***Frequencies in a Sorted Array***

Given a sorted [array](https://www.geeksforgeeks.org/array-data-structure/), **arr[]** consisting of **N** integers, the task is to find[the](https://www.geeksforgeeks.org/counting-frequencies-of-array-elements/)frequencies of [each array element](https://www.geeksforgeeks.org/counting-frequencies-of-array-elements/).

**Examples:**

***Input:****arr[] = {1, 1, 1, 2, 3, 3, 5, 5, 8, 8, 8, 9, 9, 10}*  
***Output:****Frequency of 1 is: 3*  
*Frequency of 2 is: 1*  
*Frequency of 3 is: 2*  
*Frequency of 5 is: 2*  
*Frequency of 8 is: 3*  
*Frequency of 9 is: 2*  
*Frequency of 10 is: 1*

***Input:****arr[] = {2, 2, 6, 6, 7, 7, 7, 11}*  
***Output:****Frequency of 2 is: 2*  
*Frequency of 6 is: 2*  
*Frequency of 7 is: 3*  
*Frequency of 11 is: 1*

**Naive Approach:**The simplest approach is to[traverse the array](https://www.geeksforgeeks.org/c-program-to-traverse-an-array/)and keep the count of every element encountered in a [HashMap](https://www.geeksforgeeks.org/java-util-hashmap-in-java-with-examples/) and then, in the end, print the frequencies of every element by[traversing the HashMap.](https://www.geeksforgeeks.org/traverse-through-a-hashmap-in-java/) This approach is already implemented [here](https://www.geeksforgeeks.org/find-frequency-of-each-element-in-a-limited-range-array-in-less-than-on-time/).

***Time Complexity:****O(N)*  
***Auxiliary Space:****O(N)*

**Efficient Approach:**The above approach can be optimized in terms of space used based on the fact that, in a sorted array, the same elements occur consecutively, so the idea is to maintain a variable to keep track of the frequency of elements while traversing the array. Follow the steps below to solve the problem:

* Initialize a variable, say **freq** as **1** to store the frequency of elements.
* [Iterate in the range](https://www.geeksforgeeks.org/range-based-loop-c/)**[1, N-1]**using the variable **i**and perform the following steps:
  + If the value of **arr[i]** is equal to **arr[i-1]**, increment **freq** by **1**.
  + Else print value the frequency of **arr[i-1]** obtained in **freq** and then update **freq** to **1**.
* Finally, after the above step, print the frequency of the last distinct element of the array as **freq**.

Below is the implementation of the above approach:

C++Java

// Java program for the above approach

import java.io.\*;

import java.lang.\*;

import java.util.\*;

class GFG {

// Function to print the frequency

// of each element of the sorted array

static void printFreq(int arr[], int N)

{

// Stores the frequency of an element

int freq = 1;

// Traverse the array arr[]

for (int i = 1; i < N; i++) {

// If the current element is equal

// to the previous element

if (arr[i] == arr[i - 1]) {

// Increment the freq by 1

freq++;

}

// Otherwise,

else {

System.out.println("Frequency of "

+ arr[i - 1]

+ " is: " + freq);

// Update freq

freq = 1;

}

}

// Print the frequency of the last element

System.out.println("Frequency of " + arr[N - 1]

+ " is: " + freq);

}

// Driver Code

public static void main(String args[])

{

// Given Input

int arr[]

= { 1, 1, 1, 2, 3, 3, 5, 5, 8, 8, 8, 9, 9, 10 };

int N = arr.length;

// Function Call

printFreq(arr, N);

}

}

**Output**

Frequency of 1 is: 3

Frequency of 2 is: 1

Frequency of 3 is: 2

Frequency of 5 is: 2

Frequency of 8 is: 3

Frequency of 9 is: 2

Frequency of 10 is: 1

***Time Complexity:****O(N)*  
***Auxiliary Space:****O(1)*