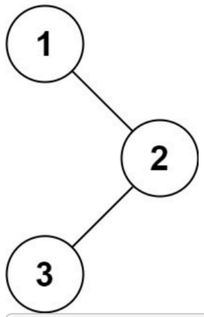
Binary Tree Inorder Traversal

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:

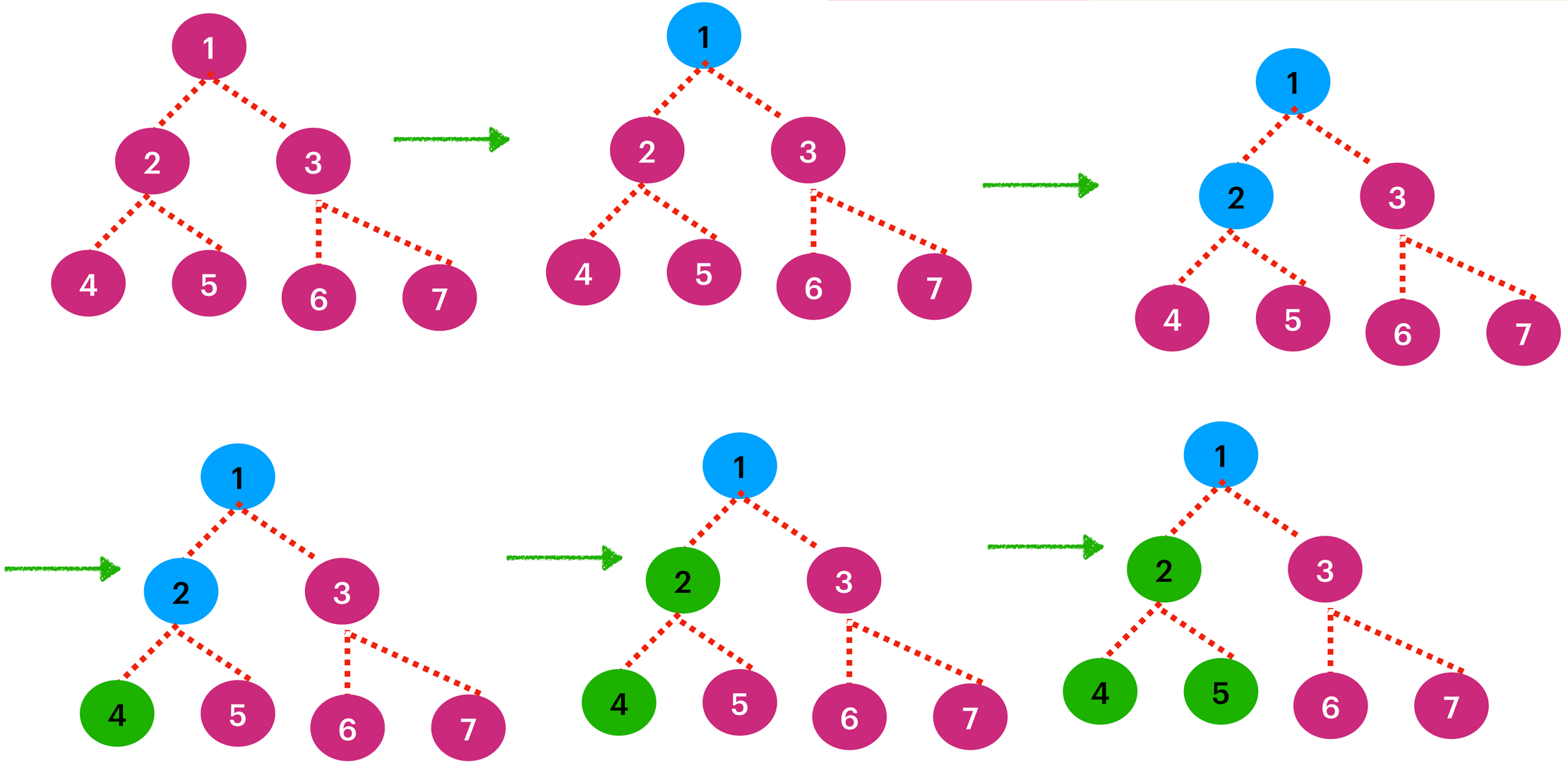
```
Input: root = [1]
Output: [1]
```

Constraints:

