**94. Binary Tree Inorder Traversal**

Given the root of a binary tree, return the inorder traversal of its nodes' values.

**Example 1:**



**Input:** root = [1,null,2,3]

**Output:** [1,3,2]

**Example 2:**

**Input:** root = []

**Output:** []

**Example 3:**

**Input:** root = [1]

**Output:** [1]

In binary tree traversal , first we process the left part of the tree, then the node itself then the right part.

/\*\*

\* Definition for a binary tree node.

\* function TreeNode(val, left, right) {

\* this.val = (val===undefined ? 0 : val)

\* this.left = (left===undefined ? null : left)

\* this.right = (right===undefined ? null : right)

\* }

\*/

/\*\*

\* @param {TreeNode} root

\* @return {number[]}

\*/

var inorderTraversal = function(root) {

let result = [];

inorder(root,result);

return result;

};

function inorder(root, result){

if(!root) return null;

inorder(root.left,result);

result.push(root.val);

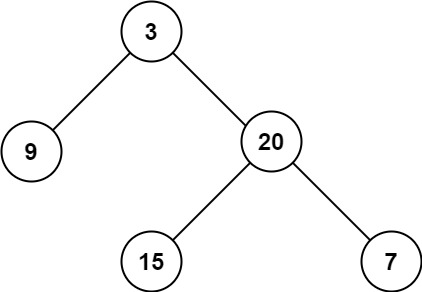
inorder(root.right,result);

}

Given the root of a binary tree, return *its maximum depth*.

A binary tree's **maximum depth** is the number of nodes along the longest path from the root node down to the farthest leaf node.

**Example 1:**



**Input:** root = [3,9,20,null,null,15,7]

**Output:** 3

**Example 2:**

**Input:** root = [1,null,2]

**Output:** 2

var maxDepth = function(root) {

if(!root){

return 0;

} else{

return Math.max(maxDepth(root.left)+1,maxDepth(root.right)+1)

}

};