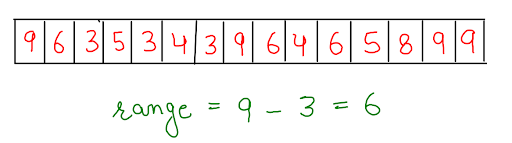
**1. PROBLEM DISCUSSION :**

Count sort outperforms the previously stated sorting algorithms in terms of time complexity.This sorting technique is mostly used when the input array's range doesn't vary greatly and the data is enormous. For example, in a competitive test with 100 marks in which 100000 students took part. In this case, the range is 100, but the n is 100000. (n here is the length of the array). How do we go about doing that? So count sort which we are going to discuss now can be used for sorting such kind of data. As with all sorting algorithms, We are provided with an integer array as input. We have to use count sort to sort the given array in ascending order. For example: Sample Input : 5, 7, -2, 4, 1, 3 Sample Output : -2, 1, 3, 4, 7

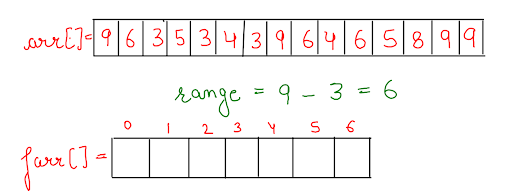
**2. APPROACH :**

Let’s understand the steps followed in sorting the array using count sort by taking the following example :

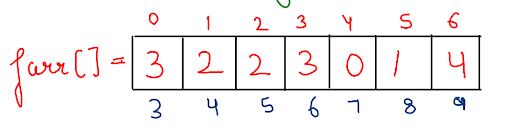
● Firstly, we have to find the minimum and maximum elements in the given array (arr[]). In our example, the minimum and maximum values are 3 and 9, respectively.

****

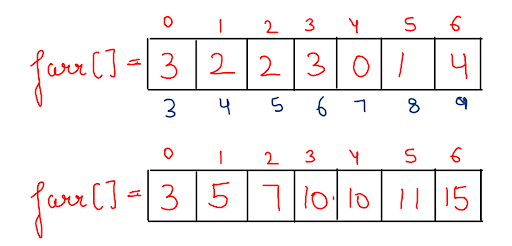
● Then we create a frequency array (farr[]), the length of this array is equal to the ‘range’.We define ‘range’ as the difference between the maximum and minimum element plus one, i.e. 7 in this instance ((9 - 3) + 1). This farr[] is used to store the frequency of each individual value of arr[]. We can observe that farr[0] refers to the frequency of the first and smallest element of range, which in this case is 3, and so on for the full farr[].

****

● Now, we traverse back over the original array arr[] to count the frequency of distinct elements and update farr[].

****

● When we've finished storing the frequencies of each element in farr[], we need to replace them with cumulative frequencies at each index, from 0 to last, and turn farr[] into a prefix sum array like this :

****

● We create a new answer array ans[], which will hold our sorted and stable array. To accomplish this, we must traverse the initial array arr[] in reverse order. While travelling arr[], we look for count in the prefix sum array corresponding to that element everytime we arrive at a specific element. We reduce the count by one and use that value as the index for the element we're looking at in the original array. And save that particular element in ans[] at this index.

****

● So at the end, in the ans[] array our final sorted and stable array gets stored.

Here, a question can arrive that after step 3, we could just apply a for loop to each element and print it for frequency times to print our output array. However, this is not a long-term solution. And if we do it that way, we won't be able to call this the Stable Sort. Let me tell you what exactly a stable sort is. Let's imagine there are six students with their grades, and when we sort them by grade, the one with the lowest grade will be at the top of the list. But what if two or more individuals have the same grade? Who gets to go first on the list then? Yes, the person who comes first on the list. Same in the case of a given array. If two or more common elements are present in the array, then in the sorted array that element must come first which is first present in the array. This is what we call Stable Sort.

**3. CODE :**

Note : Now the approach to solve this problem is clear to you. So before reading the Code, I recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

ConsoleJava

import java.io.\*;

import java.util.\*;

public class Main {

public static void countSort(int[] arr, int min, int max) {

int range = max - min + 1;

int[] ans = new int[arr.length];

//make frequency arr

int[] farr = new int[range];

for (int i = 0 ; i < arr.length; i++) {

farr[arr[i] - min]++;

}

//convert it into prefix sum array

for (int i = 1 ; i < farr.length; i++) {

farr[i] += farr[i - 1];

}

//stable sorting(filling ans array)

for (int i = arr.length - 1; i >= 0; i--) {

int pos = farr[arr[i] - min] - 1;

ans[pos] = arr[i];

farr[arr[i] - min]--;

}

//filling original array with the help of ans array

for (int i = 0 ; i < arr.length; i++) {

arr[i] = ans[i];

}

}

public static void print(int[] arr) {

for (int i = 0; i < arr.length; i++) {

System.out.println(arr[i]);

}

}

public static void main(String[] args) throws Exception {

Scanner scn = new Scanner(System.in);

int n = scn.nextInt();

int[] arr = new int[n];

int max = Integer.MIN\_VALUE;

int min = Integer.MAX\_VALUE;

for (int i = 0; i < n; i++) {

arr[i] = scn.nextInt();

max = Math.max(max, arr[i]);

min = Math.min(min, arr[i]);

}

countSort(arr, min, max);

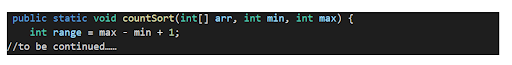
print(arr);

}

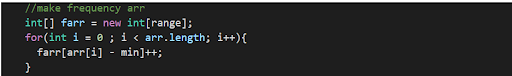
}

**4. CODE DISCUSSION :**

● First, using both these values min and max, we find range and using range we find length of our frequency array, farr[].

****

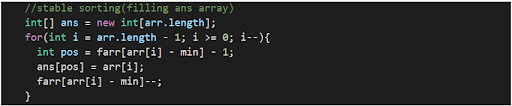
● Then we define our frequency array and fill farr[] by traversing arr[] and as soon as we are at arr[i], we increment corresponding frequency count (i.e. farr[arr[i] - min]++) in farr[].

****

● Then we convert this frequency array into a prefix sum array, which stores the count of all the smaller elements plus the count of elements to which it's corresponding to.

****

● Then we define an answer array, ans[] of the same length as of arr[]. And fill that while we traverse arr[] in reverse order. We use the element of arr[] as data for ans[] and count the prefix sum array corresponding to this element with a decrement of 1 as index to store that data in ans[]. We also need to decrement the frequency count of the prefix sum array corresponding to this element so that next time we visit it we store this value at a unique position.

****

● After filling our ans[] array in a stable and sorted way, we copy these values in similar order, to our original array[].

****

● Now, our array gets sorted in increasing order.

For more clarity of the question, watch the question video

Play Video

**5. ANALYSIS :**

Time Complexity :

O(n + k) We travelled arr[] and farr[] two times. (n+k) + (n+k) =2 (n+k). which can be expressed as O(n+k)

Space Complexity :

O(n+k) Two arrays one farr[] of k length and another ans[] of n length were used.

Hope that you liked the article on “ Count Sort”. Subscribe to Pepcoding’s youtube channel for more such amazing content on Data Structures & Algorithms and follow the resources available for all students in the resources section of Pepcoding’s website or in the content section on NADOS. You can suggest any improvements to the article on our telegram Channel, in the Nados doubt section or on the youtube channel’s comment section.