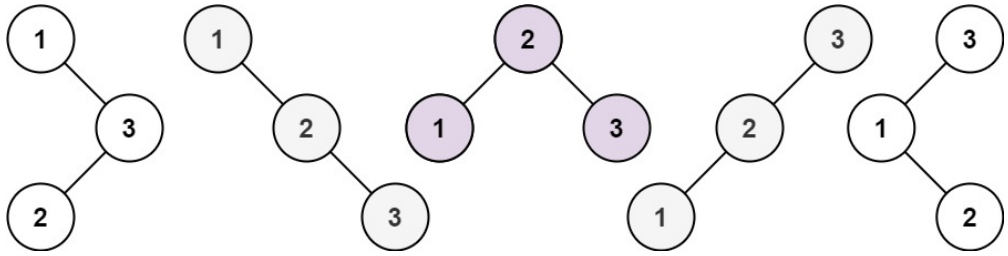
Day 42 / 100 :

Topic - BST, Array

1 Problem statement: [Unique Binary Search Tree II](#) (Medium)

Given an integer n, return all the structurally unique BST's (binary search trees), which has exactly n nodes of unique values from 1 to n. Return the answer in any order.

Example 1:



Input: n = 3
Output: [[1,null,2,null,3],[1,null,3,2],[2,1,3],[3,1,null,null,2],[3,2,null,1]]

Example 2:
Input: n = 1
Output: [[1]]

Solutions :
Approach 1 - Backtracking + Binary Tree

Intuition:

Given n, find all structurally unique BST's (binary search trees) that has n nodes with unique values from 1 to n in my case i take n as end number

Approach:

We will use a recursive approach generateTrees(int end, int start = 1)that receives a range (start to end, within n) and returns all BST's rooted in that range.

- The function generate Trees considers each number 'i' in the range [start, end] as the root of a binary search tree.
- For each 'i', it recursively generates all possible left subtrees from the values in the range [start, i - 1] and right subtrees from the values in the range [i + 1, end].
- It creates new trees with current node 'i' as the root and connects the left and right subtrees to it, forming all unique binary search trees for the current range.

- The process continues recursively until all possible combinations of unique binary search trees have been generated, and returns the vector 'ans' containing pointers to these trees.

Time Complexity:

- Time complexity:O(4^n)

```
class Solution {
public:
//copyright 2023 mahesh vishnoi
    vector<TreeNode*> generateTrees(int end, int start = 1) {
        vector<TreeNode*> ans;
        // If start > end, then subtree will be empty so we will directly return
null pointer
        if ( start> end)
            return {nullptr};

        // Consider every number in range [start, end] as root
        for (int i = start; i <= end; i++) {
            // generate all possible trees in range [start, i)
            for (auto left : generateTrees(i - 1, start)) {
                // generate all possible trees in range (i, end]
                for (auto right : generateTrees(end, i + 1))
                    // make new trees with 'i' as the root
                    ans.push_back(new TreeNode(i, left, right));
            }
        }
        return ans;
    }
};
```

Solutions :

Approach 2 - Divide and Conquer

Intuition:

As the test cases are small so we can go with the brute force

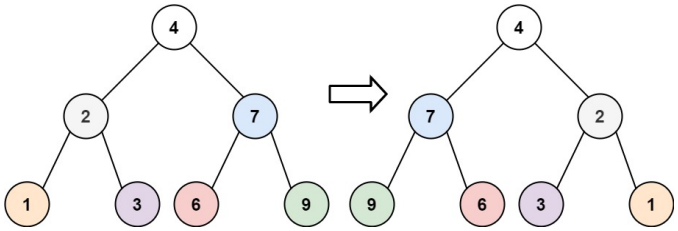
```
class Solution {
public:
    vector<TreeNode*>func(int low, int high){

        if(low>high)return {NULL};
```

```
        if(low==high)return {new TreeNode(low)};

        vector<TreeNode*>ans;
        for(int i = low; i<=high; i++){
            vector<TreeNode*>lefttrees = func(low,i-1);
            vector<TreeNode*>righttrees = func(i+1,high);

            for(auto x: lefttrees){
                for(auto y: righttrees){
                    TreeNode*currnode = new TreeNode(i);
                    currnode->left = x;
                    currnode->right = y;
                    ans.push_back(currnode);
                }
            }
        }

        return ans;
    }

    vector<TreeNode*> generateTrees(int n) {
        return func(1,n);
    }
};
```

