Do you remember the Counting Sort algorithm from the Algorithmic Toolbox class? Here is its pseudocode with comments:

```
CountSort($A[1 \dots n]$)
$Count[1 \dots M] \leftarrow [0, \dots, 0]$
for $i$ from 1 to $n$:
    $Count[A[i]] \leftarrow Count[A[i]] + 1$
{$k$ appears $Count[k]$ times in $A$}
$Pos[1 \dots M] \leftarrow [0, \dots, 0]$
$Pos[1] \leftarrow 1$
for $j$ from 2 to $M$:
    $Pos[j] \leftarrow Pos[j-1] + Count[j-1]$
{$k$ will occupy range $[Pos[k] \dots Pos[k+1]-1]$}
for $i$ from 1 to $n$:
    $A'[Pos[A[i]]] \leftarrow A[i]$
    $Pos[A[i]] \leftarrow Pos[A[i]] + 1$
```

A simple, but crucial observation: it is ***stable***. It means that it keeps the order of equal elements. Of course, it doesn't matter for the sorting algorithm itself in what order to put equal elements: they can go in any order in the sorted array. But for some algorithms that ***use*** sorting it is important, as we will see in the following lecture. If you sort an array which has equal elements using Counting Sort, and one of the two equal elements was before another one initially in the array, it will still go first after sorting. Also see this answer for an example of difference between stable sorting and a non-stable sorting algorithms.

Note that we can sort not only integers using Counting Sort. We can sort any objects which can be numbered, if there are not many different objects. For example, if we want to sort characters, and we know that the characters have integer codes from 0 to 256, such that smaller characters have smaller integer codes, then we can sort them using Counting Sort by sorting the integer codes instead of characters themselves. The number of possible values for a character is different in different programming languages, so find out what is the range of integer codes for characters in your programming language of choice before using this in a Programming Assignment!

✓ Complete