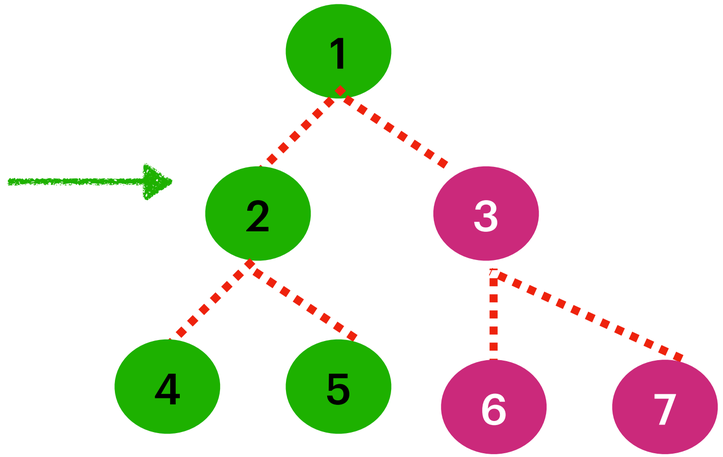
- The number of nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

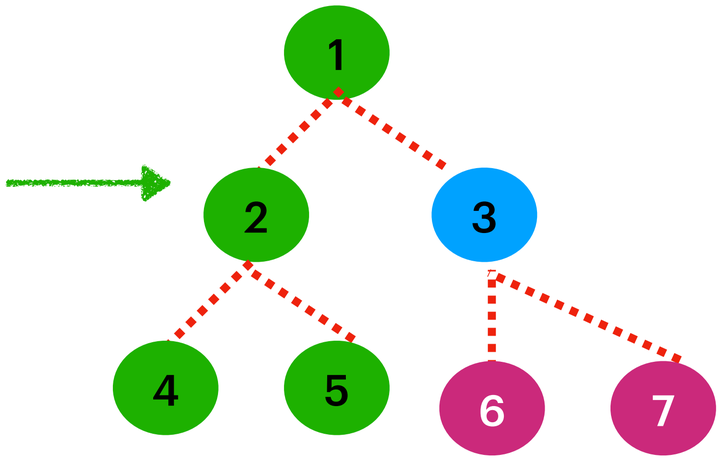
Follow up: Recursive solution is trivial, could you do it iteratively?

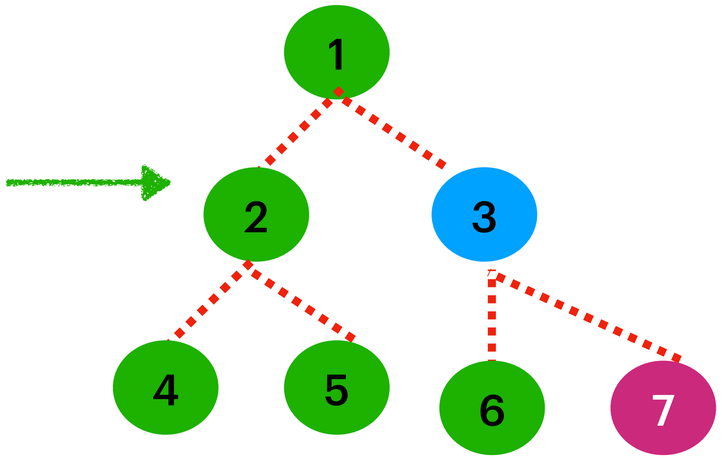
InOrder Traversal Left -> Root -> Right [Recursively]

