Find K most occurring elements in the given Array

Given an array of **N** numbers and a positive integer **K**. The problem is to find **K** numbers with the most occurrences, i.e., the top **K** numbers having the maximum frequency. If two numbers have the same frequency then the number with a larger value should be given preference. The numbers should be displayed in decreasing order of their frequencies. It is assumed that the array consists of at least K numbers.

Examples:

Input: *arr[] = {3, 1, 4, 4, 5, 2, 6, 1}, K = 2*

Output: *4 1*

Explanation:

Frequency of 4 = 2, Frequency of 1 = 2

These two have the maximum frequency and 4 is larger than 1.

Input: *arr[] = {7, 10, 11, 5, 2, 5, 5, 7, 11, 8, 9}, K = 4*

Output: *5 11 7 10*

Explanation:

Frequency of 5 = 3, Frequency of 11 = 2, Frequency of 7 = 2, Frequency of 10 = 1

These four have the maximum frequency and 5 is largest among rest.

Find K most occurring elements in the given Array using Map

To solve the problem using this approach follow the below idea:

create a Map to store the element-frequency pair. Map is used to perform insertion and updation in constant time. Then sort the element-frequency pair in decreasing order of frequency. This gives the information about each element and the number of times they are present in the array. To get K elements of the array, print the first K elements of the sorted array.

Follow the given steps to solve the problem:

- Create a map mp, to store key-value pair, i.e. element-frequency pair.
- Traverse the array from start to end.
- For every element in the array update mp*[array[i]]++*
- Store the element-frequency pair in a vector and sort the vector in decreasing order of frequency.
- Print the first k elements of the sorted array.

Below is the Implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

```
// C++ implementation to find k numbers with most
// occurrences in the given array
#include <bits/stdc++.h>
using namespace std;

// Comparison function to sort the 'freq_arr[]'
bool compare(pair<int, int> p1, pair<int, int> p2)
{
    // If frequencies of two elements are same
    // then the larger number should come first
    if (p1.second == p2.second)
        return p1.first > p2.first;

    // Sort on the basis of decreasing order
    // of frequencies
    return p1.second > p2.second;
}

// Function to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int N, int K)
{
    // unordered_map 'mp' implemented as frequency hash
```

```
    // table
    unordered_map<int, int> mp;
    for (int i = 0; i < N; i++)
        mp[arr[i]]++;

    // store the elements of 'mp' in the vector 'freq_arr'
    vector<pair<int, int> > freq_arr(mp.begin(), mp.end());

    // Sort the vector 'freq_arr' on the basis of the
    // 'compare' function
    sort(freq_arr.begin(), freq_arr.end(), compare);

    // display the top k numbers
    cout << K << " numbers with most occurrences are:\n";
    for (int i = 0; i < K; i++)
        cout << freq_arr[i].first << " ";
}

// Driver's code
int main()
{
    int arr[] = { 3, 1, 4, 4, 5, 2, 6, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int K = 2;

    // Function call
    print_N_mostFrequentNumber(arr, N, K);
```

```
    return 0;
}
```

Learn Data Structures & Algorithms with GeeksforGeeks

Output

```
2 numbers with most occurrences are:
4 1
```

Time Complexity: O(D log D), where D is the count of **distinct** elements in the array
Auxiliary Space: O(D), where D is the count of **distinct** elements in the array

---

--

Find K most occurring elements in the given Array using Max-Heap

To solve the problem using this approach follow the below idea:

Approach: *Create a Map to store element-frequency pair. Map is used to perform insertion and updation in constant time. Then use a priority queue to store the element-frequency pair (Max-Heap). The element which has maximum frequency, comes at the root of the Priority Queue. Remove the top or root of Priority Queue K times and print the element.*

Follow the given steps to solve the problem:

- Create a map mp, to store key-value pair, i.e. element-frequency pair.
- Traverse the array from start to end.
- For every element in the array update mp*[array[i]]++*
- Store the element-frequency pair in a Priority Queue
- Run a loop k times, and in each iteration remove the root of the priority queue and print the element.

Below is the Implementation of the above approach:

- C++
- Java
- Python3
- C#
- Javascript

```cpp
// C++ implementation to find k numbers with most
// occurrences in the given array

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

// Comparison function defined for the priority queue
struct compare {
    bool operator()(pair<int, int> p1, pair<int, int> p2)
    {
        // If frequencies of two elements are same
        // then the larger number should come first
        if (p1.second == p2.second)
            return p1.first < p2.first;

        // Insert elements in the priority queue on the
        // basis of decreasing order of frequencies
        return p1.second < p2.second;
    }
};

// Function to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int N, int K)
{
    // unordered_map 'mp' implemented as frequency hash
    // table
    unordered_map<int, int> mp;
```

```
    for (int i = 0; i < N; i++)
        mp[arr[i]]++;

    // priority queue 'pq' implemented as max heap on the
    // basis of the comparison operator 'compare' element
    // with the highest frequency is the root of 'pq' in
    // case of conflicts, larger element is the root
    priority_queue<pair<int, int>, vector<pair<int, int> >,
                   compare>
        pq(mp.begin(), mp.end());

    // Display the top k numbers
    cout << K << " numbers with most occurrences are:\n";
    for (int i = 1; i <= K; i++) {
        cout << pq.top().first << " ";
        pq.pop();
    }
}

// Driver's code
int main()
{
    int arr[] = { 3, 1, 4, 4, 5, 2, 6, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int K = 2;

    // Function call
```

```
    print_N_mostFrequentNumber(arr, N, K);
    return 0;
}
```


Output

```
2 numbers with most occurrences are:
4 1
```

Time Complexity: O(K log D + D log D), where **D** is the count of **distinct** elements in the array.

