جامعة المستقبل
AL MUSTAQBAL UNIVERSITY

قسم المن السيبراني

**DEPARTMENT OF CYBER SECURITY**

**SUBJECT:**

**SEARCHING AND SORTING ALGORITHMS**

**CLASS:**

**SECOND**

**LECTURER:**

**ASST. PROF. DR. ALI KADHUM AL-QURABY**

# LECTURE: (5)

# TRAVERSAL BINARY TREE

- **In-order traversal is defined as a type of tree traversal technique which follows the Left-Root-Right pattern. Below is the code implementation of the inorder traversal:**

```cpp
// C++ program for inorder traversals

#include <bits/stdc++.h>
using namespace std;

// Structure of a Binary Tree Node
struct Node {
    int data;
    struct Node *left, *right;
    Node(int v)
    {
        data = v;
        left = right = nullptr;
    }
};

// Function to print inorder traversal
void printInorder(struct Node* node)
{
    if (node == nullptr)
        return;

    // First recur on left subtree
    printInorder(node->left);

    // Now deal with the node
    cout << node->data << " ";

    // Then recur on right subtree
    printInorder(node->right);
}

// Driver code
int main()
{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
```

```
    root->left->right = new Node(5);
    root->right->right = new Node(6);

    // Function call
    cout << "Inorder traversal of binary tree is: \n";
    printInorder(root);

    return 0;
}
```

### Output

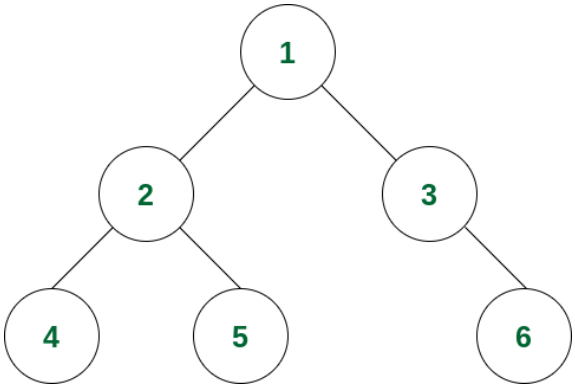
```
Inorder traversal of binary tree is:
4 2 5 1 3 6
```

- **Time Complexity:** O(N) where N is the total number of nodes. Because it traverses all the nodes at least once.

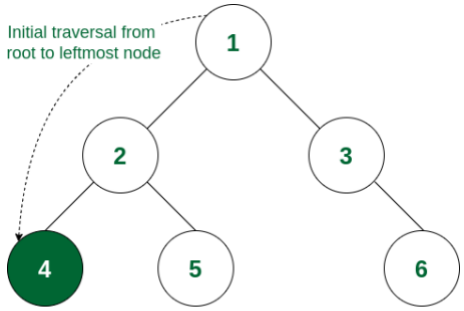
- **Auxiliary Space:** O(h) where h is the height of the tree. This space is required for recursion calls.
  - In the worst case, **h** can be the same as **N** (when the tree is a skewed tree)
  - In the best case, **h** can be the same as **log N** (when the tree is a complete tree)

## ❖ How does Inorder Traversal of Binary Tree work?

- Let us understand the algorithm with the below example tree

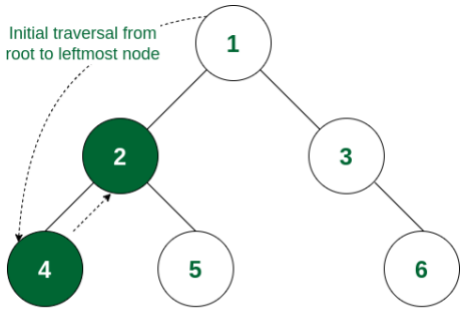


- ***Step 1:*** *The traversal will go from 1 to its left subtree i.e., 2, then from 2 to its left subtree root, i.e., 4. Now 4 has no left subtree, so it will be visited. It also does not have any right subtree. So, no more traversal from 4*