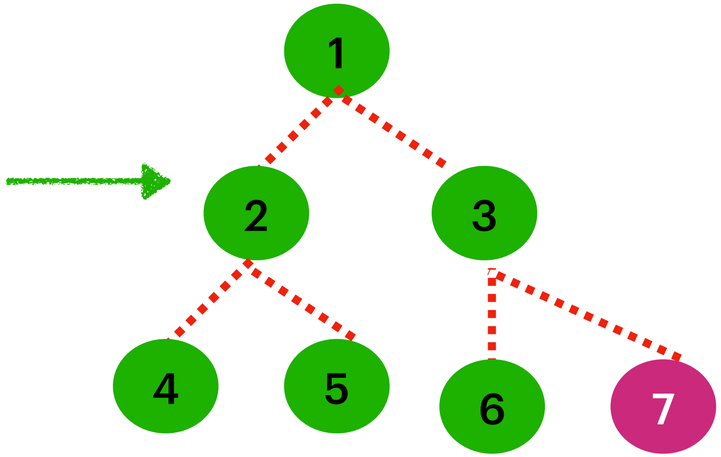


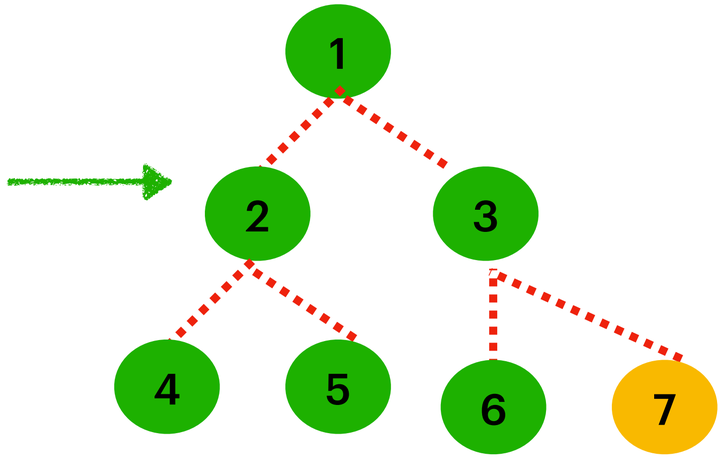
InOrder Traversal Left -> Root -> Right [Recursively]





