LeetCode 347 [Top K Frequent Elements]

Given an integer array **nums** and an integer **k**, return the **k** most frequent elements. You may return the answer in any order.

| 1 | 1 | 1 | 2 | 2 | 3 |

K=2

1=3 times
2=2 times
3=1 times

Ans: [1,2]

Hashmap + Heap (Priority Queue)

| 1 | 3 | 4 | 3 | 4 | 2 | 3 | 4 | 2 | 5 | 4 | 5 | 5 |

K=3

HashMap

1	1
2	2
3	3
4	4
5	3

Sorted array

4	3	5	2	1

[4,3,5] ← Ans

Time= n log n

Instead of this approach we will use a Priority Queue

1	2	3	
1	2	3	← 4
2	3	4	← 5

Result →

3	3	4
5	3	4

Time: O(n log K)
Space: O(n)

Java Solution

```
public int[] topKFrequent(int[] nums, int k){
    if(k == nums.length){
        return nums;
    }

    Map<Integer, Integer> count = new HashMap<>();

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

    Queue<Integer> heap = new PriorityQueue<>(Comparator.
                                    comparingInt(count::get));

    for(int n : count.keySet()){
        heap.add(n);
        if(heap.size() > k){
            heap.poll();
        }
    }

    int[] ans = new int[k];
    for(int i=0; i < k; i++){
        ans[i] = heap.poll();
    }

    return ans;
}
```

Kotlin Solution

```
fun topKFrequent(nums: IntArray, k: Int): IntArray {
    if (nums.size == k) return nums

    val count = mutableMapOf<Int, Int>()

    for (num in nums) {
        count[num] = count.getOrDefault(num,0) +1
    }

    val heap = PriorityQueue<Int>(compareBy { count[it] })

    for (n in count.keys) {
        heap.add(n)
        if(heap.size > k) {
            heap.poll()
        }
    }

    val ans = IntArray(k)
    for (i in 0 until k) {
        ans[i] = heap.poll()
    }

    return ans
}
```

Map to count each number frequency

Min Heap to keep k frequent

We add each key from map to heap is the length is greater than k, we delete less frequent element

Array with our answer