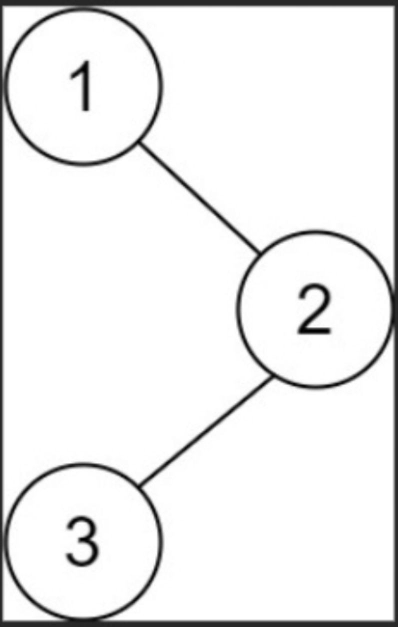
145. Binary Tree Postorder Traversal

Easy 6.4K 181

Companies

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?

Accepted 1M | Submissions 1.5M | Acceptance Rate 69.3%

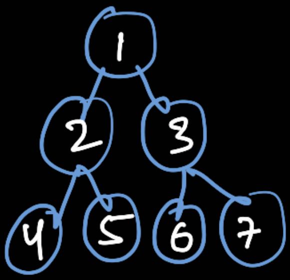
Approach 1: Recursive implementation

$T(n): O(n)$
$S(n)$: Call stack space

Approach 2: Iterative implementation

1. do converse preorder which is kind of simple.
2. Then reverse the ans vector.

note: Converse preorder : Root Right left



Converse Preorder : 1 3 7 6 2 5 4
Postorder : 4 5 2 6 7 3 1

```
→ Curr = root

while ( Curr is not null || Stack is not empty )
{
        if ( Curr is not null )
        {
                add val to ans
                Push Curr to stack
                Curr = Curr → right
        }
        else    means we found a node without
          {   LST so now explore its right.
               temp = top of stack
               pop
               curr = temp → left
          }
     }


     reverse (ans.begin(), ans.end())
```

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

Approach 3: Iterative implementation

```
    s.push(root)

    while (s is not empty)
    {
          auto node = s.top()

          if (node is null)     all children nodes are
          {                      visited.
               s.pop()
             ans.push_back(s.top() → val)
             s.pop()
             continue
          }

          s.push(NULL)   To mark the parent node
    if(node→right)
        s.push(node→right)
    if( node → left)
        s.push(node→ left)

}
```

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