

Grokking the Coding Interview: Patterns for Coding Questions

64% completed



- Problem Challenge 2
- Solution Review: Problem Challenge 2**
- Problem Challenge 3
- Solution Review: Problem Challenge 3

Pattern: Modified Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- 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: Bitwise XOR

- Introduction
- Single Number (easy)
- Two Single Numbers (medium)
- Complement of Base 10 Number (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

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

Solution Review: Problem Challenge 2

We'll cover the following

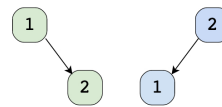
- Structurally Unique Binary Search Trees (hard)
- Solution
- Code
 - Time complexity
 - Space complexity
- Memoized version

Structurally Unique Binary Search Trees (hard)

Given a number 'n', write a function to return all structurally unique Binary Search Trees (BST) that can store values 1 to 'n'?

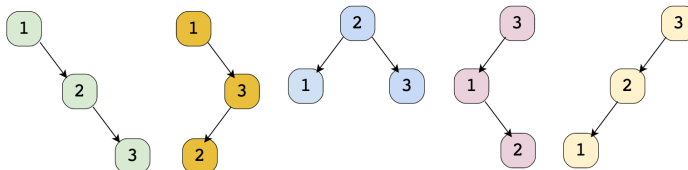
Example 1:

Input: 2
Output: List containing root nodes of all structurally unique BSTs.
Explanation: Here are the 2 structurally unique BSTs storing all numbers from 1 to 2:



Example 2:

Input: 3
Output: List containing root nodes of all structurally unique BSTs.
Explanation: Here are the 5 structurally unique BSTs storing all numbers from 1 to 3:



Solution

This problem follows the [Subsets](#) pattern and is quite similar to [Evaluate Expression](#). Following a similar approach, we can iterate from 1 to 'n' and consider each number as the root of a tree. All smaller numbers will make up the left sub-tree and bigger numbers will make up the right sub-tree. We will make recursive calls for the left and right sub-trees

Code

Here is what our algorithm will look like:

```
Java Python3 C++ JS
1 class TreeNode {
2     constructor(val, left = null, right = null) {
3         this.val = val;
4         this.left = left;
5         this.right = right;
6     }
7 }
8
9
10 function find_unique_trees(n) {
11     if (n <= 0) {
12         return [];
13     }
14     return findUnique_trees_recursive(1, n);
15 }
16
17 function findUnique_trees_recursive(start, end) {
18     const result = [];
19     // base condition, return 'null' for an empty sub-tree
20     // consider n = 1, in this case we will have start = end = 1, this means we should have only one tree
21     // we will have two recursive calls, findUniqueTreesRecursive(1, 0) & (2, 1)
22     // both of these should return 'null' for the left and the right child
23     if (start > end) {
```

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

Pattern: Topological Sort (Graph)

Introduction

Topological Sort (medium)

Tasks Scheduling (medium)

Tasks Scheduling Order (medium)

All Tasks Scheduling Orders (hard)

Alien Dictionary (hard)

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

24 result.push(null);

25 return result;

26 }

27

28 for (let i = start; i < end + 1; i++) {

RUN

SAVE

RESET

Close

Output

2.285s

Total trees: 2

Total trees: 5

Time complexity

The time complexity of this algorithm will be exponential and will be similar to [Balanced Parentheses](#). Estimated time complexity will be $O(n * 2^n)$ but the actual time complexity ($O(4^n / \sqrt{n})$) is bounded by the [Catalan number](#) and is beyond the scope of a coding interview. See more details [here](#).

Space complexity

The space complexity of this algorithm will be exponential too, estimated at $O(2^n)$, but the actual will be ($O(4^n / \sqrt{n})$).

Memoized version

Since our algorithm has overlapping subproblems, can we use memoization to improve it? We could, but every time we return the result of a subproblem from the cache, we have to clone the result list because these trees will be used as the left or right child of a tree. This cloning is equivalent to reconstructing the trees, therefore, the overall time complexity of the memoized algorithm will also be the same.

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