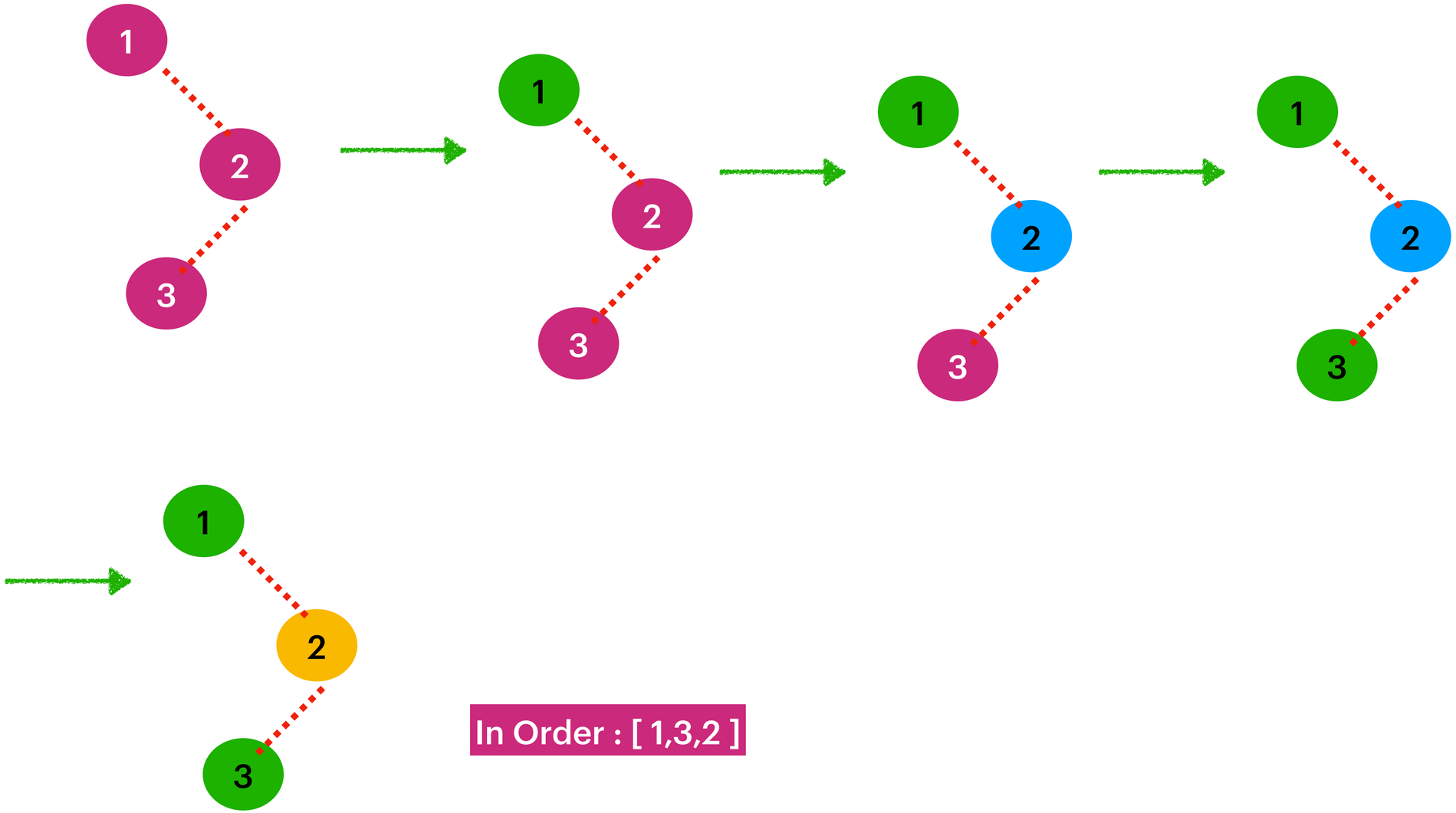






Point out all the green Nodes InOrder : [4, 2, 5, 1 , 6, 3, 7]

InOrder Traversal Left -> Root -> Right [Recursively]



In Order : [1,3,2]

inOrder recursive :

```
ublic void inOrder(TreeNode root)
{
if(root == null)
return;

inOrder(root.left);
print(root.val);
inOrder(root.right)

}
```

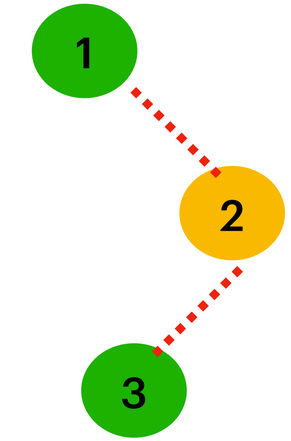
Time Complexity : O(N)
Space Complexity : O(logn)
In Worst Case : O(n)

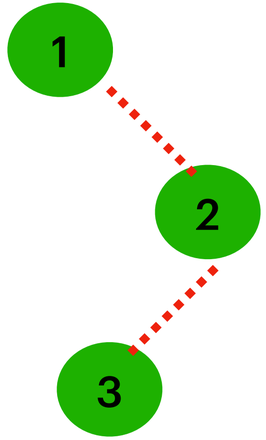
inOrder Iterative : DFS

```
Public void inOrder(TreeNode root)
{

Stack<TreeNode> stack = new Stack<>();
TreeNode current = root;
while(current != null || !stack.isEmpty() )
{
while(current != null)
{
stack.push(current);
current = current.left;
}
current = stack.pop();
print(current.val);
current = current.right;
}

}
```

1 3 2

Time Complexity : O(N)
Space Complexity : O(logn)
In Worst Case : O(n)

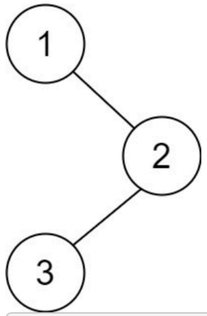




Binary Tree Postorder Traversal

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:

