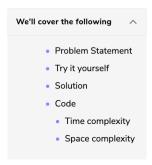




# Top 'K' Frequent Numbers (medium)



## Problem Statement

Given an unsorted array of numbers, find the top 'K' frequently occurring numbers in it.

### Example 1:

```
Input: [1, 3, 5, 12, 11, 12, 11], K = 2
Output: [12, 11]
Explanation: Both '11' and '12' apeared twice.
```

#### Example 2:

```
Input: [5, 12, 11, 3, 11], K = 2
Output: [11, 5] or [11, 12] or [11, 3]
Explanation: Only '11' appeared twice, all other numbers appeared once.
```

# Try it yourself

Try solving this question here:

# Solution |

This problem follows Top 'K' Numbers. The only difference is that in this problem, we need to find the most frequently occurring number compared to finding the largest numbers.

We can follow the same approach as discussed in the **Top K Elements** problem. However, in this problem, we first need to know the frequency of each number, for which we can use a **HashMap**. Once we have the frequency map, we can use a **Min Heap** to find the 'K' most frequently occurring number. In the **Min Heap**, instead of comparing numbers we will compare their frequencies in order to get frequently occurring numbers

### Code 1

Here is what our algorithm will look like:

```
Python3 G C++
   import java.util.*;
  class TopKFrequentNumbers {
    public static List<Integer> findTopKFrequentNumbers(int[] nums, int k) {
      Map<Integer, Integer> numFrequencyMap = new HashMap<>();
      for (int n : nums)
       numFrequencyMap.put(n, numFrequencyMap.getOrDefault(n, 0) + 1);
      for (Map.Entry<Integer, Integer> entry : numFrequencyMap.entrySet()) {
       minHeap.add(entry);
        if (minHeap.size() > k) {
         minHeap.poll();
      List<Integer> topNumbers = new ArrayList<>(k);
      while (!minHeap.isEmpty()) {
        topNumbers.add(minHeap.poll().getKey());
            topNumbers;
Run
                                                                        Save Reset []
```

### Time complexity

The time complexity of the above algorithm is O(N + N \* log K).

### Space complexity

The space complexity will be O(N). Even though we are storing only 'K' numbers in the heap. For the frequency map, however, we need to store all the 'N' numbers.

