Description

129. Sum Root to Leaf Numbers

Given a binary tree containing digits from 0-9 only, each root-to-leaf path could represent a number.

An example is the root-to-leaf path 1->2->3 which represents the number 123.

Find the total sum of all root-to-leaf numbers.

Note: A leaf is a node with no children.

```
Example:
Input: [1,2,3]
    1
   / \
  2   3
Output: 25
Explanation:
The root-to-leaf path 1->2 represents the number 12.
The root-to-leaf path 1->3 represents the number 13.
Therefore, sum = 12 + 13 = 25.

Example 2:
Input: [4,9,0,5,1]
    4
   / \
  9   0
 / \
5   1
Output: 1026
Explanation:
The root-to-leaf path 4->9->5 represents the number 495.
The root-to-leaf path 4->9->1 represents the number 491.
The root-to-leaf path 4->0 represents the number 40.
Therefore, sum = 495 + 491 + 40 = 1026.
```

Idea

My primitive idea is to do a DFS and list all the root-to-leave paths. And then post process this list of paths to get the final sum. Which take O(n) time to traverse the tree, and O(H) space of recursion stack, and extra nlogn space to use a List to store all the paths. We don't really need to store all the paths, instead we can just sum up all the numbers while we going from top to bottom. Which will still take O(n) time, but only O(H) recursion space where n is the total number of nodes and H is the tree height. Take the path 4->9->5 for example. The first number we get is 4, and sum is 4. Then we get the second number 9. We multiply 10 to the current sum (4) to get 40 and add the current number 9 to get 49. We multiply 10 because we go down one level, the sum should actually be increased by 10. Then we go down to 5, use 49 * 10 + 5 to get 495 the path sum. Then if it is a leaf node, we add the path sum into the global sum to return.

Java

```
class Solution {

    public int sumNumbers(TreeNode root) {
        int[] res = new int[1];
        helper(root, res, 0);
        return res[0];
    }

    private void helper(TreeNode root, int[] res, int sum) {
        if (root == null) {
            return;
        }

        if (root.left == null && root.right == null) {
            res[0] += sum * 10 + root.val;
            return;
        }

        sum = sum * 10 + root.val;
        helper(root.left, res, sum);
        helper(root.right, res, sum);
    }
}
```

In C++ we can just use a reference instead of Java's int[] res.
C++

```
class Solution {
public:
    int sumNumbers(TreeNode* root) {
        int total = 0;
        helper(root, total, 0);
        return total;
    }

private:
    void helper(TreeNode* root, int& total, int sum) {
        if (root == NULL) {
            return;
        }

        if (root->left == NULL && root->right == NULL) {
            total += sum * 10 + root->val;
        }

        helper(root->left, total, sum * 10 + root->val);
        helper(root->right, total, sum * 10 + root->val);
    }
};
```

Summary

- Divid and conquer.