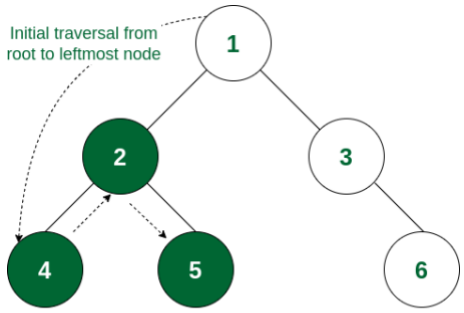

**Leftmost node of the tree is visited**

- ***Step 2:*** *As the left subtree of 2 is visited completely, now it read data of node 2 before moving to its right subtree.*



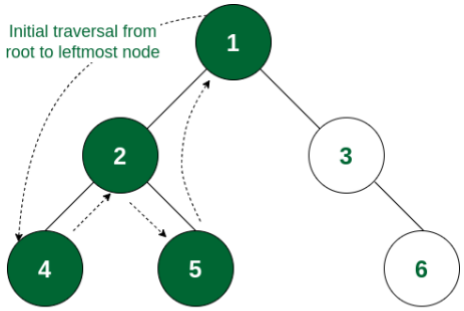

**Left subtree of 2 is fully traversed. So 2 is visited next**

- ***Step 3:*** *Now the right subtree of 2 will be traversed i.e., move to node 5. For node 5 there is no left subtree, so it gets visited and after that, the traversal comes back because there is no right subtree of node 5.*



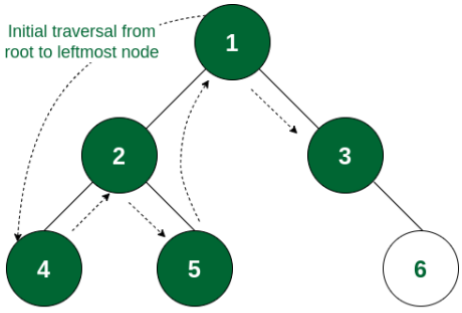

Right subtree of 2 (i.e., 5) is traversed

- ✓ ***Step 4:*** *As the left subtree of node 1 is, the root itself, i.e., node 1 will be visited.*



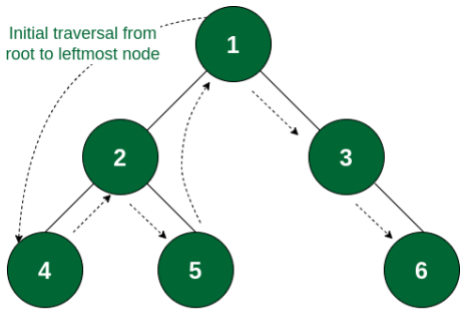

Left subtree of 1 is fully traversed. So 1 is visited next

- ✓ ***Step 5:*** *Left subtree of node 1 and the node itself is visited. So now the right subtree of 1 will be traversed i.e., move to node 3. As node 3 has no left subtree so it gets visited.*




3 has no left subtree, so it is visited

✓ ***Step 6:*** *The left subtree of node 3 and the node itself is visited. So traverse to the right subtree and visit node 6. Now the traversal ends as all the nodes are traversed.*




Right Child of 3 is visited

✓ *So, the order of traversal of nodes is **4 -> 2 -> 5 -> 1 -> 3 -> 6**.*

## Program to Implement Postorder Traversal of Binary Tree

Below is the code implementation of the postorder traversal:

```
// C++ program for postorder traversals

#include <bits/stdc++.h>
using namespace std;

// Structure of a Binary Tree Node
struct Node {
    int data;
    struct Node *left, *right;
    Node(int v)
    {
        data = v;
        left = right = nullptr;
    }
};

// Function to print postorder traversal
```

```
void printPostorder(struct Node* node)
{
    if (node == nullptr)
        return;

    // First recur on left subtree
    printPostorder(node->left);

    // Then recur on right subtree
    printPostorder(node->right);

    // Now deal with the node
    cout << node->data << " ";
}

// Driver code
int main()
{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);

    // Function call
    cout << "Postorder traversal of binary tree is: \n";
    printPostorder(root);

    return 0;
}
```

