Merge k sorted arrays

BRUTE FORCE :

Pick two arrays and merge them then pick next array and merge it repeat until k==0

Code:

```
#include <bits/stdc++.h>
vector<int> merge(vector<int> a,vector<int> b)
{
    int m=a.size(),n=b.size(),i=0,j=0,k=0;
    vector<int> ans(m+n,0);
    while(i<m && j<n)
    {
        if(a[i]<=b[j])
        {
            ans[k]=a[i];
            i++;
        }
        else
        {
            ans[k]=b[j];
            j++;
        }
        k++;

    }
    while(i<m)
    {
        ans[k]=a[i];
        i++;
        k++;
    }
    while(j<n)
    {
        ans[k]=b[j];
        j++;
        k++;
    }
    return ans;
}
vector<int> mergeKSortedArrays(vector<vector<int>> &kArrays, int k)
{
    // Write your code here.
    vector<int> ans;
    if(k==0)
        return ans;
```

```
    if(k==1)
        return kArrays[0];
    ans=merge(kArrays[0],kArrays[1]);
    for(int i=2;i<k;i++)
    {
        ans=merge(ans,kArrays[i]);
    }

    return ans;
}
```

- Time Complexity : O(k *(m+n))
- Space Complexity : O(1)

Optimal Approach :

We maintain an index array which stores the index to be picked from each of k arrays. We make use of min heap to find the min element each time and we also store the array number along with the element in the minheap to know the element belongs to which array. Initially we push the first index of all kArrays into the minheap and initialize index array to 0. For each iteration we pop the top element from the minheap add it to our answer and then compare whether the array to which the min element belongs has a valid index of index+1 if yes then we push the next element of the array to the minheap and also update index array with the current updated index+1 in the index array.

Code :

```
#include <bits/stdc++.h>
vector<int> mergeKSortedArrays(vector<vector<int>> &kArrays, int k)
{
    // Write your code here.
    vector<int> ans;
    priority_queue<pair<int,int>,vector<pair<int,int>>,greater<pair<int,int>>> pq;
    vector<int> ind(k,0);
    for(int i=0;i<k;i++)
    {
      pq.push({kArrays[i][0], i});
    }
    while(!pq.empty())
    {
        auto tp=pq.top();
        pq.pop();
        ans.push_back(tp.first);
```

```
        if(ind[tp.second]+1<kArrays[tp.second].size())
        {
          pq.push({kArrays[tp.second][ind[tp.second] + 1], tp.second});
          ind[tp.second] += 1;
        }
    }

    return ans;
}
```

- Time Complexity : O(nlogn) where n is total number of elements in merged array
- Space Compelxity :O(n)