Counting sort is a sorting technique based on keys between a specific range. It works by counting the number of objects having distinct key values(kind of hashing). Then doing some arithmetic to calculate the position of each object in the output sequence.

1). Take a count array to store the count of each unique object

2). Modify the count array such that each element at each index stores the sum of previous counts.

The modified count array indicates the position of each object in the output sequence.

3). Output each object from the input sequence followed by decreasing its count by 1.

```
# alphabetical order
def countSort(arr):
    # the output character array that will have sorted
arr
    output = [0 for i in range(256)]

    # create a count array to store the count of
individual
    # characters and initialize count array as 0
    count = [0 for i in range(256)]

    # for storing the resulting answer since the string
is immutable
    ans = ["" for _ in arr]

    # store count of each character
    for i in arr:
        count[ord(i)] += 1

    # change count[i] so that count[i] now contains
actual
    # position of this character in output array
    for i in range(256):
        output[count[ord(arr[i])]-1] = arr[i]
        count[ord(arr[i])] -= 1

    # copy the output array to arr, so that arr now
    # contains sorted characters
    for i in range(len(arr)):
        ans[i] = output[i]
    return ans
```

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 sigficantly greater than the number of objects to be sorted.
2. It is not a comparision based sorting. Its 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 algorithms such as radix sort.
4. Counting sort uses a partial hashing to count the occurance of the data object in O(1).