

# cducative Grokking the Coding Interview: Patterns for **Coding Questions** C 81% 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) (medium) Frequency Sort (medium) Kth Largest Number in a Stream 'K' Closest Numbers (medium) Maximum Distinct Elements (medium) Sum of Elements (medium) Rearrange String (hard) Problem Challenge 1 Solution Review: Problem Challenge 1 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 Smallest Number Range (Hard) Problem Challenge 1

Solution Review: Problem

Pattern: 0/1 Knapsack

0/1 Knapsack (medium)

Equal Subset Sum Partition

Challenge 1

Introduction

(medium)

(Dynamic Programming)

## Solution Review: Problem Challenge 1



#### Rearrange String K Distance Apart (hard)

Given a string and a number 'K', find if the string can be rearranged such that the same characters are at least 'K' distance apart from each other.

#### Example 1:

```
Input: "mmpp", K=2
Output: "mpmp" or "pmpm"
Explanation: All same characters are 2 distance apart.
```

#### Example 2:

```
Output: "rgmPrgmiano" or "gmringmrPoa" or "gmrPagimnor" and a few more
Explanation: All same characters are 3 distance apart.
```

#### Example 3:

```
Explanation: All same characters are 2 distance apart.
```

### Example 4:

```
Input: "aappa", K=3
Explanation: We cannot find an arrangement of the string where any two 'a' are 3 distance apar
```

## Solution

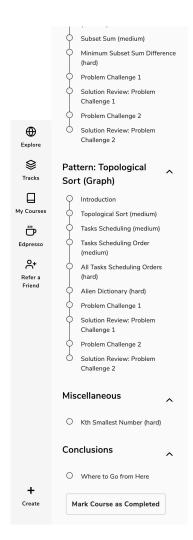
This problem follows the Top 'K' Numbers pattern and is quite similar to Rearrange String. The only difference is that in the 'Rearrange String' the same characters need not be adjacent i.e., they should be at least '2' distance apart (in other words, there should be at least one character between two same characters), while in the current problem, the same characters should be 'K' distance apart.

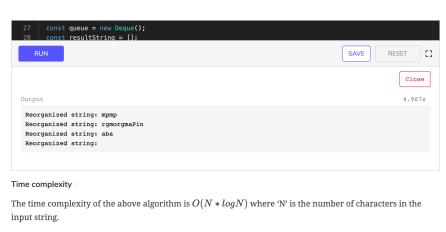
Following a similar approach, since we were inserting a character back in the heap in the next iteration, in this problem, we will re-insert the character after 'K' iterations. We can keep track of previous characters in a queue to insert them back in the heap after 'K' iterations.

### Code

Here is what our algorithm will look like:

```
Python3
                     ⊘ C++
function reorganize_string(str, k) {
   return str;
 charFrequencyMap = {};
for (i = 0; i < str.length; i++) {</pre>
    if (!(chr in charFrequencyMap)) {
     charFrequencyMap[chr] = 1;
      charFrequencyMap[chr]++;
 const maxHeap = new Heap([], null, ((a, b) \Rightarrow a[0] - b[0]));
 Object.keys(charFrequencyMap).forEach((char) => {
    maxHeap.push([charFrequencyMap[char], char]);
```





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  $\odot$ 

