



A computer science portal for geeks

Practice

IDE

Q&amp;A

GeeksQuiz

## Count frequencies of all elements in array in $O(1)$ extra space and $O(n)$ time

Given an unsorted array of  $n$  integers which can contain integers from 1 to  $n$ . Some elements can be repeated multiple times and some other elements can be absent from the array. Count frequency of all elements that are present and print the missing elements.

### Examples:

Input: `arr[] = {2, 3, 3, 2, 5}`

Output: Below are frequencies of all elements

```
1 -> 0
2 -> 2
3 -> 2
4 -> 0
5 -> 1
```

Input: `arr[] = {4, 4, 4, 4}`

Output: Below are frequencies of all elements

```
1 -> 0
2 -> 0
3 -> 0
4 -> 4
```

A **Simple Solution** is to create a count array of size  $n$  as the elements are in range from 1 to  $n$ . This solution works in  $O(n)$  time, but requires  $O(n)$  extra space.

### How to do it in $O(1)$ extra space and $O(n)$ time?

Below are two methods to solve this in  $O(n)$  time and  $O(1)$  extra space. Both method modify given array to achieve  $O(1)$  extra space.

#### Method 1 (By making elements negative)

The idea is to traverse the given array, use elements as index and store their counts at the index. For example, when we see element 7, we go to index 6 and store the count. There are few problems to handle, one is the counts can get mixed with the elements, this is handled by storing the counts as negative. Other

problem is losing the element which is replaced by count, this is handled by first storing the element to be replaced at current index.

#### Algorithm:

```

1) Initialize i as 0
2) Do following while i < n

    // If this element is already processed,
    // then nothing to do
    a) If arr[i] <= 0
        i++;
        continue the loop from beginning

    // Find index corresponding to this element
    // For example, index for 5 is 4
    b) elementIndex = arr[i]-1;

    // If the elementIndex has an element that is not
    // processed yet, then first store that element
    // to arr[i] so that we don't lose anything.
    c) if (arr[elementIndex] > 0)
        (i) arr[i] = arr[elementIndex];

        // After storing arr[elementIndex], change it
        // to store initial count of 'arr[i]'
        (ii) arr[elementIndex] = -1;

    d) else
        // If this is NOT first occurrence of arr[i],
        // then increment its count.
        (i) arr[elementIndex]--;

        // And initialize arr[i] as 0 means the element
        // 'i+1' is not seen so far
        (ii) arr[i] = 0;
        (iii) i++;

3) Now -arr[i] stores count of i+1.

```

Below is C++ implementation of the above approach.

```

// C++ program to print frequencies of all array
// elements in O(1) extra space and O(n) time
#include<bits/stdc++.h>
using namespace std;

// Function to find counts of all elements present in
// arr[0..n-1]. The array elements must be range from
// 1 to n
void findCounts(int *arr, int n)
{
    // Traverse all array elements

```

```

int i = 0;
while (i < n)
{
    // If this element is already processed,
    // then nothing to do
    if (arr[i] <= 0)
    {
        i++;
        continue;
    }

    // Find index corresponding to this element
    // For example, index for 5 is 4
    int elementIndex = arr[i]-1;

    // If the elementIndex has an element that is not
    // processed yet, then first store that element
    // to arr[i] so that we don't lose anything.
    if (arr[elementIndex] > 0)
    {
        arr[i] = arr[elementIndex];

        // After storing arr[elementIndex], change it
        // to store initial count of 'arr[i]'
        arr[elementIndex] = -1;
    }
    else
    {
        // If this is NOT first occurrence of arr[i],
        // then increment its count.
        arr[elementIndex]--;

        // And initialize arr[i] as 0 means the element
        // 'i+1' is not seen so far
        arr[i] = 0;
        i++;
    }
}

printf("\nBelow are counts of all elements\n");
for (int i=0; i<n; i++)
    printf("%d -> %d\n", i+1, abs(arr[i]));
}

// Driver program to test above function
int main()
{
    int arr[] = {2, 3, 3, 2, 5};
    findCounts(arr, sizeof(arr)/ sizeof(arr[0]));

    int arr1[] = {1};
    findCounts(arr1, sizeof(arr1)/ sizeof(arr1[0]));

    int arr3[] = {4, 4, 4, 4};
    findCounts(arr3, sizeof(arr3)/ sizeof(arr3[0]));

    int arr2[] = {1, 3, 5, 7, 9, 1, 3, 5, 7, 9, 1};
    findCounts(arr2, sizeof(arr2)/ sizeof(arr2[0]));

    int arr4[] = {3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3};
    findCounts(arr4, sizeof(arr4)/ sizeof(arr4[0]));

    int arr5[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11};
    findCounts(arr5, sizeof(arr5)/ sizeof(arr5[0]));
}

