LeetCode 347 — Top K Frequent Elements

(Link: https://leetcode.com/problems/top-k-frequent-elements/)

1. Problem Title & Link

Title: LeetCode 347: Top K Frequent Elements

Link: https://leetcode.com/problems/top-k-frequent-elements/

2. Problem Statement (Short Summary)

Given an array of integers, return the **k most frequent elements**.

The output **order does not matter**, only the top k highest-frequency elements must be returned.

3. Examples (Input → Output)

Example 1

Input: nums = [1,1,1,2,2,3], k = 2

Output: [1,2]

Explanation:

- freq(1) = 3
- freq(2) = 2
- Top 2 frequent → 1 and 2

Example 2

Input: nums = [1], k = 1

Output: [1]

Example 3

Input: nums = [4,4,4,5,5,6], k=1

Output: [4]

4. Constraints

- $1 \le nums.length \le 10^5$
- $-10^4 \le nums[i] \le 10^4$
- $1 \le k \le$ unique elements count
- Must run **better than O(n log n)** (i.e., O(n) or O(n log k))

5. Core Concept (Pattern / Topic)

⭐ **Heap / Bucket Sort / Hashmap Counting**

6. Thought Process (Step-by-Step Explanation)

Brute Force Approach (Slow)

1. Count frequency of each element
2. Sort by frequency (descending) → O(n log n)
3. Pick first k → works but too slow for large inputs

Optimized Thinking

We don't need full sorting.
We only need **top k**, not sorted frequency list.

Better Approach 1 — Min Heap (O(n log k))

- Push (frequency, element) to a **min-heap**
- Keep heap size ≤ k
- Pop smallest when exceeding
- Remaining k elements = top frequent

Better Approach 2 — Bucket Sort (O(n))

- Maximum frequency = n
- Make buckets: index = frequency
- Fill buckets based on freq
- Traverse from high freq → low freq
- Pick k elements
- Achieves **O(n)**

7. Visual / Intuition Diagram

Frequency Count

nums = [1,1,1,2,2,3]

freq:

1 → 3

2 → 2

3 → 1

Bucket Sort

Index: 0 1 2 3

Bucket: _ [3] [2] [1]

Meaning:

3 → [1]

2 → [2]

1 → [3]

Pick from rightmost until k.

8. Pseudocode (Language Independent)

```
count = frequency map of nums
bucket = array of size n+1

for (num, f) in count:
    bucket[f].append(num)

result = []
for freq from n down to 1:
    for num in bucket[freq]:
        result.append(num)
        if len(result) == k:
            return result
```

9. Code Implementation

✅ **Python**

```
from collections import Counter

class Solution:
    def topKFrequent(self, nums, k):
        freq = Counter(nums)
        bucket = [[] for _ in range(len(nums) + 1)]

        for num, f in freq.items():
            bucket[f].append(num)

        res = []
        for f in range(len(bucket) - 1, 0, -1):
            for num in bucket[f]:
                res.append(num)
                if len(res) == k:
                    return res
```

✅ **Java**

```
import java.util.*;

class Solution {
    public int[] topKFrequent(int[] nums, int k) {
        Map<Integer, Integer> freq = new HashMap<>();
        for (int num : nums)
            freq.put(num, freq.getOrDefault(num, 0) + 1);
```

```
        List<Integer>[] bucket = new ArrayList[nums.length + 1];
        for (int i = 0; i <= nums.length; i++)
            bucket[i] = new ArrayList<>();

        for (int num : freq.keySet())
            bucket[freq.get(num)].add(num);

        int[] res = new int[k];
        int idx = 0;

        for (int f = nums.length; f >= 1; f--) {
            for (int num : bucket[f]) {
                res[idx++] = num;
                if (idx == k) return res;
            }
        }
        return res;
    }
}
```

10. Time & Space Complexity

Bucket Sort Approach

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

Why?
We store frequencies and buckets up to n.

11. Common Mistakes / Edge Cases

❌ Sorting whole list → O(n log n) (too slow)

❌ Forgetting that multiple numbers can have same frequency

❌ Using max-heap of full size instead of k

❌ Returning sorted order (order doesn't matter)

Edge cases:
✔ k = 1
✔ All elements same
✔ All elements unique

12. Detailed Dry Run

nums = [1,1,1,2,2,3], k=2

Step	Action	Structure
Count freq	{1:3, 2:2, 3:1}	freq
Bucket fill	bucket[3] = [1], bucket[2] = [2], bucket[1] = [3]	bucket
Traverse	Start from freq=6→0	
freq=3	Add 1	res=[1]
freq=2	Add 2	res=[1,2] → STOP

Output: **[1,2]**

13. Common Use Cases (Real-Life / Interview)

- Finding top-k frequent search queries
- Top-k frequent logs / error codes
- Frequency-based ranking
- Word frequency counters

14. Common Traps (Important!)

⚠ Using dictionary sorting incorrectly

⚠ Forgetting to handle negative numbers

⚠ Not considering multiple numbers having same freq

⚠ Min-heap popping incorrectly

15. Builds To (Related LeetCode Problems)

- LC 451 — Sort Characters By Frequency
- LC 692 — Top K Frequent Words
- LC 973 — K Closest Points
- LC 215 — Kth Largest Element (Heap mastery)

16. Alternate Approaches + Comparison

Approach	Time	Space	When to Use
Bucket Sort	O(n)	O(n)	Best for this problem
Min Heap	O(n log k)	O(k)	When k is small
Sorting	O(n log n)	O(n)	Simple but slower

17. Why This Solution Works (Short Intuition)

Frequencies range from 1 to n, so we group elements by frequency and pick from highest frequency bucket downward — giving top k efficiently without sorting.

18. Variations / Follow-Up Questions

- What if the output must be **sorted by frequency**?
- What if we need **streaming input**? (Use min-heap)
- What if k is very large?
- Return top k least frequent?
- Return top k frequent words (string version)?