

[Courses](#)[Practice](#)[Roadmap](#)[Pro](#)

Algorithms and Data Structures for Beginners

20 / 35

19 - Depth-First Search



Mark Lesson Complete









[View Code](#)

[Prev](#)

[Next](#)

18 Remove

Suggested Problems

Status	Star	Problem 	Difficulty 	Video Solution	Code
<input type="checkbox"/>		Binary Tree Inorder Traversal	Easy		
<input type="checkbox"/>		Kth Smallest Element In a Bst	Medium		C++
<input type="checkbox"/>		Construct Binary Tree From Preorder And Inorder Traversal	Medium		C++

Depth-First Search

Depth First Search (DFS) is a way of traversing binary search trees that prioritizes depth rather than breadth.

The idea is to keep traversing down either the left subtree or the right subtree until there are no more nodes left. There are various methods under which depth-first search is performed. These methods visit the nodes - `root`, `left_child`, `right_child` in different orders. These three methods are:

- Inorder
- Preorder
- Postorder

Depth first search is best implemented using recursion. Again, you could use a stack and do it iteratively but it is a lot more complicated.

Take a tree with the nodes `[4,3,6,2,null,5,7]` going from left to right.

Inorder Traversal

An inorder traversal prioritizing left before right will visit the `left-child` first, then the parent node and then the `right-child`. Inorder traversal will result in the nodes being visited in a sorted order.

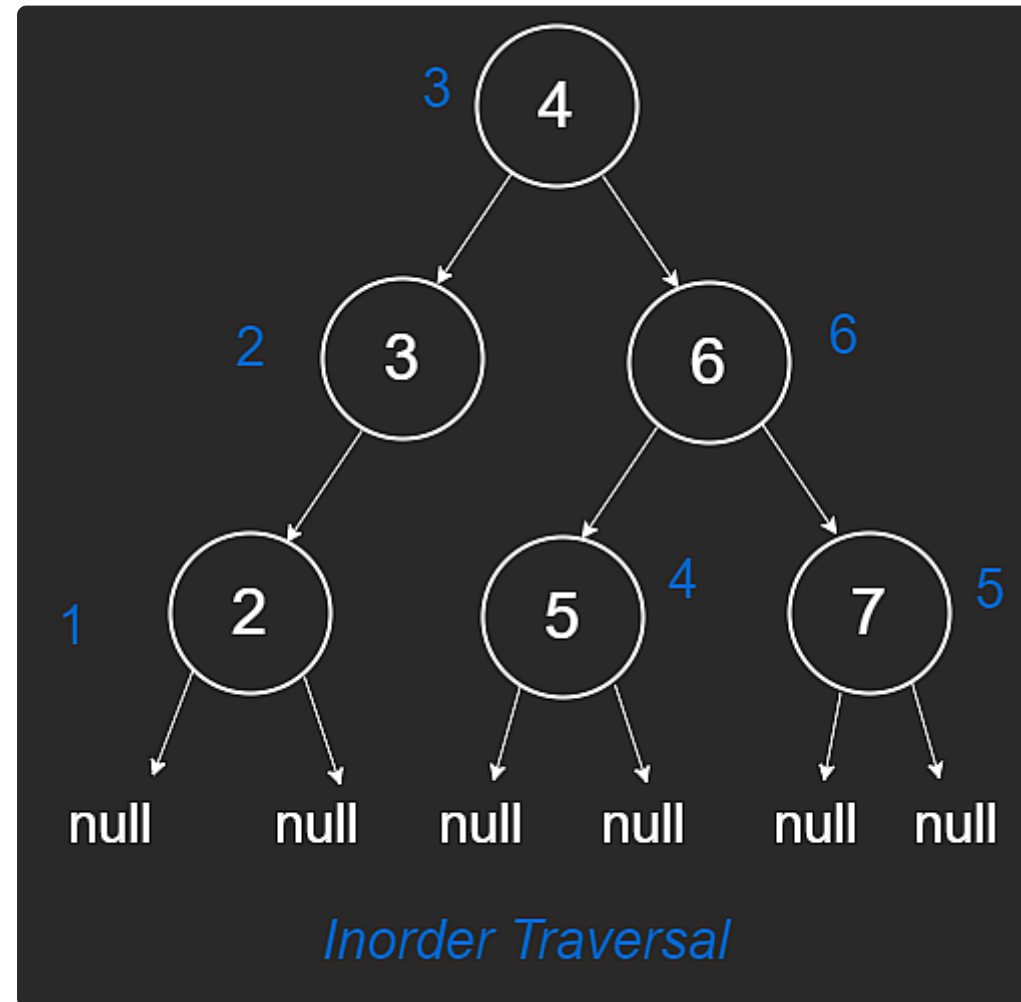
```
fn inorder(root):  
    if root is null:  
        return  
    inorder(root.left)  
    print(root.val)  
    inorder(root.right)
```

The order in which these nodes will be visited is `[2,3,4,5,6,7]`, which is sorted. It is important to note that this only works when left is prioritized before right. If we

prioritize right before left, we will end up with a reverse sorted array.