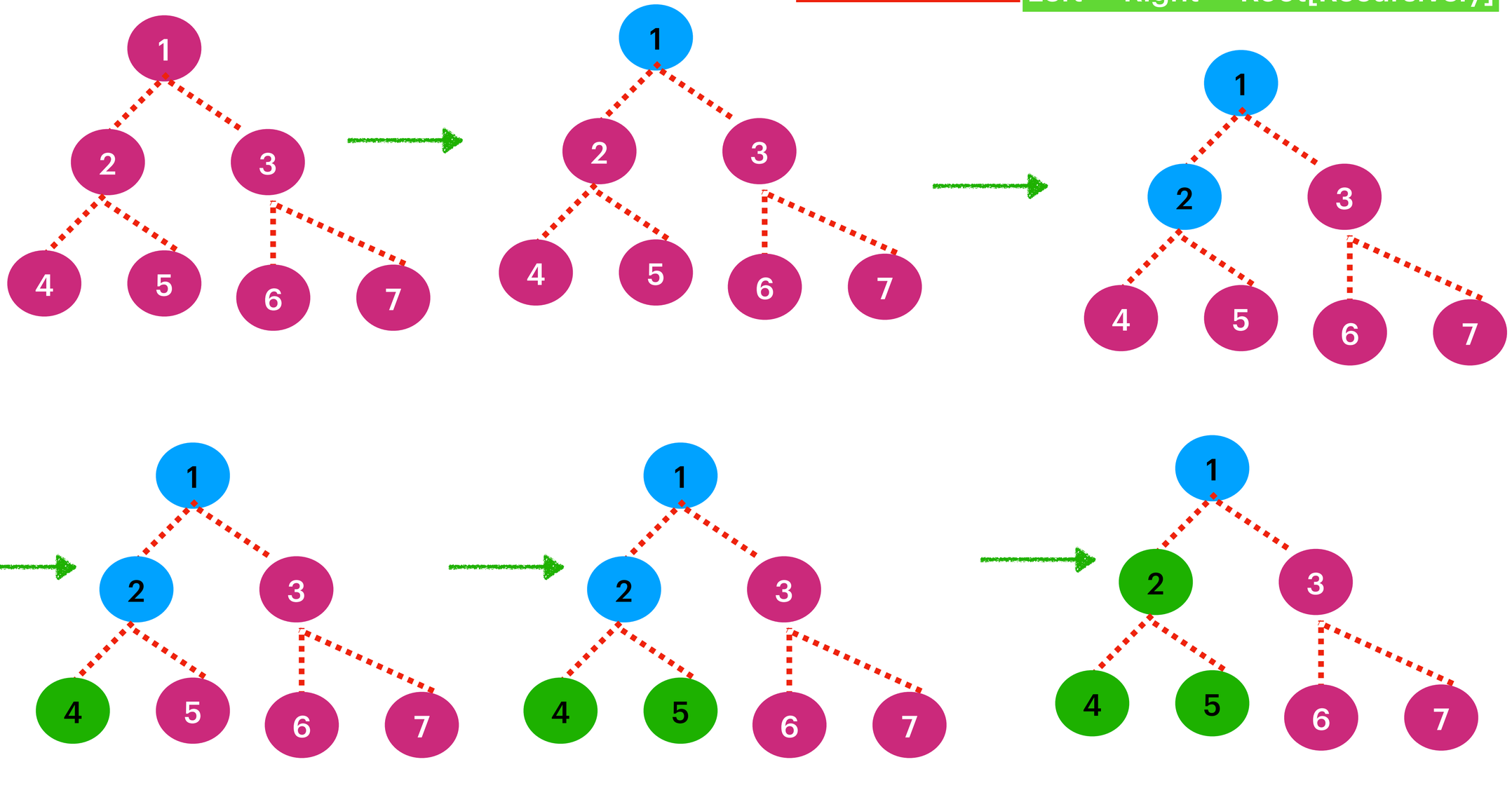
```
Input: root = [1]
Output: [1]
```

Constraints:

