

Grokking the Coding Interview: Patterns for Coding Questions

82% completed



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)
- Subset Sum (medium)
- Minimum Subset Sum Difference (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Solution Review: Problem Challenge 2

We'll cover the following



- Scheduling Tasks (hard)
- Solution
- Code
 - Time complexity
 - Space complexity

Scheduling Tasks (hard)

You are given a list of tasks that need to be run, in any order, on a server. Each task will take one CPU interval to execute but once a task has finished, it has a cooling period during which it can't be run again. If the cooling period for all tasks is 'K' intervals, find the minimum number of CPU intervals that the server needs to finish all tasks.

If at any time the server can't execute any task then it must stay idle.

Example 1:

```
Input: [a, a, a, b, c, c], K=2
Output: 7
Explanation: a -> c -> b -> a -> c -> idle -> a
```

Example 2:

```
Input: [a, b, a], K=3
Output: 5
Explanation: a -> b -> idle -> idle -> a
```


Solution


This problem follows the [Top 'K' Elements](#) pattern and is quite similar to [Rearrange String K Distance Apart](#). We need to rearrange tasks such that same tasks are 'K' distance apart.


Following a similar approach, we will use a **Max Heap** to execute the highest frequency task first. After executing a task we decrease its frequency and put it in a waiting list. In each iteration, we will try to execute as many as **k+1** tasks. For the next iteration, we will put all the waiting tasks back in the **Max Heap**. If, for any iteration, we are not able to execute **k+1** tasks, the CPU has to remain idle for the remaining time in the next iteration.


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 schedule_tasks(tasks, k) {
5      let intervalCount = 0;
6      taskFrequencyMap = {};
7      tasks.forEach((chr) => {
8          if (!(chr in taskFrequencyMap)) {
9              taskFrequencyMap[chr] = 1;
10         } else {
11             taskFrequencyMap[chr]++;
12         }
13     });
14
15
16     const maxHeap = new Heap([], null, ((a, b) => a[0] - b[0]));
17
18     // add all tasks to the max heap
19     Object.keys(taskFrequencyMap).forEach((char) => {
20         maxHeap.push([taskFrequencyMap[char], char]);
21     });
22
23     while (maxHeap.length > 0) {
24         const waitList = [];
25         let n = k + 1; // try to execute as many as 'k+1' tasks from the max-heap
26         while (n > 0 && maxHeap.length > 0) {
27             intervalCount++;
28             const [frequency, char] = maxHeap.pop();
```

RUN

SAVE

RESET



Close

Output7.399s

Minimum intervals needed to execute all tasks: 7
Minimum intervals needed to execute all tasks: 5

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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

Time complexity

The time complexity of the above algorithm is $O(N * \log N)$ where 'N' is the number of tasks. Our **while loop** will iterate once for each occurrence of the task in the input (i.e. 'N') and in each iteration we will remove a task from the heap which will take $O(\log N)$ time. Hence the overall time complexity of our algorithm is $O(N * \log N)$.

Space complexity

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

Problem Challenge 2

Problem Challenge 3

Mark as Completed

Report an Issue

Ask a Question