The reason the nodes will print in a sorted order is because of the BST property. Since we know all values to the left of a node are smaller, this means we won't hit our base case until we reach the left-most node which is also the smallest node. After visiting this, we will traverse up, visit the parent and then visit the right-subtree. The visual below shows this process.

The order in which the nodes are visited is represented by the numbers in blue next to the node.

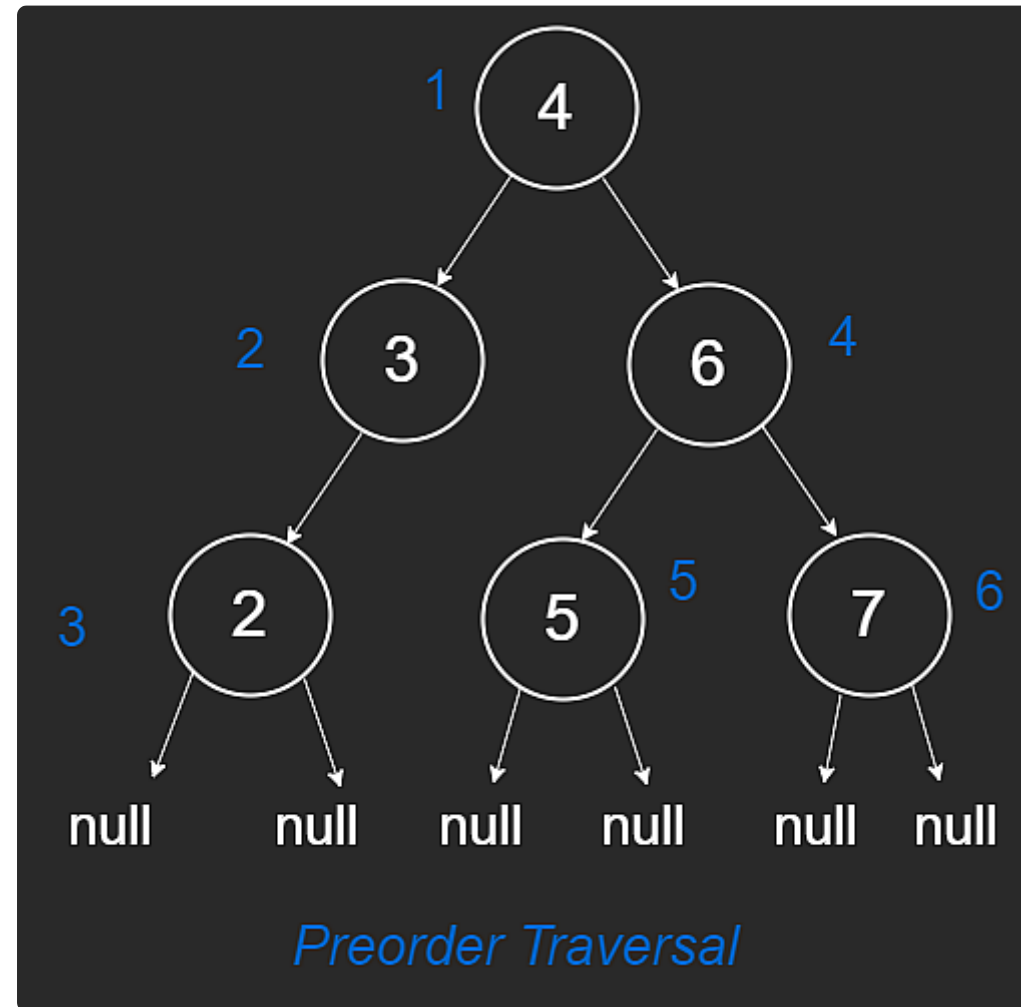


Preorder Traversal

A preorder traversal prioritizing left before right will visit the parent, the left-child and the right-child, in that order.

```
fn preorder(root):  
    if root is null:  
        return  
    print(root.val)  
    preorder(root.left)  
    preorder(root.right)
```

