Given an array of size n and a number k, find all elements that appear more than n/k times

Last Updated: 30-10-2020

Given an array of size n, find all elements in array that appear more than n/k times. For example, if the input arrays is {3, 1, 2, 2, 1, 2, 3, 3} and k is 4, then the output should be [2, 3]. Note that size of array is 8 (or n = 8), so we need to find all elements that appear more than 2 (or 8/4) times. There are two elements that appear more than two times, 2 and 3.

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

A **simple method** is to pick all elements one by one. For every picked element, count its occurrences by traversing the array, if count becomes more than n/k, then print the element. Time Complexity of this method would be $O(n^2)$.
A better solution is to **use sorting**. First, sort all elements using a O(nLogn) algorithm. Once the array is sorted, we can find all required elements in a linear scan of array. So overall time complexity of this method is O(nLogn) + O(n) which is O(nLogn).
Following is an interesting **O(nk) solution:**

We can solve the above problem in O(nk) time using O(k-1) extra space. Note that there can never be more than k-1 elements in output (Why?). There are mainly three steps in this algorithm.

1) Create a temporary array of size (k-1) to store elements and their counts (The output elements are going to be among these k-1 elements). Following is structure of temporary array elements.

```
struct eleCount {
    int element;
    int count;
};
struct eleCount temp[];
```

This step takes O(k) time.
2) Traverse through the input array and update temp[] (add/remove an element or increase/de-

3) Iterate through final (k-1) potential candidates (stored in temp[]). or every element, check if it actually has count more than n/k. This step takes O(nk) time.

The main step is step 2, how to maintain (k-1) potential candidates at every point? The steps used in step 2 are like famous game: Tetris. We treat each number as a piece in Tetris, which falls down in our temporary array temp[]. Our task is to try to keep the same number stacked on the same column (count in temporary array is incremented).

```
Consider k = 4, n = 9
Given array: 3 1 2 2 2 1 4 3 3

i = 0
         3 _ _
temp[] has one element, 3 with count 1

i = 1
         3 1 _
temp[] has two elements, 3 and 1 with
counts 1 and 1 respectively

i = 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 1 respectively.
```

```
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 2 respectively.

i = 4
         - - 2
         - - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 3 respectively.

i = 5
         - - 2
         - 1 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 2 and 3 respectively.
```

Now the question arises, what to do when temp[] is full and we see a new element – we remove the bottom row from stacks of elements, i.e., we decrease count of every element by 1 in temp[]. We ignore the current element.

```
i = 6
         - - 2
         - 1 2
temp[] has two elements, 1 and 2 with
counts as 1 and 2 respectively.

i = 7
           - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 1, 1 and 2 respectively.

i = 8
         3 - 2
         3 1 2
temp[] has three elements, 3, 1 and 2 with
counts as 2, 1 and 2 respectively.
```

Finally, we have at most k-1 numbers in temp[]. The elements in temp are {3, 1, 2}. Note that the counts in temp[] are useless now, the counts were needed only in step 2. Now we need to check whether the actual counts of elements in temp[] are more than n/k (9/4) or not. The elements 3 and 2 have counts more than 9/4. So we print 3 and 2.

Note that the algorithm doesn't miss any output element. There can be two possibilities, many occurrences are together or spread across the array. If occurrences are together, then count will be high and won't become 0. If occurrences are spread, then the element would come again in temp[]. Following is the implementation of the above algorithm.

```
// A C++ program to print elements with count more than n/k
#include <iostream>
using namespace std;

// A structure to store an element and its current count
struct eleCount {
    int e; // Element
    int c; // Count
};

// Prints elements with more
// than n/k occurrences in arr[]
// of size n. If there are no
// such elements, then it prints
// nothing.
void moreThanNdK(int arr[], int n, int k)
{
    // k must be greater than
    // 1 to get some output
    if (k < 2)
        return;

    /* Step 1: Create a temporary
       array (contains element
       and count) of size k-1.
       Initialize count of all
       elements as 0 */
    struct eleCount temp[k - 1];
    for (int i = 0; i < k - 1; i++)
        temp[i].c = 0;
```

```
{
    int j;

    /* If arr[i] is already present in
       the element count array,
       then increment its count
     */
    for (j = 0; j < k - 1; j++)
    {
        if (temp[j].e == arr[i])
        {
            temp[j].c += 1;
            break;
        }
    }

    /* If arr[i] is not present in temp[] */
    if (j == k - 1) {
        int l;

        /* If there is position available
          in temp[], then place arr[i] in
          the first available position and
          set count as 1*/
        for (l = 0; l < k - 1; l++)
        {
            if (temp[l].c == 0)
            {
                temp[l].e = arr[i];
                temp[l].c = 1;
                break;
            }
        }

        /* If all the position in the
           temp[] are filled, then decrease
           count of every element by 1 */
        if (l == k - 1)
            for (l = 0; l < k; l++)
                temp[l].c -= 1;
    }
}

/*Step 3: Check actual counts of
 * potential candidates in temp[]*/
for (int i = 0; i < k - 1; i++)
{
    // Calculate actual count of elements
    int ac = 0; // actual count
    for (int j = 0; j < n; j++)
```

```
        // then print it
         if (ac > n / k)
             cout << "Number:" << temp[i].e
                  << " Count:" << ac << endl;
    }
}

/* Driver code */
int main()
{
    cout << "First Test\n";
    int arr1[] = { 4, 5, 6, 7, 8, 4, 4 };
    int size = sizeof(arr1) / sizeof(arr1[0]);
    int k = 3;
    moreThanNdK(arr1, size, k);

    cout << "\nSecond Test\n";
    int arr2[] = { 4, 2, 2, 7 };
    size = sizeof(arr2) / sizeof(arr2[0]);
    k = 3;
    moreThanNdK(arr2, size, k);

    cout << "\nThird Test\n";
    int arr3[] = { 2, 7, 2 };
    size = sizeof(arr3) / sizeof(arr3[0]);
    k = 2;
    moreThanNdK(arr3, size, k);

    cout << "\nFourth Test\n";
    int arr4[] = { 2, 3, 3, 2 };
    size = sizeof(arr4) / sizeof(arr4[0]);
    k = 3;
    moreThanNdK(arr4, size, k);

    return 0;
}
```

