**Assignment22**

**2.**

class Node {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

}

}

function flipBinaryTree(root) {

if (root === null || (root.left === null && root.right === null)) {

return root;

}

const flippedLeft = flipBinaryTree(root.left);

const flippedRight = flipBinaryTree(root.right);

root.left = flippedRight;

root.right = flippedLeft;

return root;

}

// Example:

const root = new Node(1);

root.left = new Node(2);

root.right = new Node(3);

root.left.left = new Node(4);

root.left.right = new Node(5);

const flippedRoot = flipBinaryTree(root);

function inOrderTraversal(node) {

if (node === null) {

return;

}

inOrderTraversal(node.left);

console.log(node.val);

inOrderTraversal(node.right);

}

inOrderTraversal(flippedRoot);

**3.**

class Node {

constructor(value) {

this.val = value;

this.left = null;

this.right = null;

}

}

function printRootToLeafPaths(root) {

if (root === null) {

return;

}

const stack = [];

stack.push({ node: root, path: `${root.val}` });

while (stack.length > 0) {

const { node, path } = stack.pop();

if (node.left === null && node.right === null) {

console.log(path);

}

if (node.right !== null) {

stack.push({ node: node.right, path: `${path}->${node.right.val}` });

}

if (node.left !== null) {

stack.push({ node: node.left, path: `${path}->${node.left.val}` });

}

}

}

// Example:

const root = new Node(6);

root.left = new Node(3);

root.right = new Node(5);

root.left.left = new Node(2);

root.left.right = new Node(5);

root.right.right = new Node(4);

root.left.right.left = new Node(7);

root.left.right.right = new Node(4);

printRootToLeafPaths(root);

**4.**

function checkTraversals(preorder, inorder, postorder) {

if (preorder.length !== inorder.length || inorder.length !== postorder.length) {

return false;

}

if (preorder.length === 0) {

return true;

}

const root = preorder[0];

const rootIndex = inorder.indexOf(root);

const leftInorder = inorder.slice(0, rootIndex);

const rightInorder = inorder.slice(rootIndex + 1);

const leftPreorder = preorder.slice(1, leftInorder.length + 1);

const rightPreorder = preorder.slice(leftInorder.length + 1);

const leftPostorder = postorder.slice(0, leftInorder.length);

const rightPostorder = postorder.slice(leftInorder.length, postorder.length - 1);

const leftSubtree = checkTraversals(leftPreorder, leftInorder, leftPostorder);

const rightSubtree = checkTraversals(rightPreorder, rightInorder, rightPostorder);

return leftSubtree && rightSubtree;

}

// Example:

const inorder1 = [4, 2, 5, 1, 3];

const preorder1 = [1, 2, 4, 5, 3];

const postorder1 = [4, 5, 2, 3, 1];

console.log(checkTraversals(preorder1, inorder1, postorder1));

Output: true

const inorder2 = [4, 2, 5, 1, 3];

const preorder2 = [1, 5, 4, 2, 3];

const postorder2 = [4, 1, 2, 3, 5];

console.log(checkTraversals(preorder2, inorder2, postorder2));

Output: false