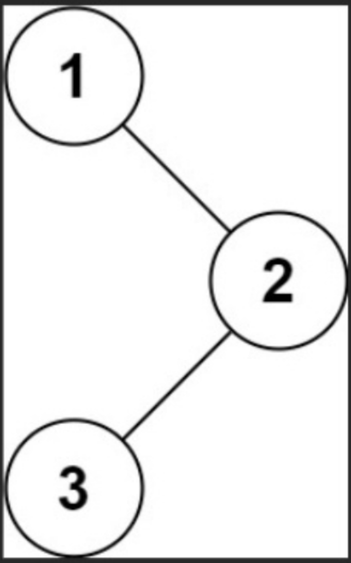
94. Binary Tree Inorder Traversal

Easy 12.3K 643

Companies

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]
```

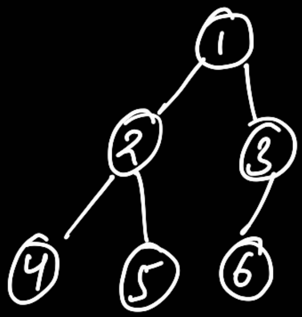
Constraints:

- 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?

Accepted 2.1M | Submissions 2.9M | Acceptance Rate 74.6%

Approach 1: Recursive implementation



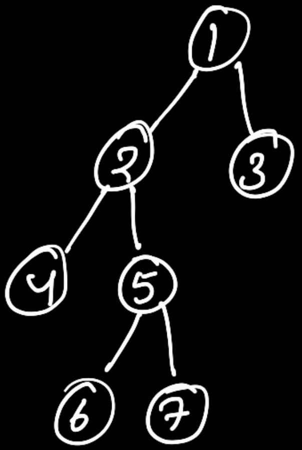
< Left Root Right >

```
inorder(root)
{
    if (root is null) return

    inorder(left)
    print val
    inorder(right)
}
```

Approach 2: Iterative implementation

we can simulate the exact recursion using explicit stack.



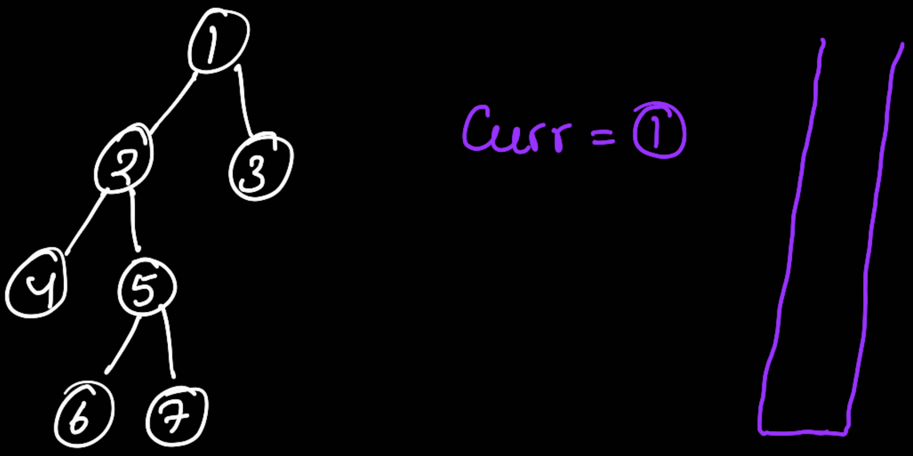
Procedure:

→ Create a stack and take tree node pointer Curr.

→ Assign root to Curr.

Curr = root

→



```
while ( Curr is not null || stack is not empty)
{
    if ( curr is not null)          indicates that we
        push Curr to stack          are going to LST
                                    repeatedly
    else
    {   we entered here which tells that
        we encountered a node i.e. previous
                                        node
      that not has a left child.
        store top of stack in a temp pointer
        add val of top to ans
        pop top of stack

        if ( temp has right node)
               push that right node to stack

        make Curr to point to temp right.
       here if temp has no right node
```

```
            then curr will become null and in
            next iteration we end up in else
            case
        }

        if ( curr is not null)
            then make curr to point to curr → left
                i.e. to go to LST
    }
```

$T(n) : O(n)$
$S(n) : O(n)$