

94. Binary Tree Inorder Traversal

269

10

Add to List ▾

[Description \(/problems/binary-tree-inorder-traversal/description/\)](/problems/binary-tree-inorder-traversal/description/) [Hints \(/problems/binary-tree-inorder-traversal/hints/\)](/problems/binary-tree-inorder-traversal/hints/) [Submissions \(/problems/binary-tree-inorder-traversal/submissions/\)](/problems/binary-tree-inorder-traversal/submissions/)

Solution

Approach #1 Recursive Approach [Accepted]

The first method to solve this problem is using recursion. This is the classical method and is straightforward. We can define a helper function to implement recursion.

Java

Copy

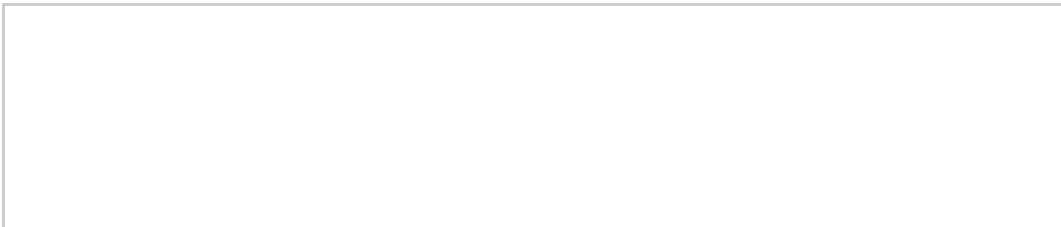
Complexity Analysis

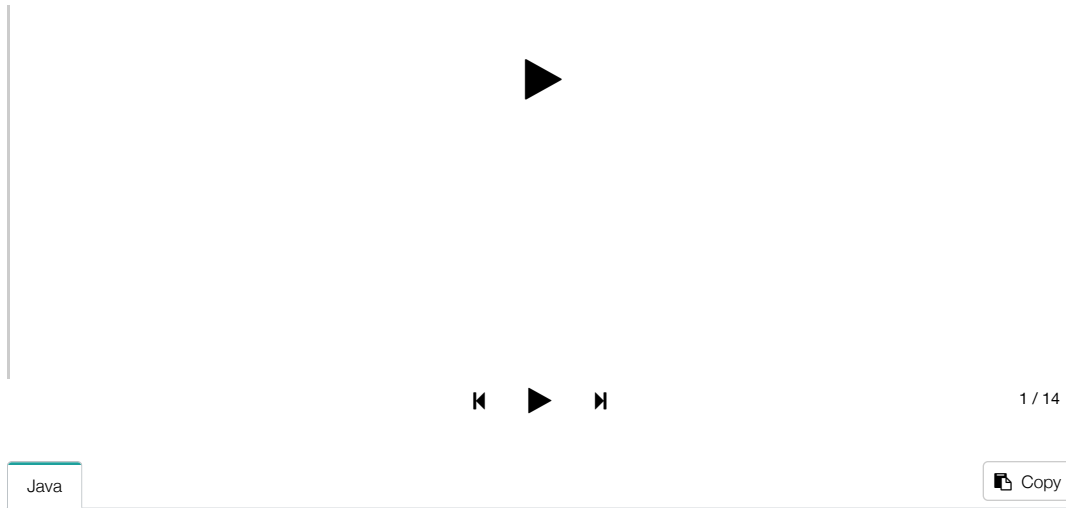
- Time complexity : $O(n)$. The time complexity is $O(n)$ because the recursive function is $T(n) = 2 * T(n/2) + 1$.
- Space complexity : The worst case space required is $O(n)$, and in the average case it's $O(\log(n))$ where n is number of nodes.

Approach #2 Iterating method using Stack [Accepted]

The strategy is very similar to the first method, the different is using stack.

Here is an illustration:





Complexity Analysis

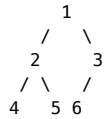
- Time complexity : $O(n)$.
- Space complexity : $O(n)$.

Approach #3 Morris Traversal [Accepted]

In this method, we have to use a new data structure-Threaded Binary Tree, and the strategy is as follows:

```
Step 1: Initialize current as root
Step 2: While current is not NULL,
If current does not have left child
a. Add current's value
b. Go to the right, i.e., current = current.right
Else
a. In current's left subtree, make current the right child of the rightmost node
b. Go to this left child, i.e., current = current.left
```

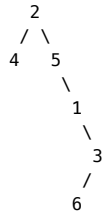
For Example:



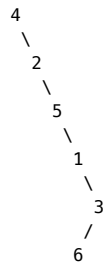
First, 1 is the root, so initialize 1 as current, 1 has left child which is 2, the current's left subtree is



So in this subtree, the rightmost node is 5, then make the current(1) as the right child of 5. Set current = current.left (current = 2). The tree now looks like:



For current 2, which has left child 4, we can continue with the same process as we did above



then add 4 because it has no left child, then add 2, 5, 1, 3 one by one, for node 3 which has left child 6, do the same as above. Finally, the inorder traversal is [4,2,5,1,6,3].

For more details, please check Threaded binary tree (https://en.wikipedia.org/wiki/Threaded_binary_tree) and Explanation of Morris Method (<https://stackoverflow.com/questions/5502916/explain-morris-inorder-tree-traversal-without-using-stacks-or-recursion>)

Java

 Copy

Complexity Analysis

- Time complexity : $O(n)$. To prove that the time complexity is $O(n)$, the biggest problem lies in finding the time complexity of finding the predecessor nodes of all the nodes in the binary tree. Intuitively, the complexity is $O(n \log n)$, because to find the predecessor node for a single node related to the height of the tree. But in fact, finding the predecessor nodes for all nodes only needs $O(n)$ time. Because a binary Tree with n nodes has $n - 1$ edges, the whole processing for each edges up to 2 times, one is to locate a node, and the other is to find the predecessor node. So the complexity is $O(n)$.
- Space complexity : $O(n)$. ArrayList of size n is used.

Analysis written by: @monkeykingyan (<https://leetcode.com/monkeykingyan>)



Join the conversation

Signed in as **Xiaotian_Fu**.

Post a Reply