Problems

Easy Problems:

1. **Print Leaf Nodes**: Write a function to print all leaf nodes of a binary tree. A leaf node is a node with no children. medium.com
2. **Check Node Existence**: Write a function to check if a given node exists in a binary tree. medium.com
3. **Count Leaf Nodes**: Write a function to count the number of leaf nodes in a given binary tree. medium.com
4. **Invert a Binary Tree**: Given the root of a binary tree, write a function to invert the tree and return its root. igotanoffer.com

Medium Problems:

1. **Finding Diameter of a Tree**: Write a function to find the diameter of a binary tree. The diameter of a binary tree is the length of the longest path between any two nodes in a tree. iq.opengenus.org
2. **Find Number of Universal Value Subtrees**: Write a function to find the number of universal value subtrees in a binary tree. A subtree is universal if all of its nodes have the same value. iq.opengenus.org
3. **Find if a given Binary Tree is a Sub-Tree of another Binary Tree**: Write a function to check if a given binary tree is a subtree of another binary tree. iq.opengenus.org
4. **Finding nodes at distance K from a given node**: Given a node in a binary tree, write a function to find all nodes that are a distance K from the given node. iq.opengenus.org

Hard Problems:

1. **Copy a binary tree where each node has a random pointer**: Given a binary tree where each node has a random pointer along with left and right pointers, write a function to create a deep copy of the tree. iq.opengenus.org
2. **Sum of k smallest elements in Binary Search Tree**: Given a binary tree, write a function to find the sum of the smallest k elements. iq.opengenus.org

High-level Explanation

Easy Problems:

1. **Print Leaf Nodes**: Traverse the entire tree and print the nodes that do not have any children. This can be done by recursively visiting each node and checking if both left and right child nodes are null.
2. **Check Node Existence**: Similar to the above, traverse the tree and compare the value of each node with the given value. If a match is found, return true. If the entire tree is traversed without finding the value, return false.
3. **Count Leaf Nodes**: This is similar to printing leaf nodes, but instead of printing, increment a counter when a leaf node is found.
4. **Invert a Binary Tree**: For each node in the tree, swap its left and right child nodes. This should be done recursively for all nodes starting from the root.

Medium Problems:

1. **Finding Diameter of a Tree**: The diameter of a tree is the maximum distance between any two leaf nodes. This can be calculated as the maximum of the sum of the heights of the left and right subtrees for all nodes.
2. **Find Number of Universal Value Subtrees**: A universal subtree is one where all nodes have the same value. To find these, traverse the tree and for each node, check if all nodes in its subtree have the same value.
3. **Find if a given Binary Tree is a Sub-Tree of another Binary Tree**: For each node in the main tree, check if the subtree rooted at that node is identical to the given tree.
4. **Finding nodes at distance K from a given node**: Start from the given node and perform a breadth-first search up to K levels deep. All nodes found at the Kth level are the required nodes.

Hard Problems:

1. **Copy a binary tree where each node has a random pointer**: Create a new node for each node in the original tree, copying over the value and the random pointer. The random pointer may need to be updated later if the node it points to hasn't been copied yet.
2. **Sum of k smallest elements in Binary Search Tree**: Traverse the tree in an inorder manner (left - root - right), which results in visiting nodes in ascending order. Keep a count of nodes visited and stop after visiting K nodes. The sum of these K nodes is the required sum.