

Grokking the Coding Interview: Patterns for Coding Questions

76% completed



First Search

Pattern: Tree Depth First Search



Pattern: Two Heaps



Pattern: Subsets



Pattern: Modified Binary Search



Pattern: Bitwise XOR



Pattern: Top 'K' Elements



- Introduction
- Top 'K' Numbers (easy)
- Kth Smallest Number (easy)
- 'K' Closest Points to the Origin (easy)
- Connect Ropes (easy)
- Top 'K' Frequent Numbers (medium)**
- Frequency Sort (medium)
- Kth Largest Number in a Stream (medium)
- 'K' Closest Numbers (medium)
- Maximum Distinct Elements (medium)
- 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)



- Introduction
- 0/1 Knapsack (medium)

Top 'K' Frequent Numbers (medium)

We'll cover the following

- Problem Statement
- Try it yourself
- Solution
- Code
 - Time complexity
 - Space complexity

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' appeared 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:

JavaPython3JS C++

```
1 const find_k_frequent_numbers = function(nums, k) {
2   topNumbers = [];
3   // TODO: 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
```

RUNSAVERESET

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:

JavaPython3C++JS

```
1 const Heap = require('./collections/heap'); //http://www.collectionsjs.com
2
3
4 function find_k_frequent_numbers(nums, k) {
5   // find the frequency of each number
6   numFrequencyMap = {};
7   nums.forEach((num) => {
8     if (!(num in numFrequencyMap)) {
9       numFrequencyMap[num] = 1;
10    } else {
11      numFrequencyMap[num]++;
12    }
13  });
14
15  const minHeap = new Heap([], null, ((a, b) => b[0] - a[0]));
16
17  // go through all numbers of the numFrequencyMap and push them in the minHeap, which will have
18  // top k frequent numbers. If the heap size is more than k, we remove the smallest(top) number
19  Object.keys(numFrequencyMap).forEach((num) => {
20    minHeap.push([numFrequencyMap[num], num]);
21  });
22}
```

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

Equal Subset Sum Partition (medium)

Subset Sum (medium)

Minimum Subset Sum Difference (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Pattern: Topological Sort (Graph)

Introduction

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Miscellaneous

Kth Smallest Number (hard)

Conclusions

Where to Go from Here

Mark Course as Completed

21if (minHeap.length > k) {

22minHeap.pop();

23}

24});

25

26// create a list of top k numbers

27const topNumbers = [];

28while (minHeap.length > 0) {

RUN

SAVE

RESET

Close

Output7.636s

Here are the K frequent numbers: 12,11

Here are the K frequent numbers: 12,11

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 applying to them. [See how](#)

Back

Next

Connect Ropes (easy)

Frequency Sort (medium)

Mark as Completed

Report an Issue

Ask a Question