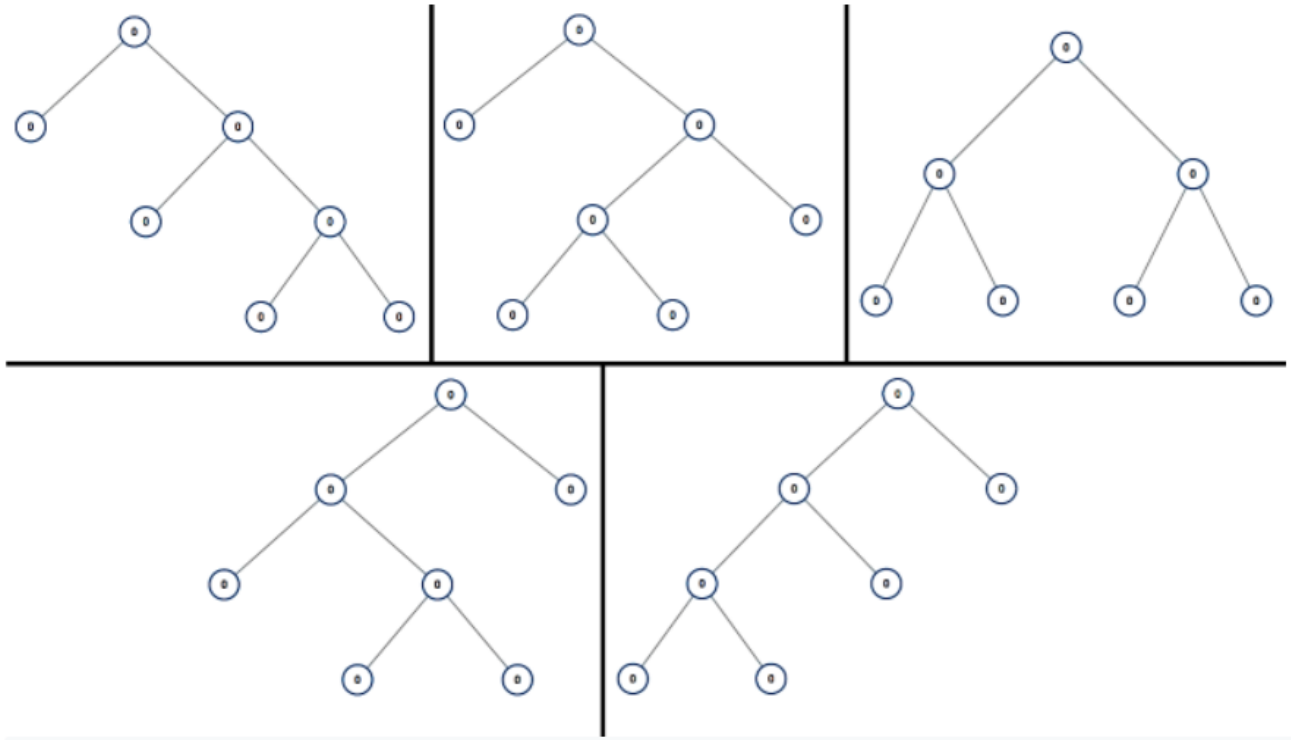
```
894. |  Medium  |  All Possible Full BInary Trees | Recursion.

Given an integer n, return a list of all possible full binary trees with n nodes.
Each node of each tree in the answer must have Node.val == 0.
Each element of the answer is the root node of one possible tree.
You may return the final list of trees in any order.

A full binary tree is a binary tree where each node has exactly 0 or 2 children.

Constraints:
1 <= n <= 20
```

Example 1:



Input: n = 7
Output: [[0,0,0,null,null,0,0,null,null,0,0],[0,0,0,null,null,0,0,0,0],[0,0,0,0,0,0,0],
[0,0,0,0,0,null,null,null,null,0,0],[0,0,0,0,0,null,null,0,0]]

Example 2:

Input: n = 3
Output: [[0,0,0]]

```cpp
vector<TreeNode*> allPossibleFBT(int n) {
    vector<TreeNode*> result, leftTree, rightTree;

    // base condition
    if(n == 1) {
        TreeNode* root = new TreeNode(0);
        result.push_back(root);
        return result;
    }

    // main logic
    for(int i=1; i<n; i += 2) {
        leftTree = allPossibleFBT(i);
        rightTree = allPossibleFBT(n-i-1);

        for(int j=0; j<leftTree.size(); j++) {
            for(int k=0; k<rightTree.size(); k++) {
                TreeNode* root = new TreeNode(0);
                root->left = leftTree[j];
                root->right = rightTree[k];
                result.push_back(root);
            }
        }
    }
    return result;
}
```

We can further optimise the time, space and memory with dynamic programming by just writing 2 line in the same code.

```cpp
unordered_map<int, vector<TreeNode*>> dp;
vector<TreeNode*> allPossibleFBT(int n) {
    vector<TreeNode*> result, leftTree, rightTree;

    // checking if present in the hashmap then don't calculate again.
    if(dp.count(n) > 0) {
        return dp[n];
    }

    // base condition
    if(n == 1) {
        TreeNode* root = new TreeNode(0);
        result.push_back(root);
        return result;
    }

    // main logic
    for(int i=1; i<n; i += 2) {
        leftTree = allPossibleFBT(i);
        rightTree = allPossibleFBT(n-i-1);

        for(int j=0; j<leftTree.size(); j++) {
            for(int k=0; k<rightTree.size(); k++) {
                TreeNode* root = new TreeNode(0);
                root->left = leftTree[j];
                root->right = rightTree[k];
                result.push_back(root);
            }
        }
    }

    // storing in the dp.
    dp.insert({n, result});
    return result;
}
```

#100daysofDSA





/rvislive

Rakesh Vishwakarma