## Output

```
Postorder traversal of binary tree is:
4 5 2 6 3 1
```

- ## Complexity Analysis:

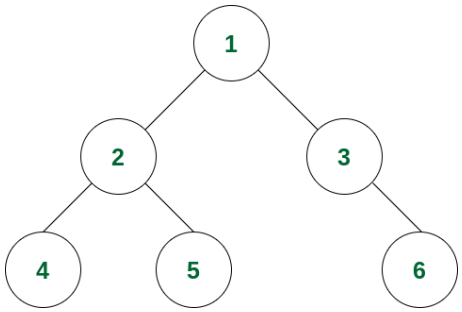
❖ **Time Complexity:** O(N) where N is the total number of nodes. Because it traverses all the nodes at least once.

❖ **Auxiliary Space:** O(1) if no recursion stack space is considered. Otherwise, O(h) where h is the height of the tree

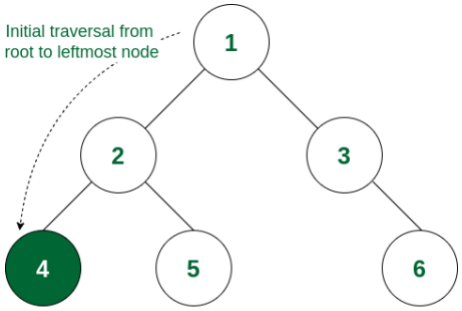
- In the worst case, **h** can be the same as **N** (when the tree is a skewed tree)
- In the best case, **h** can be the same as **logN** (when the tree is a complete tree)

## How does Postorder Traversal of Binary Tree Work?

Consider the following tree:

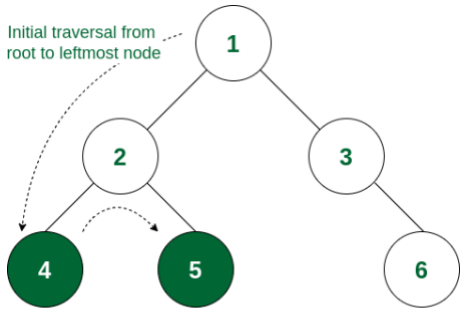



✓ ***Step 1:*** *The traversal will go from 1 to its left subtree i.e., 2, then from 2 to its left subtree root, i.e., 4. Now 4 has no subtree, so it will be visited.*



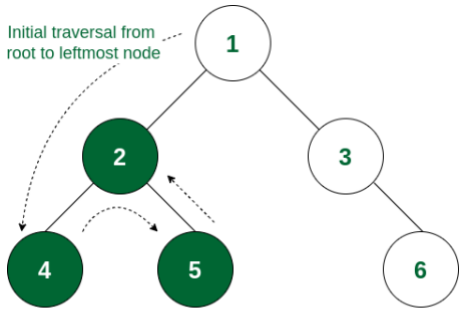

The leftmost leaf node (i.e., 4) is visited first

✓ ***Step 2:*** *As the left subtree of 2 is visited completely, now it will traverse the right subtree of 2 i.e., it will move to 5. As there is no subtree of 5, it will be visited.*



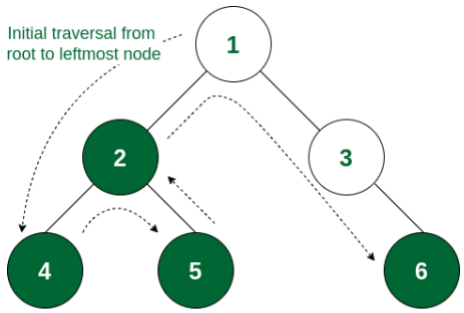

Left subtree of 2 is traversed. So 5 is visited next

- ✓ ***Step 3:*** *Now both the left and right subtrees of node 2 are visited. So now visit node 2 itself.*



