Information Technology University, Lahore
Department of Computer Science
CS243: Data Structures and Algorithms
Fall 2020

Course Instructor: Miss Momina Azam	***Dated: 31 oct, 2020***
Lab Engineer: Aqsa Khalid	***Semester: 3***
Session: 2019-2023	***Batch: BSCS2019***

Lab 5. Implementation and analysis of counting sort

Due: 31 oct, 2020

Name	**Roll No**	**Lab Marks / 20**	**Viva Marks / 5**	**Total Marks / 25**
Muhammad Muneeb Ur Rahman	BSCS19057			

Checked on: ________________________________

Signature: ________________________________

Instructions:

- Make sure to read instructions carefully before attempting questions.
- Each question carries 5 marks.
- Make separate code files for each question and name them like "**Q1.cpp**","**Q2.cpp**" etc. The code for respective questions should also be copied in the space provided.
- Write your comments on methodology and results. Mere reporting results is not useful. **Insightful comments carry marks**.
- You are allowed to discuss with fellow students and with TA for general advice, but you must submit **your** own work.
- Plagiarism will not be tolerated and will be dealt according to "**Academic Code of Conduct**".
- We will not just be looking for correct answers rather higher grades will be awarded to solutions that demonstrate a clear understanding of the material. Style matters and will be a factor in the grade.
- All code files and this manual should be in a folder. Name this folder with your roll number e.g. **Lab1_BSCS19001**. Upload the zipped folder in *Google Classroom*. **We do not receive labs through email**.
- Late submission is subject to very valid reasons for example serious illness or tragedy.

Theory:

Counting sort is in a class of algorithms called 'no comparison' sorting algorithms.

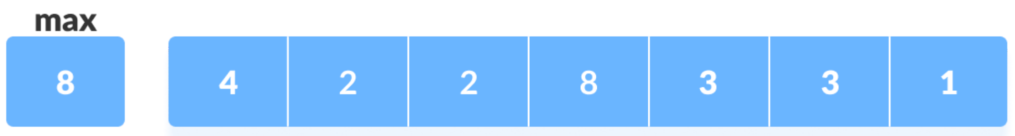
Top of Form

Bottom of Form

Counting sort is a sorting algorithm that sorts the elements of an array by counting the number of occurrences of each unique element in the array. The count is stored in an auxiliary (Other) array and the sorting is done by mapping the count as an index of the auxiliary array.

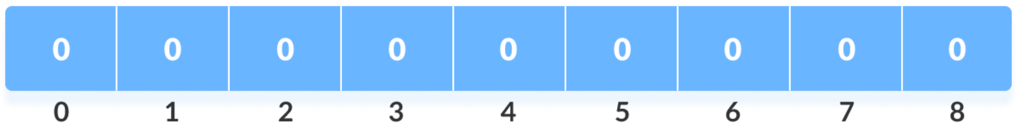
1.1 How Counting Sort Works?

1. Find out the maximum element (let it be `max`) from the given array.



Given array

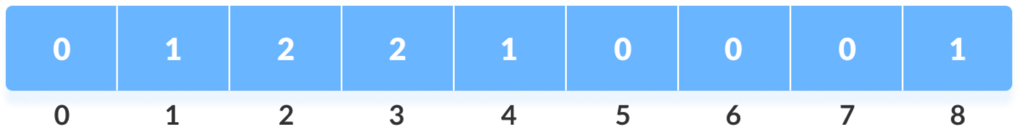
2. Initialize an array of length `max+1` with all elements 0. This array is used for storing the count of the elements in the array.



Count array

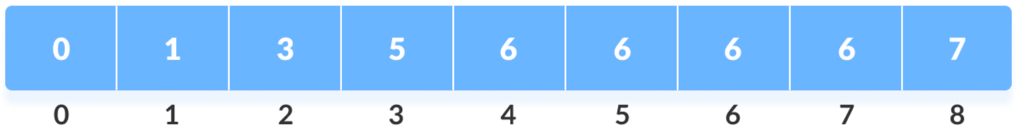
3. Store the count of each element at their respective index in `count` array

   For example: if the count of element 3 is 2 then, 2 is stored in the 3rd position of `count` array. If element "5" is not present in the array, then 0 is stored in 5th position.



