Counting Sort

[Counting sort](http://en.wikipedia.org/wiki/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.

Let us understand it with the help of an example.

For simplicity, consider the data in the range 0 to 9.

Input data: 1, 4, 1, 2, 7, 5, 2

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

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

Count: 0 2 2 0 1 1 0 1 0 0

2) Modify the count array such that each element at each index

stores the sum of previous counts.

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

Count: 0 2 4 4 5 6 6 7 7 7

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.

Process the input data: 1, 4, 1, 2, 7, 5, 2. Position of 1 is 2.

Put data 1 at index 2 in output. Decrease count by 1 to place

next data 1 at an index 1 smaller than this index.

[**Recommended: Please solve it on “*PRACTICE* ” first, before moving on to the solution.**](https://practice.geeksforgeeks.org/problems/largest-even-number/0)

Following is implementation of counting sort.

* C++
* C
* Java
* Python
* C#
* PHP

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // C++ Program for counting sort  #include<bits/stdc++.h>  #include<string.h>  using namespace std;  #define RANGE 255    // The main function that sort  // the given string arr[] in  // alphabatical order  void countSort(char arr[])  {      // The output character array      // that will have sorted arr      char output[strlen(arr)];        // Create a count array to store count of inidividul      // characters and initialize count array as 0      int count[RANGE + 1], i;      memset(count, 0, sizeof(count));        // Store count of each character      for(i = 0; arr[i]; ++i)          ++count[arr[i]];        // Change count[i] so that count[i] now contains actual      // position of this character in output array      for (i = 1; i <= RANGE; ++i)          count[i] += count[i-1];        // Build the output character array      for (i = 0; arr[i]; ++i)      {          output[count[arr[i]]-1] = arr[i];          --count[arr[i]];      }        /\*      For Stable algorithm      for (i = sizeof(arr)-1; i>=0; --i)      {          output[count[arr[i]]-1] = arr[i];          --count[arr[i]];      }        For Logic : See implementation      \*/        // Copy the output array to arr, so that arr now      // contains sorted characters      for (i = 0; arr[i]; ++i)          arr[i] = output[i];  }    // Driver  code  int main()  {      char arr[] = "geeksforgeeks";        countSort(arr);        cout<< "Sorted character array is " << arr;      return 0;  }    // This code is contributed by rathbhupendra |

**Output:**

Sorted character array is eeeefggkkorss

**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)

The problem with the previous counting sort was that we could not sort the elements if we have negative numbers in it. Because there are no negative array indices. So what we do is, we find the minimum element and we will store count of that minimum element at zero index.

* C++
* Java

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| //Counting sort which takes negative numbers as well  #include <iostream>  #include <vector>  #include <algorithm>  using namespace std;    void countSort(vector <int>& arr)  {      int max = \*max\_element(arr.begin(), arr.end());      int min = \*min\_element(arr.begin(), arr.end());      int range = max - min + 1;        vector<int> count(range), output(arr.size());      for(int i = 0; i < arr.size(); i++)          count[arr[i]-min]++;        for(int i = 1; i < count.size(); i++)             count[i] += count[i-1];        for(int i = arr.size()-1; i >= 0; i--)      {           output[ count[arr[i]-min] -1 ] = arr[i];                count[arr[i]-min]--;      }        for(int i=0; i < arr.size(); i++)              arr[i] = output[i];  }    void printArray(vector <int> & arr)  {      for (int i=0; i < arr.size(); i++)          cout << arr[i] << " ";      cout << "\n";  }    int main()  {      vector<int> arr = {-5, -10, 0, -3, 8, 5, -1, 10};      countSort (arr);      printArray (arr);      return 0;  } |

**Output:**

-10 -5 -3 -1 0 5 8 10

**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.** Is counting sort stable and online?  
**3.**Thoughts on parallelizing the counting sort algorithm.