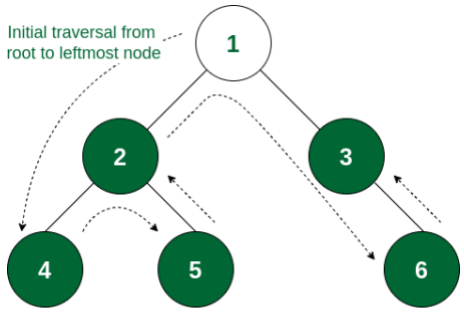

All subtrees of 2 are visited. So 2 is visited next

- ✓ ***Step 4:*** *As the left subtree of node 1 is traversed, it will now move to the right subtree root, i.e., 3. Node 3 does not have any left subtree, so it will traverse the right subtree i.e., 6. Node 6 has no subtree and so it is visited.*



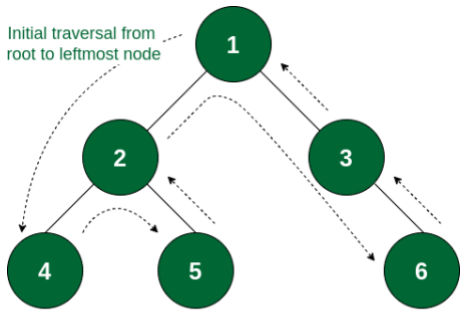

6 has no subtrees. So it is visited

- ***Step 5:*** *All the subtrees of node 3 are traversed. So now node 3 is visited.*




**3 is visited after all its subtrees are traversed**

- ***Step 6:*** *As all the subtrees of node 1 are traversed, now it is time for node 1 to be visited and the traversal ends after that as the whole tree is traversed.*




**The root of the tree (i.e., 1) is visited**

- *So the order of traversal of nodes is **4 -> 5 -> 2 -> 6 -> 3 -> 1**.*

# Program to Implement Preorder Traversal of Binary Tree

Below is the code implementation of the preorder traversal:

```cpp
// C++ program for preorder traversals

#include <bits/stdc++.h>
using namespace std;

// Structure of a Binary Tree Node
struct Node {
    int data;
    struct Node *left, *right;
    Node(int v)
    {
        data = v;
        left = right = nullptr;
    }
};

// Function to print preorder traversal
void printPreorder(struct Node* node)
{
    if (node == nullptr)
        return;

    // Deal with the node
    cout << node->data << " ";

    // Recur on left subtree
    printPreorder(node->left);

    // Recur on right subtree
    printPreorder(node->right);
}

// Driver code
int main()
{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(6);
```

```
    // Function call
    cout << "Preorder traversal of binary tree is: \n";
    printPreorder(root);

    return 0;
}
```

**Output**

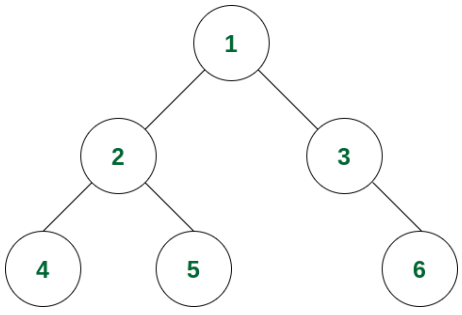
```
Preorder traversal of binary tree is:
1 2 4 5 3 6
```

- ## Complexity Analysis:

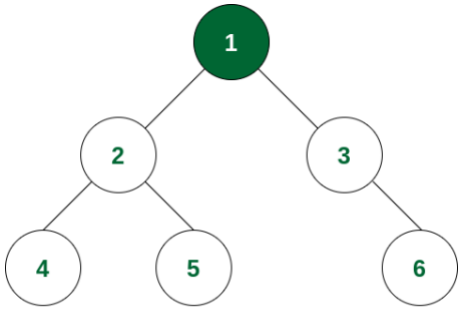
- **Time Complexity**: O(N) where N is the total number of nodes. Because it traverses all the nodes at least once.
- Auxiliary Space:
  - **O(1)** if no recursion stack space is considered.
  - Otherwise, **O(h)** where h is the height of the tree
  - In the worst case, **h** can be the same as **N** (when the tree is a skewed tree)
  - In the best case, **h** can be the same as **logN** (when the tree is a complete tree)

