(Counting-Sort Tutorial)

Faiaz Amin Khan
Roll: SH-40 (2nd Year)
Department of Computer Science and Engineering,
University of Dhaka

May 2019

Contents

1 Introduction

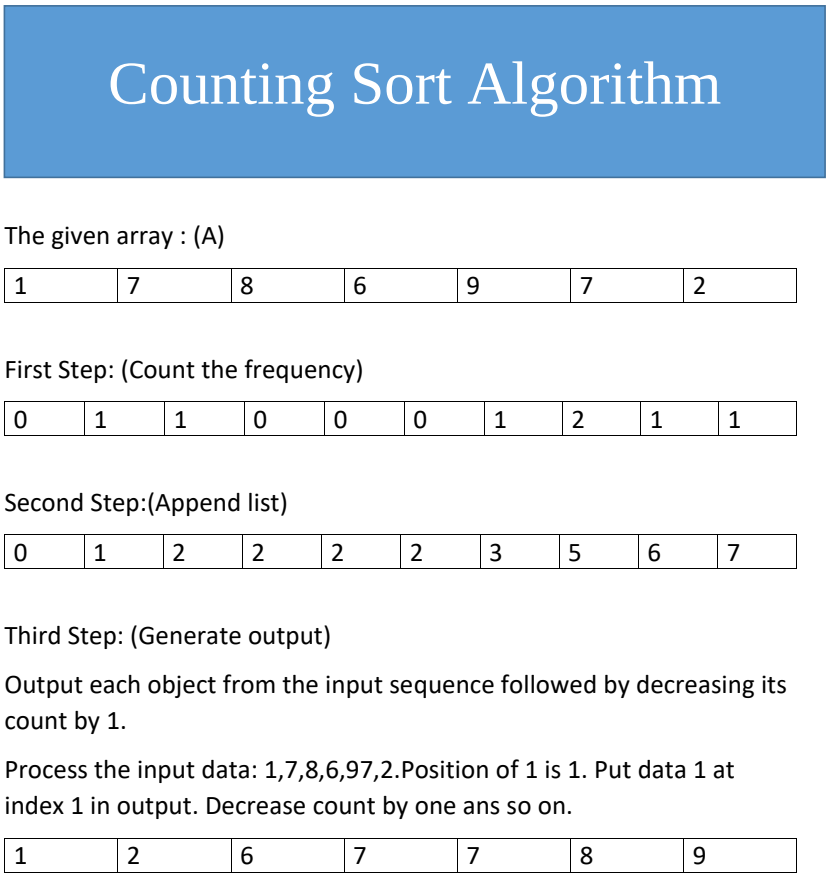
counting sort is a linear time sorting algorithm.It is a sorting technique based on keys between a specific range. It is done by counting the number of elements having unique key values . Then doing some arithmetic to calculate the position of each object in the output sequence.Counting sort is not a comparison sort rather than integer sort.Like other comparison sort such as merge sort, it does not compare key value. It just count the frequency and calculates the position in the sorted list.Counting Sort is very useful when variation in keys is not significantly greater than the number of items.It is often used as a subroutine in another sorting algorithm.
Let's assume that, array A of size N needs to be sorted.

- Initialize the temporary array temp as 0
- Traverse array A and store the count of occurrence of each element in the appropriate index of the temp array, which means, execute **temp**[**A**[**i**]]++ for each i , where i ranges from 0 to N-1 .
- Initialize the empty array final
- Place every element in the correct position of final fom temp.

2 Description

At first, find the maximum value in the given array **A**. Then declare a counter array **temp** and count the frequency of each element in the given array **A**. After that append each index value of temp to its next index. Now every value in the temp is indicating the desired position of its index of the given array **A**. Finally, take another array **final** and put all element in the array by traversing the **A**.

Let us consider the following image



It can be seen from the above image that there is an array. For simplicity the largest element here is 9. At very first step we counted the frequency of every element in the given .Then we modified the counter array such that every index value is equal to the sum of its own and its previous index.Now

every value is the correct position of its index in sorted list. Output each object from the input sequence followed by decreasing its count by 1. Process the input data: 1,7,8,6,9,7,2. Position of 1 is 1. Put data 1 at index 1 in output. Again Position of 7 is 6. Put data 7 at index 6 in output. Decrease count by 1 to place next data 7 at an index 5 smaller than this index.

3 Implementation

An implementation of counting sort is given here.

3.1 Pseudo Code

```
   CountingSort(A,n)
   //search for maximum element
   k= maximum element of A
  //A[]-- Initial Array to Sort
  //Complexity: O(k)
  for i = 0 to k do
  c[i] = 0

  //Storing Count of each element
  //Complexity: O(n)
  for j = 0 to n do
  c[A[j]] = c[A[j]] + 1

  // Change C[i] such that it contains actual
  //position of these elements in output array
  ////Complexity: O(k)
  for i = 1 to k do
  c[i] = c[i] + c[i-1]

  //Build Output array from C[i]
  //Complexity: O(n)
  for j = n-1 downto 0 do
  B[ c[A[j]]-1 ] = A[j]
  c[A[j]] = c[A[j]] - 1
end func
```

3.2 Demo Code

A demo code using C++ language is shown here :

```
#include <bits/stdc++.h>
    using namespace std;

    void counting_sort(int* A,int len){
        int max=*max_element(A,A+len);
        int temp[max+1],final[len],i;
        for(i=0;i<=max;i++) temp[i]=0;

        for(i=0;i<len;i++) temp[A[i]]++;

        for(i=1;i<=max;i++){
            temp[i]=temp[i]+temp[i-1];
        }

        for(i=0;i<len;i++){
            final[temp[A[i]]-1]=A[i];
            temp[A[i]]--;
        }
        for(i=0;i<len;i++){
            A[i]= final[i];
        }
    }
     int main() {
        int arr[] = {12, 7, 5, 17, 42, 13};
        int arr_size = sizeof(arr)/sizeof(arr[0]);

        printf("Given array is \n");
        for(int i = 0; i < arr_size; i++)
            cout<<arr[i]<<" ";
        cout<<endl;
        counting_sort(arr, arr_size);

        printf("\nSorted array is \n");
        for(int i = 0 ; i < arr_size; i++)
            cout<<arr[i]<<" ";
        cout<<endl;
        return 0;
    }
```

4 Complexity Analysis

- Time complexity for counting sort for all cases (best, average, worst) is O(len+ max). As the algorithm traverse through n elements of array A. It again runs a loop to the maximum element of the given array
- Space complexity of merge sort using arrays is O(len+max). This is because two extra array is used in the algorithm.

5 Related articles and problems

- GeeksforGeeks
- Hackerearth
- Wikipedia
- Khan Academy