```

```
int arr6[] = {11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1};  
findCounts(arr6, sizeof(arr6)/ sizeof(arr6[0]));  
  
return 0;  
}
```

[Run on IDE](#)

Output:

Below are counts of all elements

1 -> 0  
2 -> 2  
3 -> 2  
4 -> 0  
5 -> 1

Below are counts of all elements

1 -> 1

Below are counts of all elements

1 -> 0  
2 -> 0  
3 -> 0  
4 -> 4

Below are counts of all elements

1 -> 3  
2 -> 0  
3 -> 2  
4 -> 0  
5 -> 2  
6 -> 0  
7 -> 2  
8 -> 0  
9 -> 2  
10 -> 0  
11 -> 0

Below are counts of all elements

1 -> 0  
2 -> 0  
3 -> 11  
4 -> 0  
5 -> 0  
6 -> 0  
7 -> 0  
8 -> 0  
9 -> 0  
10 -> 0  
11 -> 0

Below are counts of all elements

```
1 -> 1
2 -> 1
3 -> 1
4 -> 1
5 -> 1
6 -> 1
7 -> 1
8 -> 1
9 -> 1
10 -> 1
11 -> 1
```

Below are counts of all elements

```
1 -> 1
2 -> 1
3 -> 1
4 -> 1
5 -> 1
6 -> 1
7 -> 1
8 -> 1
9 -> 1
10 -> 1
11 -> 1
```

### How does above program work?

Let us take below example to see step by step processing of above program:

`arr[] = {2, 3, 3, 2, 5}`

**`i = 0, arr[i] = 2, arr[] = {2, 3, 3, 2, 5}`**

Since `arr[i] > 0`, find `elementIndex`.

`elementIndex = arr[i] - 1 = 2 - 1 = 1,`

`arr[elementIndex]` or `arr[1]` is 3

Since `arr[elementIndex]` is postive,

`arr[i] = arr[elementIndex] = 3`

`arr[elementIndex] = -1` // 2 is seen 1 times so far

`i` is not changed.

**`i = 0, arr[i] = 3, arr[] = {3, -1, 3, 2, 5}`**

Since `arr[i] > 0`, find `elementIndex`.

`elementIndex = arr[i] - 1 = 3 - 1 = 2`

`arr[elementIndex]` or `arr[2]` is 3

Since `arr[elementIndex]` is postive

`arr[i] = arr[elementIndex] = 3`

$\text{arr}[\text{elementIndex}] = -1$  // 3 is seen 1 times so far

i is not changed.

**i = 0, arr[i] = 3, arr[] = {3, -1, -1, 2, 5}**

Since  $\text{arr}[i] > 0$ , find elementIndex.

$\text{elementIndex} = \text{arr}[i] - 1 = 3 - 1 = 2$

$\text{arr}[\text{elementIndex}]$  or  $\text{arr}[2]$  is -1

Since  $\text{arr}[\text{elementIndex}]$  is negative

$\text{arr}[\text{elementIndex}] = \text{arr}[\text{elementIndex}] - 1$

$= -2$  // 3 is seen 2 times so far

$\text{arr}[i] = 0$  // 1 is not seen so far

i is incremented

**i = 1, arr[i] = -1, arr[] = {0, -1, -2, 2, 5}**

Since  $\text{arr}[i]$  is negative, increment i

**i = 2, arr[i] = -2, arr[] = {0, -1, -2, 2, 5}**

Since  $\text{arr}[i]$  is negative, increment i

**i = 3, arr[i] = 2, arr[] = {0, -1, -2, 2, 5}**

Since  $\text{arr}[i] > 0$ , we find elementIndex.