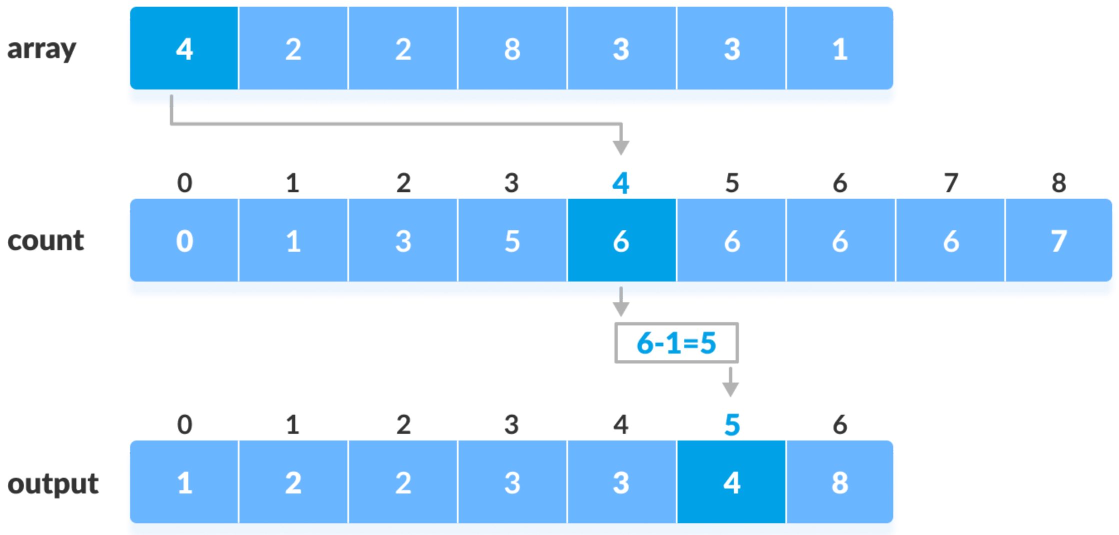
Count of each element stored

4. Store cumulative sum of the elements of the count array. It helps in placing the elements into the correct index of the sorted array.



Cumulative count

5. Find the index of each element of the original array in the count array. This gives the cumulative count. Place the element at the index calculated as shown in figure below.



Counting sort

6. After placing each element at its correct position, decrease its count by one.

TASKS

Q.1 a) Write the function that takes an array of positive integers and sort it out using counting sort, please comment your code properly explaining on every step. Please take snapshot of output and paste it. [15 marks]

```
#include <iostream>
#include <cmath>
#include <algorithm>
using namespace std;
//Finding Maximum in Array
void findMax(int* arr, int size, int& max) {
        max = arr[size-1];
        int i = size-2;
        while(i>0){
                if(arr[i]>max)
                        max = arr[i];
                --i;
        }
}
//For Printing Array
void printArray(int* arr, int size){
        for(int i =0; i<size; ++i)
                cout<<arr[i]<<" ";
        cout<<endl;
}
//Return Array with Cumulative Sum
int* cumulativeSum(int* arr, int size, int max,int newSize) {

        int* newArray = new int[newSize];
        //Initializing Array with 0s.
        fill(newArray, newArray+newSize, 0);

    //Finding the Frequency Of each Element
        for(int i = 0; i<size; ++i)
                ++newArray[arr[i]];

        //Allocating Cumulative Sum in Array
        int sum = newArray[0];
        for(int i = 1; i<newSize; i++){
            sum+=newArray[i];
            newArray[i] = sum;
        }
        return newArray;
}
```

```
//Sort Array on the Basis of Cumulative Sum
int* countSort(int*arr, int size, int max){

        int newSize = max+1;
        int *cumArray = cumulativeSum(arr, size, max, newSize);

        //Getting Sorted Array After Using
        int *sortedArray = new int[size];
        for(int i =0; i<size; i++) {
                sortedArray[cumArray[arr[i]]-1] = arr[i];
           --cumArray[arr[i]];
        }
        return sortedArray;
}

int main(){
        int max = 0;
        //Example 1
        size_t size_1 = 7;
        cout<<"Array 1 is : ";
        int arr[size_1] = {4, 2, 2, 8, 3, 3, 1};
        printArray(arr, size_1);
        findMax(arr, size_1, max);
        int *sortedArray_1 = countSort(arr, size_1, max);
        cout<<"Sorted Array 1 is : ";
        printArray(sortedArray_1, size_1);

        //Example 2
        int size_2 = 5;
        int arr_2[size_2]= {5, 5, 6, 1, 2};
        cout<<"Array 2 is : ";
        printArray(arr_2, size_2);
        findMax(arr_2, size_2, max);
        int *sortedArray_2 = countSort(arr_2, size_2, max);
        cout<<"Sorted Array 2 is : ";
        printArray(sortedArray_2, size_2);

        //Example 3
        int size_3 = 10;
        cout<<"Array 3 is : ";
        int arr_3[size_3]= {5, 5, 6, 1, 2,1,1,5,5,4};
        printArray(arr_3, size_3);
        findMax(arr_3, size_3, max);
        int *sortedArray_3 = countSort(arr_3, size_3, max);
        cout<<"Sorted Array 3 is : ";
        printArray(sortedArray_3, size_3);
```

```
	return 0;
}
```

<u>Q2. </u>) a) Work out the time complexity of the algorithm in Q 1 in terms of number of steps.

<u>b)</u> Briefly discuss the four properties in context of the counting sort algorithm that you have implemented. [10 Marks]

As there are 4 main loop, one for the given array, with size n, and maximum array with size, m;

Loop	Time Complexity
fill(newArray, newArray+newSize, 0);	O(m)
for(int i = 0; i<size; ++i)	O(n)
for(int i = 1; i<newSize; i++)	O(m)
for(int i = 0; i<size; ++i)	O(n)

O(m)+O(n)+O(m)+O(n) = O(2m+2n) = O(2(m+n))

As 2 is constant so, consider n to be very small as compared to

Time Complexity is **O(m).**

b) The Four Properties:

- **Adaptive:** It is not the adaptive algorithm because the frequency and the cumulative sum is calculated even if the array is in the sorted order.
- **In-Place:** As the extra space is allocated during the process of sorting so it is not the In-Place algorithm.
- **Stable:** It is stable because it does not matter in which order the duplicates are placed.
- **Online:** This algorithms works only with the given array, and cannot handling increasing size of array, so it is not an online algorithm.