Top K Frequent Elements

Given a non-empty array of integers, return the ***k*** most frequent elements.

For example,
Given `[1,1,1,2,2,3]` and k = 2, return `[1,2]`.

Note:

- You may assume *k* is always valid, 1 ≤ *k* ≤ number of unique elements.
- Your algorithm's time complexity **must be** better than O(*n* log *n*), where *n* is the array's size.

Solution 1

Idea is simple. Build a array of list to be buckets with length 1 to sort.

```java
public List<Integer> topKFrequent(int[] nums, int k) {

    List<Integer>[] bucket = new List[nums.length + 1];
    Map<Integer, Integer> frequencyMap = new HashMap<Integer, Integer>();

    for (int n : nums) {
        frequencyMap.put(n, frequencyMap.getOrDefault(n, 0) + 1);
    }

    for (int key : frequencyMap.keySet()) {
        int frequency = frequencyMap.get(key);
        if (bucket[frequency] == null) {
            bucket[frequency] = new ArrayList<>();
        }
        bucket[frequency].add(key);
    }

    List<Integer> res = new ArrayList<>();

    for (int pos = bucket.length - 1; pos >= 0 && res.size() < k; pos--) {
        if (bucket[pos] != null) {
            res.addAll(bucket[pos]);
        }
    }
    return res;
}
```

written by mo10 (https://leetcode.com/discuss/user/mo10) original link here (https://leetcode.com/discuss/100581/java-o-n-solution-bucket-sort)

Solution 2

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int,int> map;
        for(int num : nums){
            map[num]++;
        }

        vector<int> res;
        // pair<first, second>: first is frequency,  second is number
        priority_queue<pair<int,int>> pq;
        for(auto it = map.begin(); it != map.end(); it++){
            pq.push(make_pair(it->second, it->first));
            if(pq.size() > (int)map.size() - k){
                res.push_back(pq.top().second);
                pq.pop();
            }
        }
        return res;
    }
};
```

written by sxycwzwzq (https://leetcode.com/discuss/user/sxycwzwzq) original link here (https://leetcode.com/discuss/100562/o-log-k-unordered_map-and-priority_queue-maxheap-solution)

Solution 3

```
class Solution {
public:
    vector<int> topKFrequent(vector<int>& nums, int k) {
        unordered_map<int, int> m;
        for (int num : nums)
            ++m[num];

        vector<vector<int>> buckets(nums.size() + 1);
        for (auto p : m)
            buckets[p.second].push_back(p.first);

        vector<int> ans;
        for (int i = buckets.size() - 1; i >= 0 && ans.size() < k; --i) {
            for (int num : buckets[i]) {
                ans.push_back(num);
                if (ans.size() == k)
                    break;
            }
        }
        return ans;
    }
};
```

written by Aeonaxx (https://leetcode.com/discuss/user/Aeonaxx) original link here (https://leetcode.com/discuss/100769/simple-c-solution-using-hash-table-and-bucket-sort)