$\text{elementIndex} = \text{arr}[i] - 1 = 2 - 1 = 1$

$\text{arr}[\text{elementIndex}]$  or  $\text{arr}[1]$  is -1

Since  $\text{arr}[\text{elementIndex}]$  is negative

$\text{arr}[\text{elementIndex}] = \text{arr}[\text{elementIndex}] - 1$

$= -2$  // 2 is seen 2 times so far

$\text{arr}[i] = 0$  // 4 is not seen so far

i is incremented

**i = 4, arr[i] = 5, arr[] = {0, -2, -2, 0, 5}**

Since  $\text{arr}[i] > 0$ , we find elementIndex.

$\text{elementIndex} = \text{arr}[i] - 1 = 5 - 1 = 4$

$\text{arr}[\text{elementIndex}]$  or  $\text{arr}[4]$  is 5

Since  $\text{arr}[\text{elementIndex}]$  is postive

$\text{arr}[i] = \text{arr}[\text{elementIndex}] = 4$

$\text{arr}[\text{elementIndex}] = -1$  // 5 is seen 1 times so far

i is not changed.

**i = 1, arr[i] = -1, arr[] = {0, -2, -2, 0, -1}**

Since  $\text{arr}[i]$  is negative, increment i

## Method 2 (By adding n to keep track of counts)

- 1) Subtract 1 from every element so that the elements become in range from 0 to n-1  

```
for (int j =0; j < n; j++)
    arr[j] = arr[j]-1;
```
- 2) Use every element arr[i] as index and add 'n' to element present at arr[i]%n to keep track of count of occurrences of arr[i]  

```
for (int i=0; i < n; i++)
    arr[arr[i]%n] = arr[arr[i]%n] + n;
```
- 3) To print counts, simply print the number of times n was added at index corresponding to every element  

```
for (int i =0; i < n; i++)
    print "(i + 1) -> arr[i] "
```

Below is C++ implementation of above idea.

```
// C++ program to print frequencies of all array
// elements in O(1) extra space and O(n) time
#include<bits/stdc++.h>
using namespace std;

// Function to find counts of all elements present in
// arr[0..n-1]. The array elements must be range from
// 1 to n
void printfrequency(int arr[],int n)
{
    // Subtract 1 from every element so that the elements
    // become in range from 0 to n-1
    for (int j =0; j<n; j++)
        arr[j] = arr[j]-1;

    // Use every element arr[i] as index and add 'n' to
    // element present at arr[i]%n to keep track of count of
    // occurrences of arr[i]
    for (int i=0; i<n; i++)
        arr[arr[i]%n] = arr[arr[i]%n] + n;

    // To print counts, simply print the number of times n
    // was added at index corresponding to every element
    for (int i =0; i<n; i++)
        cout << i + 1 << " -> " << arr[i]/n << endl;
}

// Driver program to test above function
int main()
{
    int arr[] = {2, 3, 3, 2, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    printfrequency(arr,n);
    return 0;
}
```

[Run on IDE](#)

Output:

```
1 -> 0
2 -> 2
3 -> 2
4 -> 0
5 -> 1
```

Thanks to Vivek Kumar for suggesting this solution in a comment below.

This article is contributed by Shubham Gupta. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above



A Free Course From Microsoft

## Querying with Transact-SQL

**edX**  
edx.org

**Enroll now**  
Self-Paced

55 Comments Category: [Arrays](#)

### Related Posts:

- [Longest Span with same Sum in two Binary arrays](#)
- [Count Inversions of size three in a give array](#)
- [Find the subarray with least average](#)
- [Count triplets with sum smaller than a given value](#)
- [Find zeroes to be flipped so that number of consecutive 1's is maximized](#)
- [Reorder an array according to given indexes](#)
- [Find maximum value of  \$\text{Sum}\(i \* \text{arr}\[i\]\)\$  with only rotations on given array allowed](#)
- [Find maximum average subarray of k length](#)

([Login](#) to Rate and Mark)

3.3

Average Difficulty : **3.3/5.0**  
Based on **6** vote(s)

☐  
☐

Add to TODO List

Mark as DONE



**Like** **Share** 64 people like this. Be the first of your friends.

Writing code in comment? Please use [code.geeksforgeeks.org](https://code.geeksforgeeks.org), generate link and share the link here.

@geeksforgeeks, Some rights reserved

[Contact Us!](#)

[About Us!](#)

[Advertise with us!](#)