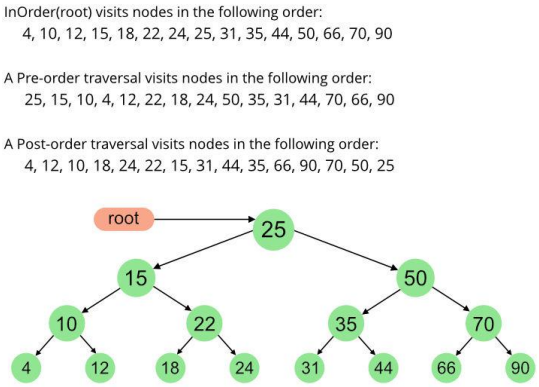
Implementation of Inorder, Preorder and Postorder Traversal

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, trees can be traversed in different ways. The following are the generally used methods for traversing trees:

Example:



Inorder Traversal

```
Algorithm Inorder(tree)
Traverse the left subtree, i.e., call Inorder(left->subtree)
```

```
Visit the root.
Traverse the right subtree, i.e., call Inorder(right->subtree)
```


Uses of Inorder Traversal:

In the case of binary search trees (BST), Inorder traversal gives nodes in non-decreasing order. To get nodes of BST in non-increasing order, a variation of Inorder traversal where Inorder traversal is reversed can be used.
Example: In order traversal for the above-given figure is 4 2 5 1 3.

C++ Java

```
// Java program for different tree traversals

/* Class containing left and right child of current
node and key value*/
class Node {
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree {
    // Root of Binary Tree
    Node root;
```

```
BinaryTree() { root = null; }

/* Given a binary tree, print its nodes in inorder*/
void printInorder(Node node)
{
    if (node == null)
        return;

    /* first recur on left child */
    printInorder(node.left);

    /* then print the data of node */
    System.out.print(node.key + " ");

    /* now recur on right child */
    printInorder(node.right);
}

// Wrappers over above recursive functions
void printInorder() { printInorder(root); }

// Driver code
public static void main(String[] args)
{
    BinaryTree tree = new BinaryTree();
    tree.root = new Node(1);
    tree.root.left = new Node(2);
    tree.root.right = new Node(3);
```

```
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        // Function call
        System.out.println(
            "\nInorder traversal of binary tree is ");
        tree.printInorder();
    }
}
```


Preorder Traversal

> *Algorithm Preorder(tree)*
>
> 1. *Visit the root.*
> 2. *Traverse the left subtree, i.e., call Preorder(left->subtree)*
> 3. *Traverse the right subtree, i.e., call Preorder(right->subtree)*

Uses of Preorder:

Preorder traversal is used to create a copy of the tree. Preorder traversal is also used to get prefix expressions on an expression tree.
Example: Preorder traversal for the above-given figure is 1 2 4 5 3.

C++ Java

```java
// Java program for different tree traversals

/* Class containing left and right child of current
node and key value*/
class Node {
    int key;
    Node left, right;

    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}

class BinaryTree {
    // Root of Binary Tree
    Node root;

    BinaryTree() { root = null; }

    /* Given a binary tree, print its nodes in preorder*/
    void printPreorder(Node node)
    {
        if (node == null)
            return;

        /* first print data of node */
```

```
        System.out.print(node.key + " ");

        /* then recur on left subtree */
        printPreorder(node.left);

        /* now recur on right subtree */
        printPreorder(node.right);
    }

    // Wrappers over above recursive functions
    void printPreorder() { printPreorder(root); }

    // Driver code
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        // Function call
        System.out.println(
            "Preorder traversal of binary tree is ");
        tree.printPreorder();
    }
}
```

Postorder Traversal

Algorithm Postorder(tree)

1. *Traverse the left subtree, i.e., call Postorder(left->subtree)*
2. *Traverse the right subtree, i.e., call Postorder(right->subtree)*
3. *Visit the root*

Uses of Postorder:

