

Grokking the Coding Interview: Patterns for Coding Questions

77% completed



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)
- Equal Subset Sum Partition (medium)

Frequency Sort (medium)

We'll cover the following

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

Problem Statement

Given a string, sort it based on the decreasing frequency of its characters.

Example 1:

```
Input: "Programming"
Output: "rrggmmPiano"
Explanation: 'r', 'g', and 'm' appeared twice, so they need to appear before any other character.
```

Example 2:

```
Input: "abcbab"
Output: "bbbaac"
Explanation: 'b' appeared three times, 'a' appeared twice, and 'c' appeared only once.
```

Try it yourself

Try solving this question here:

Java

Python3

JS

C++

```
1 const sort_character_by_frequency = function(str) {
2   // TODO: Write your code here
3   return "";
4 };
5
6
7 console.log(`String after sorting characters by frequency: ${sort_character_by_frequency("Programming")}`);
8 console.log(`String after sorting characters by frequency: ${sort_character_by_frequency("abcbab")}`);
9
```

RUN

SAVE

RESET

Solution

This problem follows the **Top 'K' Elements** pattern, and shares similarities with [Top 'K' Frequent Numbers](#).

We can follow the same approach as discussed in the [Top 'K' Frequent Numbers](#) problem. First, we will find the frequencies of all characters, then use a max-heap to find the most occurring characters.

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```
1 const Heap = require('./collections/heap'); //http://www.collectionsjs.com
2
3
4 function sort_character_by_frequency(str) {
5   // find the frequency of each character
6   charFrequencyMap = {};
7   for (i = 0; i < str.length; i++) {
8     const chr = str[i];
9     if (chr in charFrequencyMap) {
10      charFrequencyMap[chr]++;
11     } else {
12      charFrequencyMap[chr] = 1;
13     }
14   }
15
16   // add all characters to the max heap
17   const maxHeap = new Heap([], null, ((a, b) => a[0] - b[0]));
18   Object.keys(charFrequencyMap).forEach((key) => {
19     maxHeap.push([charFrequencyMap[key], key]);
20   });
21
22   // build a string, appending the most occurring characters first
23   const sortedString = [];
24   while (maxHeap.length > 0) {
25     [frequency, char] = maxHeap.pop();
26     for (let i = 0; i < frequency; i++) {
```

Explore

Tracks

My Courses

Edpresso

Refer a Friend

+

Create

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

26for (let i = 0; i < frequency; i++) {

27 sortedString.push(char);

28}

RUN

SAVE

RESET

Close

Output3.823s

String after sorting characters by frequency: rrggmmaoPin

String after sorting characters by frequency: bbbaac

Time complexity

The time complexity of the above algorithm is $O(D * \log D)$ where 'D' is the number of distinct characters in the input string. This means, in the worst case, when all characters are unique the time complexity of the algorithm will be $O(N * \log N)$ where 'N' is the total number of characters in the string.

Space complexity

The space complexity will be $O(N)$, as in the worst case, we need to store all the 'N' characters in the HashMap.

Interviewing soon? We've partnered with Hired so that companies apply to you instead of you applying to them. [See how](#)

Back

Next

Top 'K' Frequent Numbers (medium)

Kth Largest Number in a Stream (medi...

Mark as Completed

Report an Issue

Ask a Question