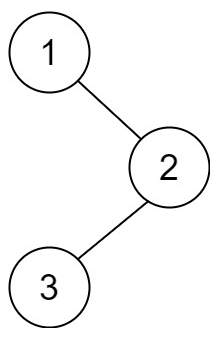
<https://leetcode.com/problems/binary-tree-postorder-traversal/>

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

**Example 1:**



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

Output: [3,2,1]

**Example 2:**

Input: root = []

Output: []

**Example 3:**

Input: root = [1]

Output: [1]

**Constraints:**

* The number of the nodes in the tree is in the range [0, 100].
* -100 <= Node.val <= 100

**Follow up:** Recursive solution is trivial, could you do it iteratively?

**Attempt 1: 2022-10-23**

**Solution 1:  Recursive traversal (10min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

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

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<Integer> postorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

if(root == null) {

return result;

}

// No modification on tree structure, can use original object 'root' to traverse

helper(root, result);

return result;

}

private void helper(TreeNode root, List<Integer> result) {

if(root == null) {

return;

}

helper(root.left, result);

helper(root.right, result);

result.add(root.val);

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Solution 2: Iterative traversal with Stack (10min)**

/\*\*

\* Definition for a binary tree node.

\* public class TreeNode {

\* int val;

\* TreeNode left;

\* TreeNode right;

\* TreeNode() {}

\* TreeNode(int val) { this.val = val; }

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

\* this.val = val;

\* this.left = left;

\* this.right = right;

\* }

\* }

\*/

class Solution {

public List<Integer> postorderTraversal(TreeNode root) {

List<Integer> result = new ArrayList<Integer>();

if(root == null) {

return result;

}

// Note that in this solution only right children are stored to stack

Stack<TreeNode> stack = new Stack<TreeNode>();

stack.push(root);

// No modification on tree structure, can use original object 'root' to traverse

while(!stack.isEmpty()) {

TreeNode node = stack.pop();

// Insert at the end instead of head

result.add(0, node.val);

if(node.left != null) {

stack.push(node.left);

}

if(node.right != null) {

stack.push(node.right);

}

}

return result;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/binary-tree-postorder-traversal/discuss/45551/Preorder-Inorder-and-Postorder-Iteratively-Summarization/188240>

**Use LinkedList and addFirst() method**

<https://leetcode.com/problems/binary-tree-postorder-traversal/discuss/45556/Java-simple-and-clean>

public List<Integer> postorderTraversal(TreeNode root) {

LinkedList<Integer> ans = new LinkedList<>();

Stack<TreeNode> stack = new Stack<>();

if (root == null) return ans;

stack.push(root);

while (!stack.isEmpty()) {

TreeNode cur = stack.pop();

ans.addFirst(cur.val);

if (cur.left != null) {

stack.push(cur.left);

}

if (cur.right != null) {

stack.push(cur.right);

}

}

return ans;

}