2 Problem statement: [Invert the Binary tree](#) (Easy)

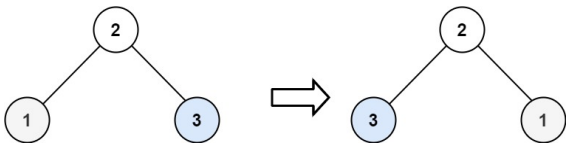
Given the root of a binary tree, invert the tree, and return its root.

Example 1:



Input: root = [4,2,7,1,3,6,9]
Output: [4,7,2,9,6,3,1]

Example 2:



Input: root = [2,1,3]
Output: [2,3,1]

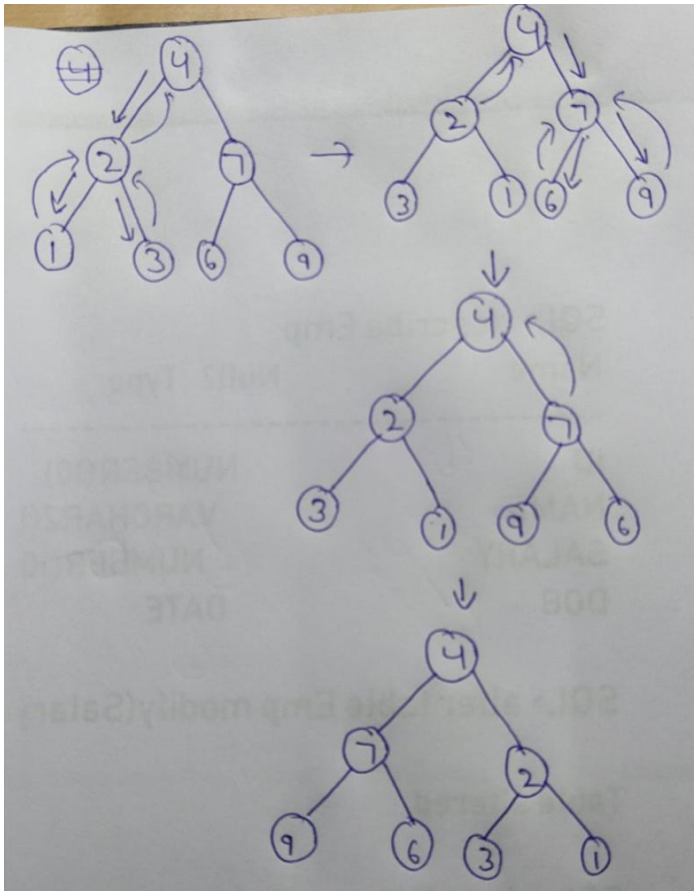
Example 3:
Input: root = []
Output: []

Solutions :

Approach 1 - Backtracking + Binary Tree

Intuition:
In this question, we have to invert the binary tree.
So we use Post Order Teversal in which first we go in Left subtree and then in Right subtree then we return to Parent node.
When we come back to the parent node, we swap its Left subtree and Right subtree.

Approach:



Time Complexity:

- Time complexity: O(N)

- Space complexity: O(N) Recursive stack space

```
C++
Python
class Solution {
public:
    TreeNode* invertTree(TreeNode* root) {
        // Base Case
        if(root==NULL)
            return NULL;
        invertTree(root->left); //Call the left substree
        invertTree(root->right); //Call the right substree
        // Swap the nodes
        TreeNode* temp = root->left;
        root->left = root->right;
        root->right = temp;
        return root; // Return the root
    }
};
```