Postorder traversal is used to delete the tree. Postorder traversal is also useful to get the postfix expression of an expression tree
Example: Postorder traversal for the above-given figure is 4 5 2 3 1

Below is the implementation of the above traversal methods:

C++ Java

```
// Java program for different tree traversals

/* Class containing left and right child of current
node and key value*/
class Node {
    int key;
    Node left, right;

    public Node(int item)
    {
```

```
            key = item;
            left = right = null;
        }
    }

    class BinaryTree {
        // Root of Binary Tree
        Node root;

        BinaryTree() { root = null; }

        /* Given a binary tree, print its nodes according to the
        "bottom-up" postorder traversal. */
        void printPostorder(Node node)
        {
            if (node == null)
                return;

            // first recur on left subtree
            printPostorder(node.left);

            // then recur on right subtree
            printPostorder(node.right);

            // now deal with the node
            System.out.print(node.key + " ");

        // Wrappers over above recursive functions
```

```
    void printPostorder() { printPostorder(root); }

    // Driver code
    public static void main(String[] args)
    {
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(1);
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);
        tree.root.left.left = new Node(4);
        tree.root.left.right = new Node(5);

        // Function call
        System.out.println(
            "\nPostorder traversal of binary tree is ");
        tree.printPostorder();
    }
}
```

Dash
All
Articles
Videos
Problems
Quiz

Output

```
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
```

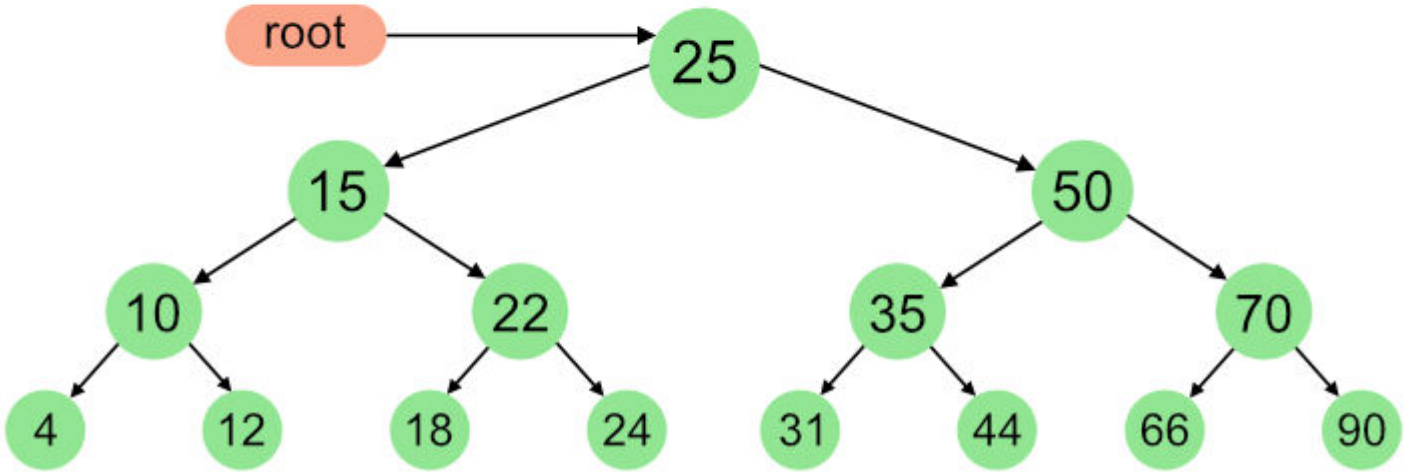
Time Complexity: O(N)
Auxiliary Space: If we don't consider the size of the stack for function calls then O(1) otherwise O(h) where h is the height of the tree.

Note: The height of the skewed tree is n (no. of elements) so the worst space complexity is O(N) and the height is (Log N) for the balanced tree so the best space
complexity is O(Log N).

InOrder(root) visits nodes in the following order:
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:
25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90

A Post-order traversal visits nodes in the following order:
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Mark as Read

Report An Issue

If you are facing any issue on this page. Please let us know.