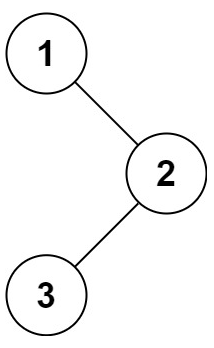
<https://leetcode.com/problems/binary-tree-inorder-traversal/>

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]

**Constraints:**

* The number of 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> inorderTraversal(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);

result.add(root.val);

helper(root.right, result);

}

}

**Test:**

import java.util.ArrayList;

import java.util.List;

public class Test {

public static void main(String[] args) {

/\*\*

1

\

3 ==> Expected: 1 2 3 4 5

/ \

2 4

\

5

\*/

Test b = new Test();

TreeNode one = b.new TreeNode(1);

TreeNode two = b.new TreeNode(2);

TreeNode three = b.new TreeNode(3);

TreeNode four = b.new TreeNode(4);

TreeNode five = b.new TreeNode(5);

one.right = three;

three.left = two;

three.right = four;

four.right = five;

List<Integer> result = b.inorderTraversal(one);

System.out.println(result);

}

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public List<Integer> inorderTraversal(TreeNode root) {

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

if (root == null) {

return result;

}

helper(root, result);

return result;

}

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

if (root == null) {

return;

}

helper(root.left, result);

result.add(root.val);

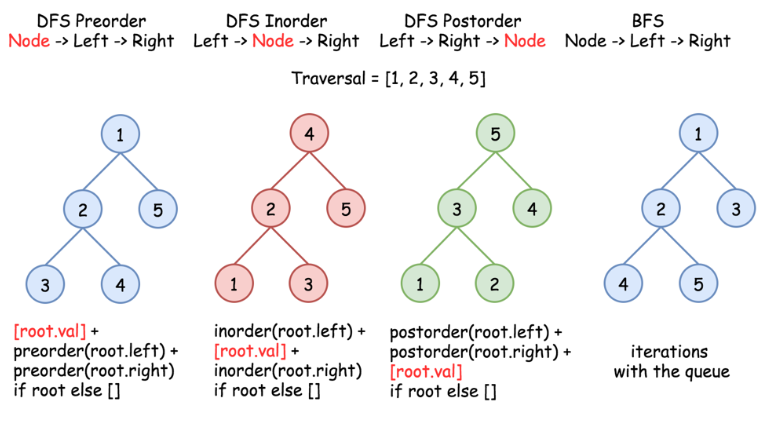
helper(root.right, result);

}

}

**Refer to**

<https://leetcode.com/problems/binary-tree-inorder-traversal/discuss/283746/All-DFS-traversals-(preorder-inorder-postorder)-in-Python-in-1-line>



def preorder(root):

return [root.val] + preorder(root.left) + preorder(root.right) if root else []

def inorder(root):

return inorder(root.left) + [root.val] + inorder(root.right) if root else []

def postorder(root):

return postorder(root.left) + postorder(root.right) + [root.val] if root else []

**Refer to**

<https://leetcode.com/problems/binary-tree-inorder-traversal/discuss/31231/C%2B%2B-Iterative-Recursive-and-Morris>

// Recursive solution

class Solution {

public:

vector<int> inorderTraversal(TreeNode\* root) {

vector<int> nodes;

inorder(root, nodes);

return nodes;

}

private:

void inorder(TreeNode\* root, vector<int>& nodes) {

if (!root) {

return;

}

inorder(root -> left, nodes);

nodes.push\_back(root -> val);

inorder(root -> right, nodes);

}

};

**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> inorderTraversal(TreeNode root) {

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

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

while(root != null || !stack.isEmpty()) {

while(root != null) {

stack.push(root);

root = root.left;

}

// Don't write as "TreeNode node = stack.pop()" which cannot update 'root'

// for next iteration, must update original iterative object 'root' by

// "root = stack.pop()"

root = stack.pop();

result.add(root.val);

root = root.right;

}

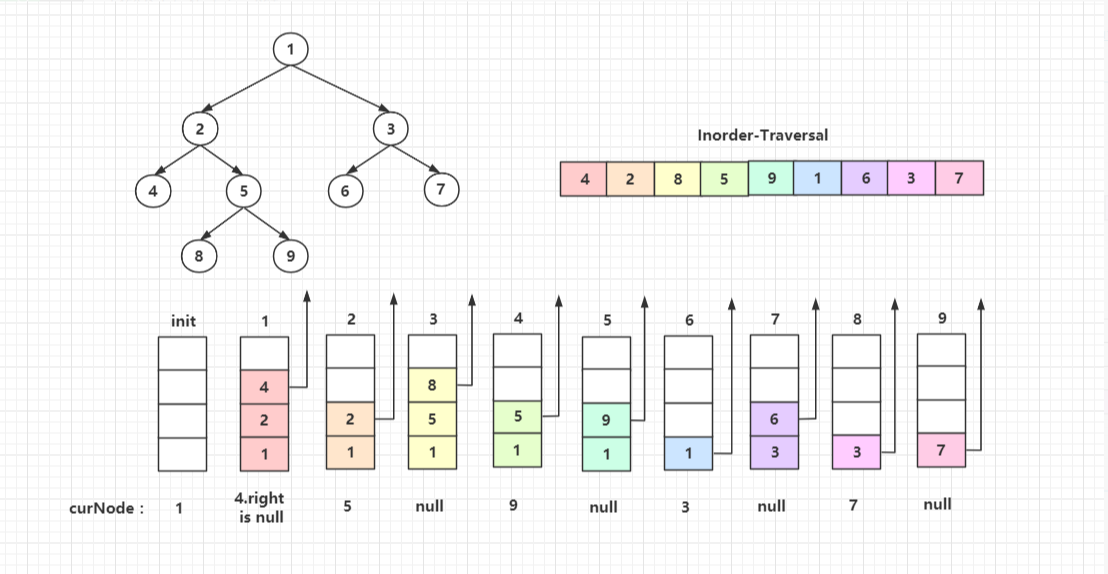
return result;

}

}

**Refer to**

<https://leetcode.com/problems/binary-tree-inorder-traversal/discuss/31213/Iterative-solution-in-Java-simple-and-readable>



public List<Integer> inorderTraversal(TreeNode root) {

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

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

TreeNode cur = root;

while(cur!=null || !stack.empty()){

while(cur!=null){

stack.add(cur);

cur = cur.left;

}

cur = stack.pop();

list.add(cur.val);

cur = cur.right;

}

return list;

}