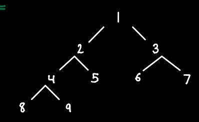
Node Depths

Input: tree =



Input: Takes in a binary tree

Output: returns the sum of its node's depth

Each binary tree node has an integer value, a left child node, and a right child node

Children nodes can be BinaryTree nodes themselves or None/null

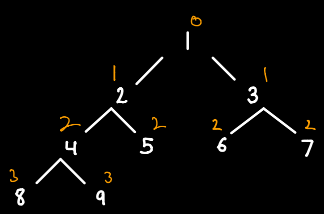
Output: 16

// The depth of the node value 2 is 1.
// The depth of the node value 3 is 1.
// The depth of the node value 4 is 2.
// The depth of the node value 5 is 2.
// Etc.
// Summing all of the depths yields 16

Recursive Solution: O(n) time ; O(h) space

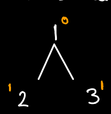
```javascript
// O(n) time | O(h) space where n is the number of
// nodes and h is the depth of the binary tree
function nodeDepths(root, depth = 0) {
  if (root === null) return 0

  return depth
    + nodeDepths(root.left, depth + 1)
    + nodeDepths(root.right, depth + 1)
}
```



Depth increases by 1 each time we move down the tree

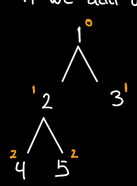
If we break the tree into just one parent with two children:



and want to get the sum of it's node depths we'd do:

$$\text{nodeDepths} = d^{0}_{parent} + d^{1}_{left\ child} + d^{1}_{right\ child}$$

If we add another branch, we would get the followind



$$\text{nodeDepths} = d^{0}_{parent} + d_{left\ child} + d^{1}_{right\ child}$$

$$d^{5}_{left\ child} = d^{1}_{parent} + d^{2}_{left\ child} + d^{2}_{right\ child}$$

This will Keep repeating recursively

For this example our answer is 6

Iterative Solution: O(n) time ; O(h) space

```javascript
// O(n) time | O(h) space where n is the number of
// nodes and h is the depth of the binary tree
function nodeDepths(root) {
  const callStack = [{node: root, depth: 0}];
  let sumOfDepths = 0;

  while (callStack.length > 0) {
    let poppedNode = callStack.pop();
    let node = poppedNode.node;
    let depth = poppedNode.depth;

    if (node === null) continue

    sumOfDepths += depth

    callStack.push({node: node.left, depth: depth + 1})
    callStack.push({node: node.right, depth: depth + 1})
  }

  return sumOfDepths
}
```

We use a Stack to solve this problem iteratively.

The Stack initially contains the root node and its depth.

sumOfDepths is also defined to keep track of the sum

We pop off the last element of the stack and get its depth / node value. If the node is null we return.

We then add its depth value to the sumOfDepths

We then push the node's children onto the stack while incrementing the depth value

Time is O(n) because we traverse every node in the tree

Space is O(h) because the call Stack, at max, contains elements that are equal to the height of the binary tree.