## ❖ How does Preorder Traversal of Binary Tree work?

Consider the following tree:

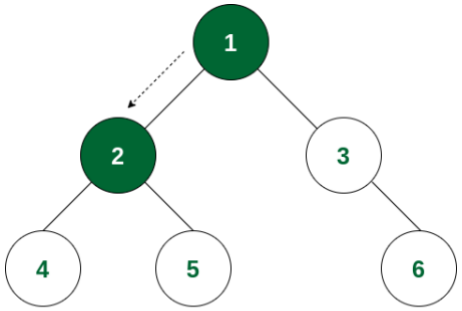



✓ **Step 1:** At first the root will be visited, i.e. node 1.



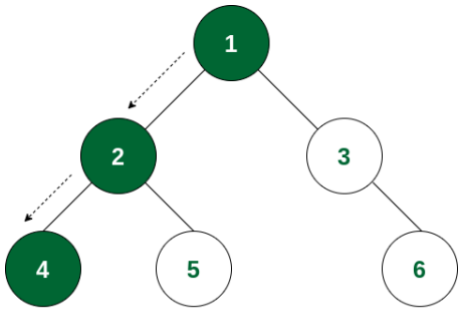

Root of the tree (i.e., 1) is visted

- ✓ **Step 2:** After this, traverse in the left subtree. Now the root of the left subtree is visited i.e., node 2 is visited.



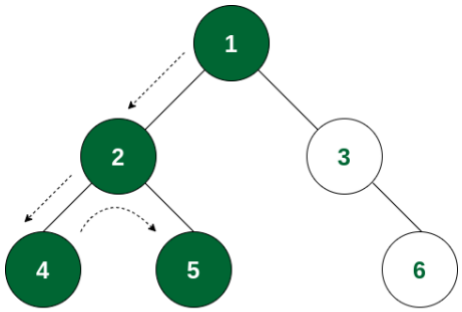

Root of left subtree of 1 (i.e., 2) is visited

- ✓ **Step 3:** Again the left subtree of node 2 is traversed and the root of that subtree i.e., node 4 is visited.



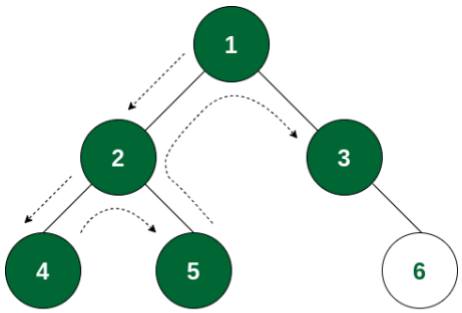

Left child of 2 (i.e., 4) is visited

✓ **Step 4:** There is no subtree of 4 and the left subtree of node 2 is visited. So now the right subtree of node 2 will be traversed and the root of that subtree i.e., node 5 will be visited.



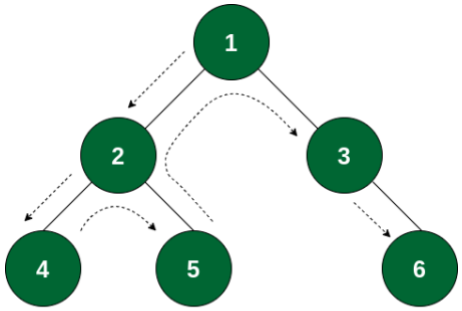

Right child of 2 (i.e., 5) is visited

✓ **Step 5:** The left subtree of node 1 is visited. So now the right subtree of node 1 will be traversed and the root node i.e., node 3 is visited.




Root of right subtree of 1 (i.e., 3) is visited

✓ **Step 6:** Node 3 has no left subtree. So the right subtree will be traversed and the root of the subtree i.e., node 6 will be visited. After that there is no node that is not yet traversed. So the traversal ends.




3 has no left subtree. So right subtree is visited

✓ So the order of traversal of nodes is **1 -> 2 -> 4 -> 5 -> 3 -> 6**.