k most frequent elements

Brute Force :

We make use of map for frequency and push all map elements into heap. A custom comparator function needs to be written which sorts according to 2nd element when first element in priority_queue pair is equal. At end we need to sort the ans vector

Code :

```
#include <bits/stdc++.h>
class comparator
{
    public:
    bool operator()(pair<int,int> a,pair<int,int> b)
    {
       if(a.first<b.first)
       {
           return true;
       }
       if(a.first==b.first)
       {
           if(a.second>b.second)
                return false;
           else
                return true;
       }
       return false;
    }
};

vector<int> KMostFrequent(int n, int k, vector<int> &arr)
{
    // Write your code here.
    map<int,int> m;
    vector<int> ans;
    for(int i=0;i<n;i++)
    {
        m[arr[i]]++;
    }
    priority_queue<pair<int,int>,vector<pair<int,int>>,comparator>pq;
    for(auto it:m)
    {

        pq.push({it.second,it.first});
    }
    while(k>0)
    {
        ans.push_back(pq.top().second);
        pq.pop();
        k--;
    }
    sort(ans.begin(),ans.end());
    return ans;
}
```

- Time Complexity : O(nlogn)
- Space Complexity : O(m) where mis the unique elements in map

Optimal Approach :

Enhanced Bucket sort.

We declare an array of size n+1 indicating that no number can occur more than size of array times. So we use this array as hash to store all the elements that have a count of 1,2,3,….n. We store freq using map and then iterate in the map to add the elements to the vector of freq which holds the numbers that occur freq[i] times. Then we iterate through this freq array backwards because we need maximum count and keep decrementing k on pushing number to ans untilk reaches 0

```
#include <bits/stdc++.h>
vector<int> KMostFrequent(int n, int k, vector<int> &arr)
{
    // Write your code here.
    vector<int> ans;
    vector<int> freq[n+1];
    map<int,int> count;
    for(int i=0;i<n;i++)
    {
        count[arr[i]]++;
    }
    for(auto m:count)
    {
        freq[m.second].push_back(m.first);
    }
    for(int i=n;i>0;i--)
    {
        for(int j=0;j<freq[i].size();j++)
        {
            ans.push_back(freq[i][j]);
            k--;
            if(k==0)
                break;
        }
        if(k==0)
            break;
    }
    sort(ans.begin(),ans.end());
    return ans;
}
```

- Time Complexity : O(n)
- Space Complexity : O(n)