Solution Review: Problem Challenge 2

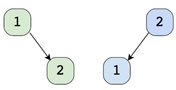
We'll cover the following

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'?

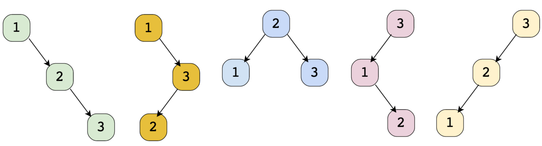
Example 1:

```
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

```java
import java.util.*;

class TreeNode {
  int val;
  TreeNode left;
  TreeNode right;

  TreeNode(int x) {
    val = x;
  }
};

class UniqueTrees {
  public static List<TreeNode> findUniqueTrees(int n) {
    if (n <= 0)
      return new ArrayList<TreeNode>();
    return findUniqueTreesRecursive(1, n);
  }

  public static List<TreeNode> findUniqueTreesRecursive(int start, int end) {
    List<TreeNode> result = new ArrayList<>();
    // base condition, return 'null' for an empty sub-tree
    // consider n=1, in this case we will have start=end=1, this means we should have only one tree
    // we will have two recursive calls, findUniqueTreesRecursive(1, 0) & (2, 1)
    // both of these should return 'null' for the left and the right child
    if (start > end) {
      result.add(null);
      return result;
```

Run | Save | Reset

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.