Counting sort

Counting sort assumes that each of the n input elements is an integer in the range 0 to k, for some integer k. When k = O(n), the sort runs in Θ(n) time.

Counting sort determines, for each input element x, the number of elements less than x. It uses this information to place element x directly into its position in the output array. For example, if 17 elements are less than x, then x belongs in output position 18. We must modify this scheme slightly to handle the situation in which several elements have the same value, since we do not want to put them all in the same position.

In the code for counting sort, we assume that the input is an array arr[1 .. len], and thus arr.length = len. We require two other arrays: the array arrSorted[1 .. len] holds the sorted output, and the array aux[1 .. k] provides temporary working storage.

Time Complexity: O(n+k) where n is the number of elements in input array and k is the range of input.
Auxiliary Space: O(n+k)

Points to be noted:

1. Counting sort is efficient if the range of input data is not significantly greater than the number of objects to be sorted. Consider the situation where the input sequence is between range 1 to 10K and the data is 10, 5, 10K, 5K.
2. It is not a comparison based sorting. It running time complexity is O(n) with space proportional to the range of data.
3. It is often used as a sub-routine to another sorting algorithm like radix sort.
4. Counting sort uses a partial hashing to count the occurrence of the data object in O(1).
5. Counting sort can be extended to work for negative inputs also.

Exercise:

1. Modify above code to sort the input data in the range from M to N.
2. Modify above code to sort negative input data.
3. Is counting sort stable and online?
4. Thoughts on parallelizing the counting sort algorithm.