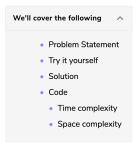
# Grokking the Coding Interview: Patterns for **Coding Questions** 76% completed Q Search Course Pattern: Tree Depth First Search Pattern: Two Heaps Pattern: Subsets Pattern: Modified **Binary Search** Pattern: Bitwise XOR Pattern: Top 'K' Elements Top 'K' Numbers (easy) Kth Smallest Number (easy) 'K' Closest Points to the Origin Connect Ropes (easy) Top 'K' Frequent Numbers Frequency Sort (medium) Kth Largest Number in a Stream 'K' Closest Numbers (medium) Maximum Distinct Elements Sum of Elements (medium) Rearrange String (hard) Problem Challenge 1 Solution Review: Problem Challenge 1 Problem Challenge 2 Solution Review: Problem Challenge 2 Problem Challenge 3 Solution Review: Problem Challenge 3 Pattern: K-way merge Introduction Merge K Sorted Lists (medium) Kth Smallest Number in M Sorted Lists (Medium) Kth Smallest Number in a Sorted Matrix (Hard) Smallest Number Range (Hard) Problem Challenge 1 Solution Review: Problem Challenge 1 Pattern : 0/1 Knapsack (Dynamic Programming)

0/1 Knapsack (medium)

# 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:

```
1 const find_k_frequent_numbers = function(nums, k) {
2    topNumbers = [];
3    // TODDO: Write your code here
4    return topNumbers;
5  };
6
7
8    console.log(`Here are the K frequent numbers: ${find_k_frequent_numbers([1, 3, 5, 12, 11, 12, 11], 2)}`)
9    console.log(`Here are the K frequent numbers: ${find_k_frequent_numbers([5, 12, 11, 3, 11], 2)}`)
10

RUN

SAVE RESET :
```

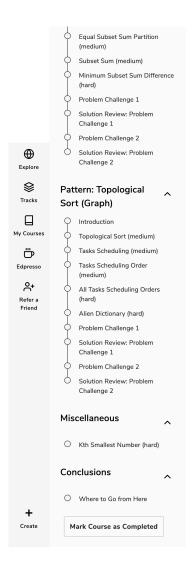
### Solution

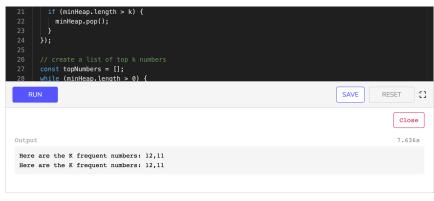
This problem follows Top 'K' Numbers. The only difference is that in this problem, we need to find the most frequently occurring number compared to finding the largest numbers.

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

Here is what our algorithm will look like:





## 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.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you  $\quad \times$  applying to them. See how  $\odot$ 