The nodes are visited in the following order **[4,3,2,6,5,7]**

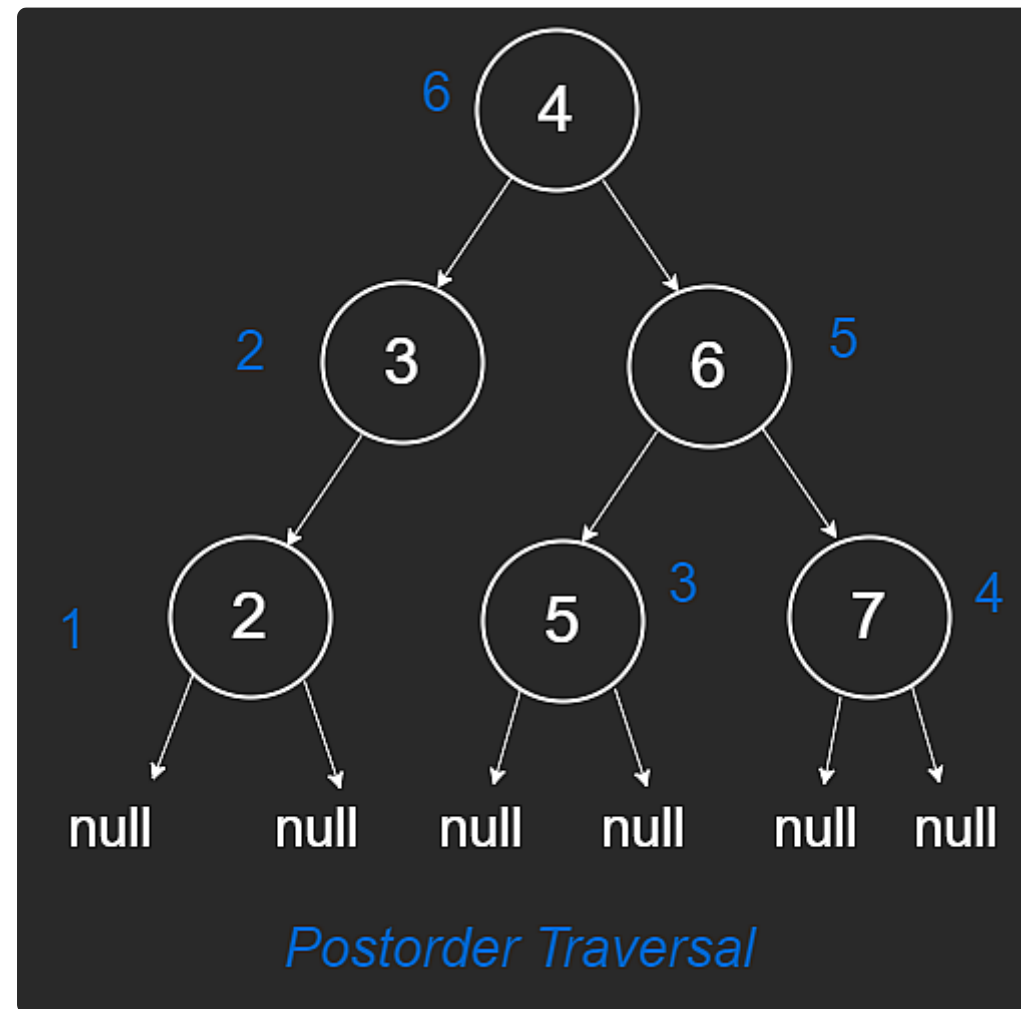


Postorder Traversal

A post-order traversal prioritizing left before right will visit the `left_child`, the `right_child` and the parent, in that order.

```
fn postorder(root):  
    if root is null:  
        return  
    postorder(root.left)  
    postorder(root.right)  
    print(root.val)
```

The order in which these nodes will be visited is: **[2,3,5,7,6]**



Time Space Complexity

We know that we have to visit every node in the tree, and if there are n nodes in the tree, the algorithm will run in $O(n)$.

Another interesting point is that we could actually sort an array by making use of a binary tree. First, we would need to build the tree and insert each value. We know that there are n values and inserting a value in the binary tree takes $\log n$ time. Therefore, the whole process of building the tree will be $O(n \log n)$. Traversing the tree will only take n steps, so in total it will be $O(n + n \log n)$.

We have mentioned before that we do not take constants into consideration. We also know that $O(n \log n)$ will grow faster than $O(n)$, so we can set our upper bound to $O(n \log n)$.

Closing Notes

Now that we have seen how a tree gets traversed depth-first, let's take a look at how it gets traversed when we prioritize breadth first.

Copyright © 2023 NeetCode.io All rights reserved.

Contact: neetcodebusiness@gmail.com

[Github](#) [Privacy](#) [Terms](#)