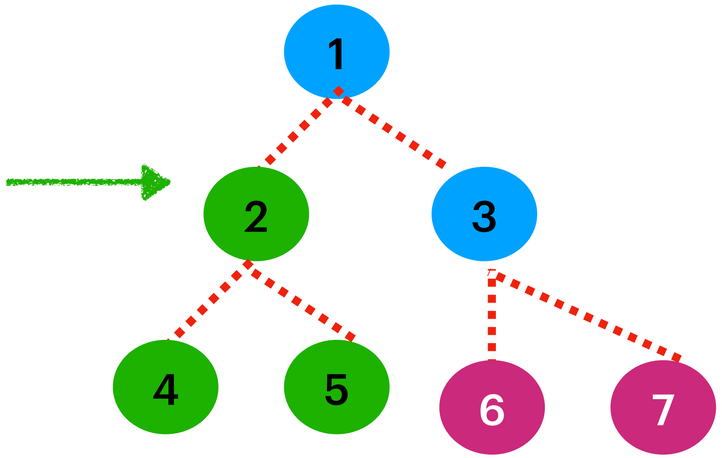
- The number of the nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

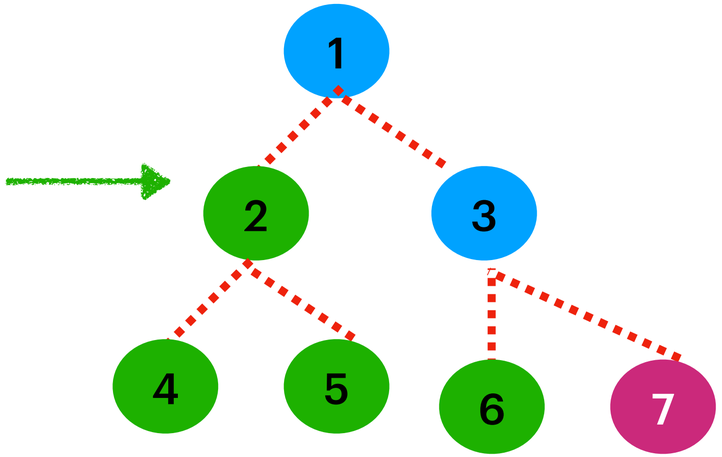
Follow up: Recursive solution is trivial, could you do it iteratively?

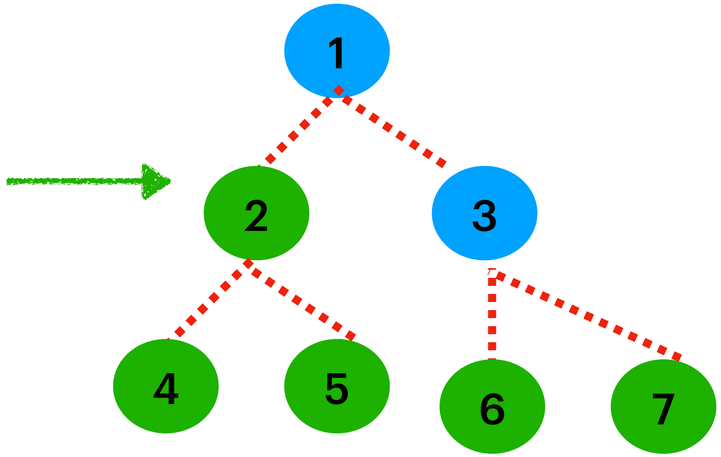
POST Traversal Left -> Right -> Root[Recursively]