Python3

```
# A Python3 program to print elements with
# count more than n/k

# Prints elements with more than n/k
# occurrences in arrof size n. If
# there are no such elements, then
# it prints nothing.
```

```
if (k < 2):
    return

# Step 1: Create a temporary array
# (contains element and count) of
# size k-1. Initialize count of all
# elements as 0
temp = [[0 for i in range(2)]
        for i in range(k)]

for i in range(k - 1):
    temp[i][0] = 0

# Step 2: Process all elements
# of input array
for i in range(n):
    j = 0

    # If arr[i] is already present in
    # the element count array, then
    # increment its count
    while j < k - 1:
        if (temp[j][1] == arr[i]):
            temp[j][0] += 1
            break

        j += 1

    # If arr[i] is not present in temp
    if (j == k - 1):
        l = 0

        # If there is position available
        # in temp[], then place arr[i]
        # in the first available position
        # and set count as 1*/
        while l < k - 1:
            if (temp[l][0] == 0):
                temp[l][1] = arr[i]
                temp[l][0] = 1
                break

            l += 1

        # If all the position in the
        # tempare filled, then decrease
        # count of every element by 1
        if (l == k - 1):
            while l < k:
                temp[l][0] -= 1
```

```
    for i in range(k - 1):

        # Calculate actual count of elements
        ac = 0  # Actual count
        for j in range(n):
            if (arr[j] == temp[i][1]):
                ac += 1

        # If actual count is more
        # than n/k, then prit
        if (ac > n // k):
            print("Number:",
                  temp[i][1],
                  " Count:", ac)


# Driver code
if __name__ == '__main__':

    print("First Test")
    arr1 = [4, 5, 6, 7, 8, 4, 4]
    size = len(arr1)
    k = 3
    moreThanNdK(arr1, size, k)

    print("Second Test")
    arr2 = [4, 2, 2, 7]
    size = len(arr2)
    k = 3
    moreThanNdK(arr2, size, k)

    print("Third Test")
    arr3 = [2, 7, 2]
    size = len(arr3)
    k = 2
    moreThanNdK(arr3, size, k)

    print("Fourth Test")
    arr4 = [2, 3, 3, 2]
    size = len(arr4)
    k = 3
    moreThanNdK(arr4, size, k)

# This code is contributed by mohit kumar 29
```

Output

```
Second Test
Number:2 Count:2

Third Test
Number:2 Count:2

Fourth Test
Number:2 Count:2
Number:3 Count:2
```

Time Complexity: O(nk)

Auxiliary Space: O(k)

Generally asked variations of this problem are, find all elements that appear n/3 times or n/4 times in O(n) time complexity and O(1) extra space.

Another Approach:

Hashing can also be an efficient solution. With a good hash function, we can solve the above problem in O(n) time on average. Extra space required hashing would be higher than O(k). Also, hashing cannot be used to solve the above variations with O(1) extra space.