- To remove the top of the priority queue O(log d) time is required, so if k elements are removed then O(k log d) time is required, and
- To construct a priority queue with D elements, O(D log D) time is required.

Auxiliary Space: O(D), where **D** is the count of **distinct** elements in the array.

Find K most occurring elements in the given Array using Bucket Sort

- Create a HashMap **elementCount** and store the count of the elements in the given array.
- Create a 2D vector **frequency** of size **N+1** to store the elements according to their frequencies.
- Now initialize a variable **count = 0.**
- While **count < K**:
    - Traverse the **frequency** vector from **N** till **0** and print the elements present in the vector and increment the count for each element.

Below is the implementation of above approach:

- C++
- Java
- Python3
- C#
- Javascript

```cpp
// C++ program to find k numbers with most
// occurrences in the given array

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

// Function to print the k numbers with most occurrences
void print_N_mostFrequentNumber(int arr[], int N, int K)
{
    // HashMap to store count of the elements
    unordered_map<int, int> elementCount;
    for (int i = 0; i < N; i++) {
        elementCount[arr[i]]++;
    }

    // Array to store the elements according
    // to their frequency
    vector<vector<int> > frequency(N + 1);

    // Inserting elements in the frequency array
    for (auto element : elementCount) {
        frequency[element.second].push_back(element.first);
    }
```

```
int count = 0;
cout << K << " numbers with most occurrences are:\n";

for (int i = frequency.size() - 1; i >= 0; i--) {

    // if frequency is same,then take number with a
    // larger value
    // so,if more than 1 number present with same
    // frequency,then sort frequency[i] in descending
    // order
    if (frequency[i].size() > 1) {
        sort(frequency[i].begin(), frequency[i].end(),
             greater<int>());
    }

    for (auto element : frequency[i]) {
        count++;
        cout << element << " ";

        // if K elements have been printed
        if (count >= K)
            return;
    }
}

return;
```

```
}

// Driver's code
int main()
{
    int arr[] = { 3, 1, 4, 4, 5, 2, 6, 1 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int K = 2;

    // Function call
    print_N_mostFrequentNumber(arr, N, K);

    return 0;
}

// This code has been improved by shubhamm050402
```


Output

```
2 numbers with most occurrences are:
4 1
```

Time Complexity: O(N logN), where N is the size of the given array. worst case is when all the elements are the same, we are sorting the entire vector.
Auxiliary Space: O(N)

Find K most occurring elements in the given Array using Quick Select:

In quick select we partition the array of unique numbers in such a way that the elements to the left of pivotpivotpivot are more frequent than pivot, and elements to the right of pivotare less frequent than pivot. Thus we can say the pivotis in its sorted position. We randomly chose such pivot until we finally find its sorted position to be K. Then we know that all the elements to the left of pivot are more frequent than pivotand each time we reduce our paritioning space w.r.t the pivot.

Partition Algorithm

Let's imagine the pivot at store_point. If the iiith element is more frequent than pivot, then the ith element will be on the left of pivot, but the store_point is instead holding an element which is either equally or less frequent than pivot, so it should go to the right of pivot. Hence, we swap the two elements and move store_point forward so that the more frequent element is on the left. If iiith element is less frequent than pivot we assume pivot is at its right place and don't move store_point .

At the end we swap store_point with pivot and return the sorted index.

- C++
- Java
- Python
- C#
- Javascript

```
// C++ program to find k numbers with most
// occurrences in the given array

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

vector<int> print_N_mostFrequentNumber(vector<int>& nums,
```

```
                                        int k,
                                        vector<int>& out)
{
    // map for counting the number of
    // occurences
    unordered_map<int, int> counts;
    // stroing the frequency of each element
    for (int num : nums)
        ++counts[num];
    // creating a vector for storing the
    // frequency
    vector<pair<int, int> > freqs;
    for (auto vt : counts)
        freqs.push_back({ vt.second, vt.first });
    // using the user defined function
    //  nth_element to extract the values
    nth_element(freqs.begin(), freqs.end() - k,
                freqs.end());
    // using user defined function transform
    // to make the desired changes
    transform(freqs.end() - k, freqs.end(), out.begin(),
              [](pair<int, int> vt) { return vt.second; });
    // store the result in the out vector
    return out;
}

// Driver's code
```

```
int main()
{
    vector<int> arr{ 3, 1, 4, 4, 5, 2, 6, 1 };
    int K = 2;

    // Function call
    vector<int> ans(K);
    print_N_mostFrequentNumber(arr, K, ans);
    cout << K
         << " numbers with most occurences are : " << endl;
    for (int i = ans.size() - 1; i >= 0; i--) {
        cout << ans[i] << " ";
    }

    return 0;
}
```


Output

```
2 numbers with most occurences are :
4 1
```

Time complexity: O(n)
Auxiliary Space: O(N)