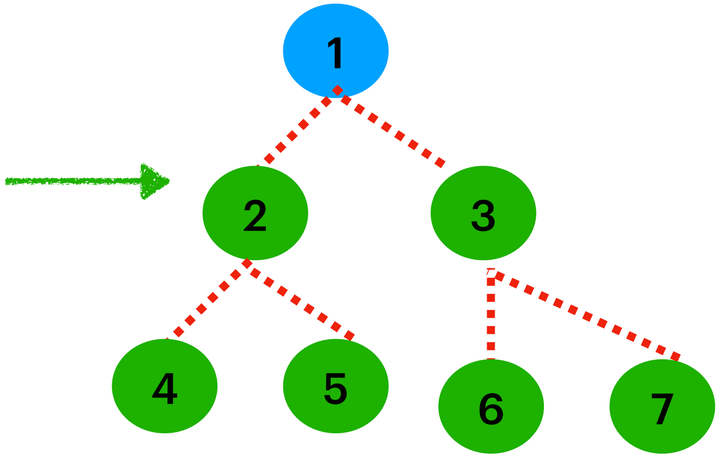


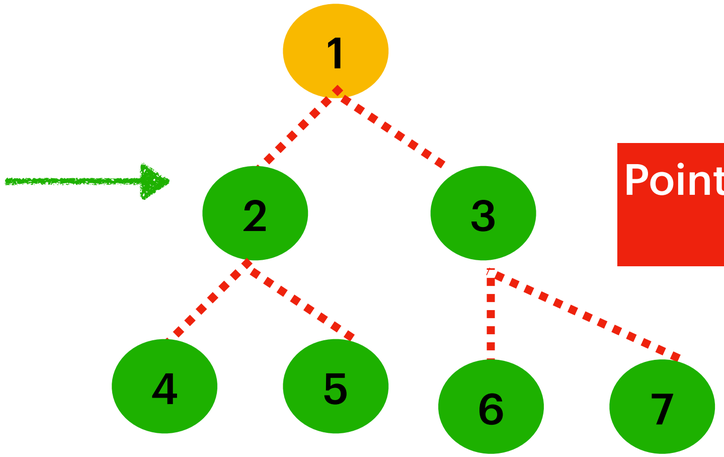
POST Traversal Left -> Right -> Root[Recursively]





