

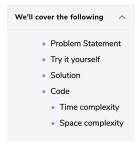


Grokking the Coding Interview: Patterns for **Coding Questions** 77% completed Q Search Course Pattern: Tree Depth First Search Pattern: Two Heaps Pattern: Subsets Pattern: Modified **Binary Search** Pattern: Bitwise XOR Pattern: Top 'K' Elements Introduction Top 'K' Numbers (easy) Kth Smallest Number (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 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)



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 characte r.
```

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:



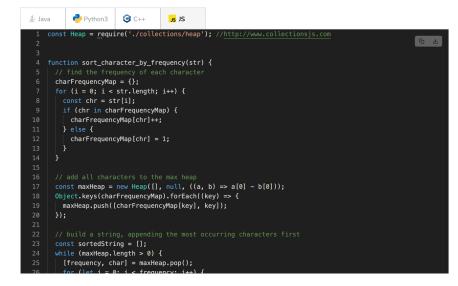
Solution

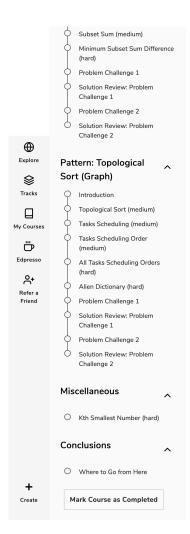
This problem follows the Top 'K' Elements pattern, and shares similarities with Top 'K' Frequent Numbers.

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:







Time complexity

The time complexity of the above algorithm is O(D*log D) where ${\mathbb D}'$ 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 ${\mathbb N}'$ 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 $\quad\times$ applying to them. See how \odot

