array to make it sorted */
15    for (int num = 0; num < k; num++) { // get the next element
16
17        int count = counts[num]; // get the count of the element
18
19        while (count > 0) {
20
21            numbers[pos] = num; // write it in the numbers array
22
23            pos++;
24
25            count--;
26        }
27    }
28
29    return numbers;
30 }
```

We assumed that the maximum element is 10, but in fact, the best strategy can be to pass the possible maximum as a parameter to the method or to determine it by traversing the input array with the time complexity  $O(n)$ .

Here is an example of how the method can be used:

```
1 countingSort(new int[] {2, 0, 3, 5, 0, 3, 3, 1}); // [0, 0, 1, 2, 3, 3, 3, 5]
```

## §2. Implementing stable counting sort

Current topic:

[Counting sort in Java](#) ...

Topic depends on:

✓ [Counting sort](#) ...

✗ [Algorithms in Java](#) ...

Table of contents:

[1 Counting sort in Java](#)

[§1. Implementing unstable counting sort](#)

[§2. Implementing stable counting sort](#)

[Feedback & Comments](#)

This algorithm also can be stable that may be required when sorting keys. To make it stable, we will calculate cumulative counts.

```

1 public static int[] stableCountingSort(int[] numbers, int max) {
2     int k = max + 1; // the length of the array containing counts
3
4     int[] counts = new int[k]; // it stores counts with indexes from 0 to k-1
5
6     for (int i = 0; i < numbers.length; i++) {
7         counts[numbers[i]]++;
8     }
9
10    for (int i = 1; i < counts.length; i++) {
11        counts[i] = counts[i - 1] + counts[i]; // cumulative counts
12    }
13
14    int[] sortedNumbers = new int[numbers.length];
15
16    for (int i = numbers.length - 1; i >= 0; i--) { // go through input array in backward
17
18        int rightmostIndex = counts[numbers[i]] - 1; // get the rightmost index
19
20        sortedNumbers[rightmostIndex] = numbers[i];
21
22        counts[numbers[i]]--; // decrease the index to calculate the previous occurrence
23    }
24
25    return sortedNumbers;
26 }

```

The method takes an array of ints and a possible maximum value as parameters. It returns a new sorted array.

Here is how this algorithm can be used:

```

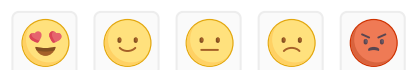
1 stableCountingSort(new int[] {2, 0, 3, 5, 0, 3, 3, 1}, 10); // [0, 0, 1, 2, 3, 3, 3, 5]

```

As you can see, it can sort the same array of ints. If you do not understand the algorithm, try to start it in the debug mode and sort different arrays.

 Report a typo

21 users liked this theory. 2 didn't like it. What about you?



Start practicing

[Comments \(0\)](#)

[Hints \(0\)](#)

[Useful links \(0\)](#)

[Show discussion](#)