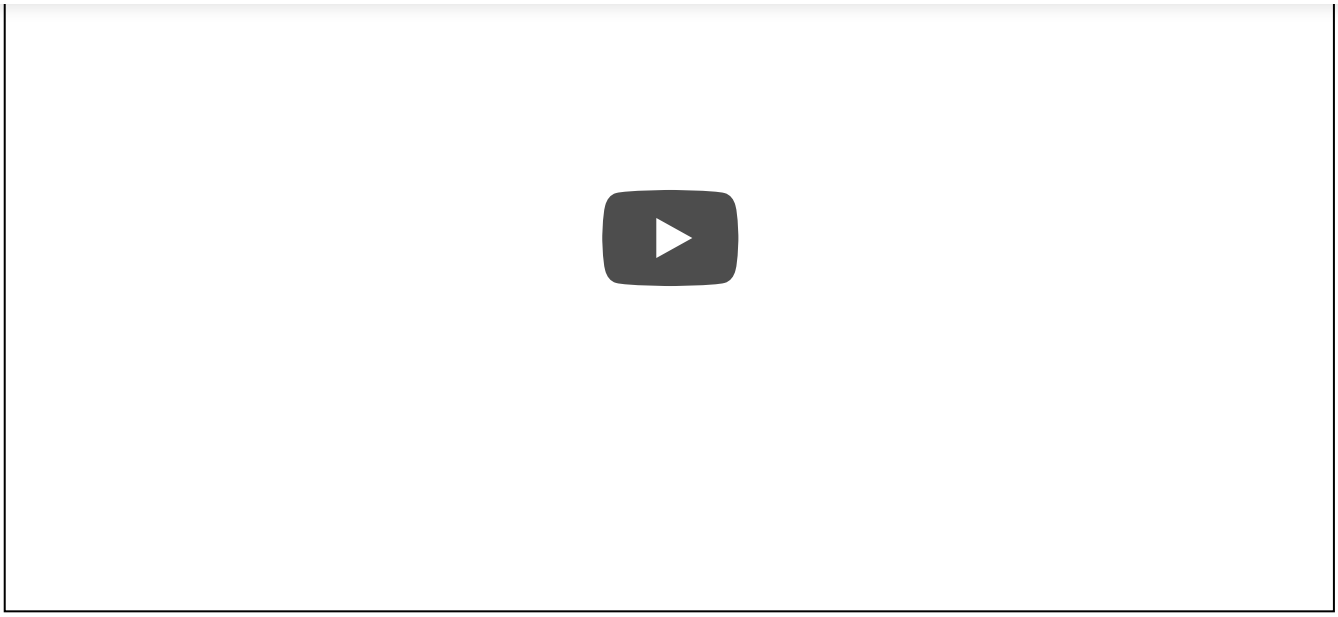
Below is the implementation of the above idea:

```
// Java Code to find elements whose
// frequency yis more than n/k
import java.util.*;

public class Main

{
    public static void morethanNdK(int a[], int n, int k)
    {
        int x = n / k;

        // Hash map initialization
        HashMap<Integer, Integer> y = new HashMap<>();

        // count the frequency of each element.
        for (int i = 0; i < n; i++)
```

```
            // if lement does exist in the hash table
            else
            {
                int count = y.get(a[i]);
                y.put(a[i], count + 1);
            }
        }

        // iterate over each element in the hash table
        // and check their frequency, if it is more than
        // n/k, print it.
        for (Map.Entry m : y.entrySet())
        {
            Integer temp = (Integer)m.getValue();
            if (temp > x)
                System.out.println(m.getKey());
        }
    }

    // Driver Code
    public static void main(String[] args)
    {

        int a[] = new int[] { 1, 1, 2, 2, 3, 5, 4,
                              2, 2, 3, 1, 1, 1 };
        int n = 12;
        int k = 4;
        morethanNdK(a, n, k);
    }
}
```

Output

```
1
2
```



Attention reader! Don't stop learning now. Get hold of all the important DSA concepts with the **DSA Self Paced Course** at a student-friendly price and become industry ready.

Recommended Posts:

Remove elements from the array which appear more than k times

Array elements that appear more than once

Remove elements that appear strictly less than k times

Find all array elements occurring more than N/3 times

Largest substring where all characters appear at least K times | Set 2

Element which occurs consecutively in a given subarray more than or equal to K times

Maximum sum possible for a sub-sequence such that no two elements appear at a distance < K in the array

Count number of permutation of an Array having no SubArray of size two or more from original Array

Remove minimum elements from the array such that 2*min becomes more than max

Check if all occurrences of a character appear together

Count of all possible Paths in a Tree such that Node X does not appear before Node Y

Maximum size of square such that all submatrices of that size have sum less than K

Maximum subarray size, such that all subarrays of that size have sum less than k

Queries to increment array elements in a given range by a given value for a given number of times

Sum of all array elements less than X and greater than Y for Q queries

Count pairs (p, q) such that p occurs in array at least q times and q occurs at least p times

Count of Array elements greater than all elements on its left and next K elements on its right

Count of Array elements greater than all elements on its left and at least K elements on its right

Improved By : ShubhamMaurya3, mohit kumar 29, depanshukumar4444

Article Tags : Arrays Searching

Practice Tags : Arrays Searching

23

3.7

To-do Done

Based on **121** vote(s)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

137 Comments GeeksforGeeks Disqus' Privacy Policy 1 Login

Recommend 5 Sort by Newest

Join the discussion…

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

AMANPREET SINGH SETIA • 9 days ago

you do not have access to this problem

Reply • Share ›

alexwest11 • 2 months ago • edited

The above problem can be solved in O(nLogk) time with the help of more appropriate data structures than array for auxiliary storage of k-1 elements. Suggest a O(nLogk) approach.

Contact Us

CS Subjects

Video Tutorials

Practice

Courses

Company-wise

Topic-wise

How to begin?

Contribute

Write an Article

Write Interview Experience

Internships

Videos