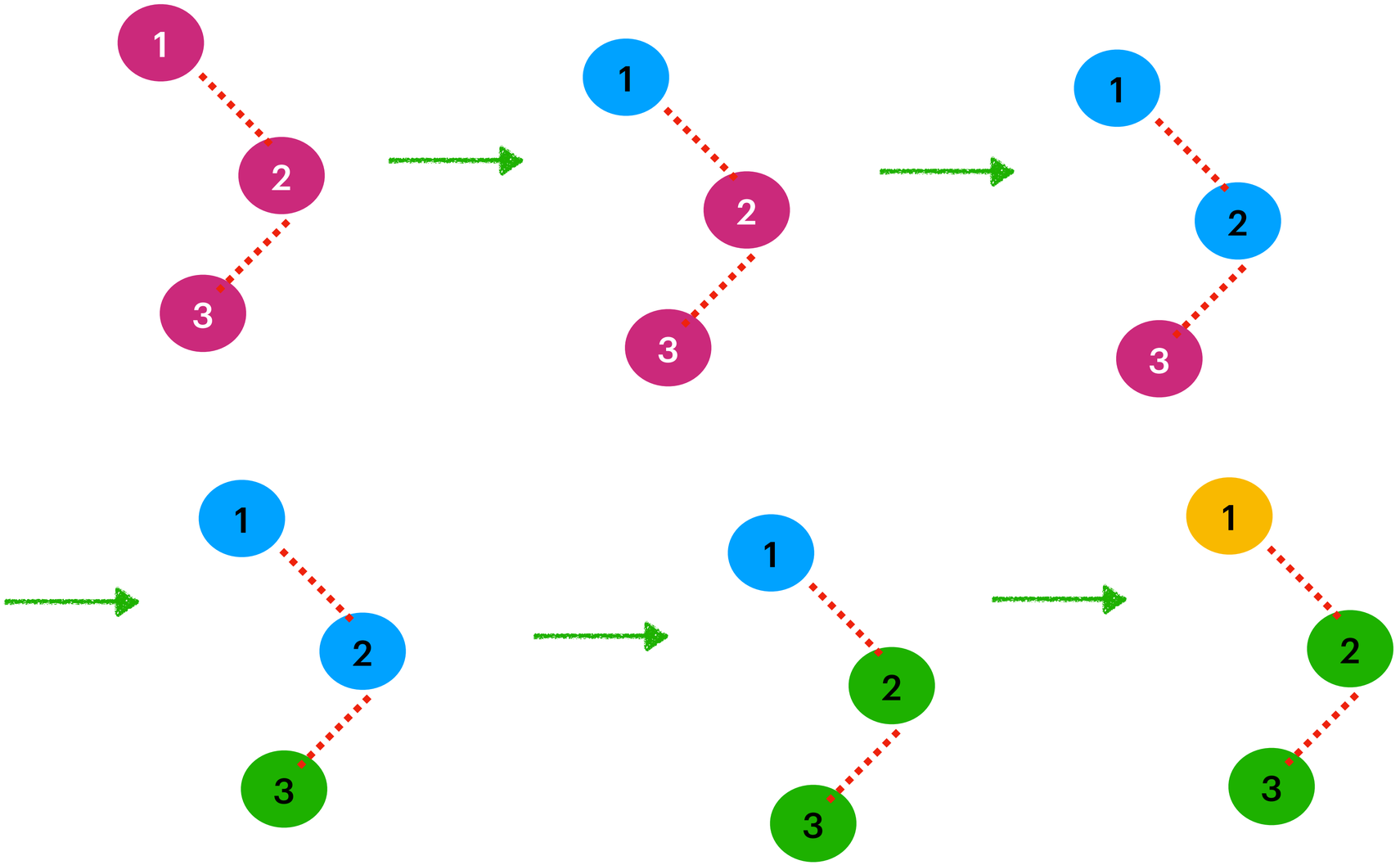






PointOut Order of GreeNodes :
[4,5,2 ,6,7,3,1]

POST Traversal Left -> Right -> Root[Recursively]



POST Traversal : [3, 2, 1]

PostOrder recursive :

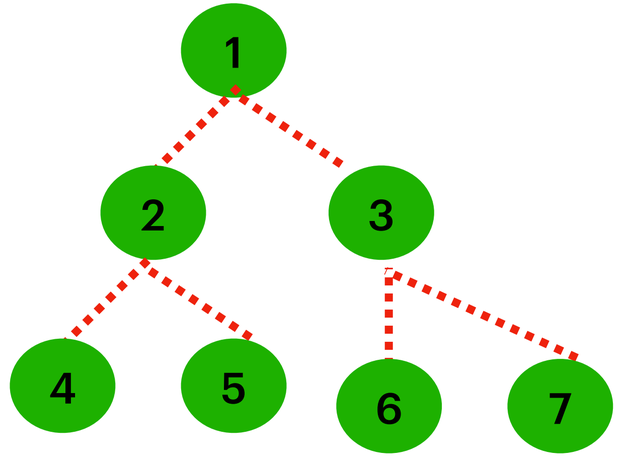
```
Public void PostorderTraversal(TreeNode root)
                {
        if(root == null)
                        return list;

        postorderTraversal(root.left);
        postorderTraversal(root.right);
        print(root.val);
                }
```

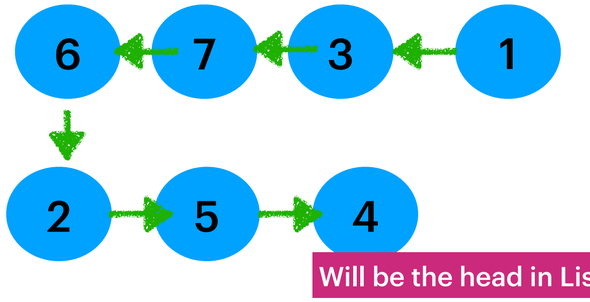
Time Complexity : O(N)
Space Complexity : O(logn)
In Worst Case : O(n)

PostOrder Iterative : DFS

```
LinkedList<Integer> list = new LinkedList<>();

        if(root == null)
                {
                return list;
                }

Stack<TreeNode> stack = new Stack<>();
        stack.push(root);

while( !stack.isEmpty() )
                {
        TreeNode current = stack.pop();

        list.addFirst(current.value);

        if(current.left != null)
                {
        stack.push(current.left);
                }

        if(current.right != null)
                {
        stack.push(current.right);
                }
                }

        return list;
```

Time Complexity : O(N)
Space Complexity : O(logn)
In Worst Case : O(n)





[4,5,2 ,6,7,3,1]