
جامـــــعة المـــــستـقبـل
AL MUSTAQBAL UNIVERSITY


قســـم الامـــــن الـــــــــسيبرانـــــــي

**Department of Cyber Security**

**Subject:**

## Searching And Sorting Algorithms

**Class:**

**Second**

**Lecturer:**

**Asst. Prof. Dr. Ali Kadhum Al-Quraby**

# Lecture: (5-1)

# In-order traversal

- **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

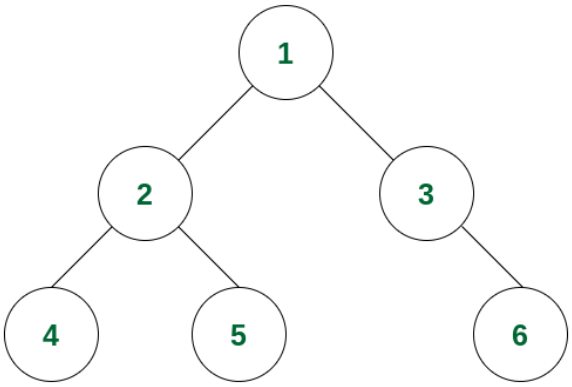
```
Inorder traversal of binary tree is:
4 2 5 1 3 6
```

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

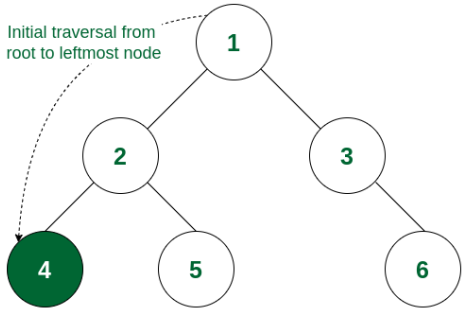
- **Auxiliary Space:** O(h) where h is the height of the tree. This space is required for recursion calls.
  - In the worst case, **h** can be the same as **N** (when the tree is a skewed tree)
  - In the best case, **h** can be the same as **log N** (when the tree is a complete tree)

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

- Let us understand the algorithm with the below example tree

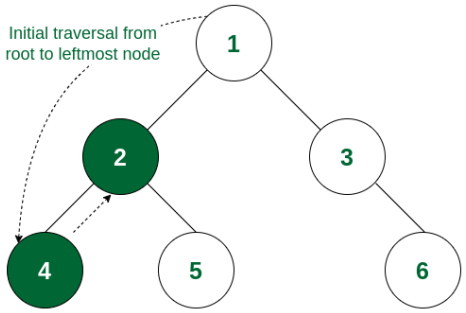


- ***Step 1:*** *The traversal will go from 1 to its left subtree i.e., 2, then from 2 to its left subtree root, i.e., 4. Now 4 has no left subtree, so it will be visited. It also does not have any right subtree. So, no more traversal from 4*



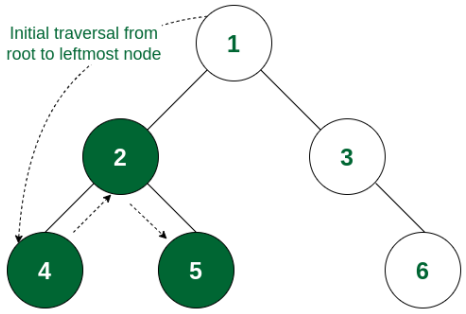

**Leftmost node of the tree is visited**

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



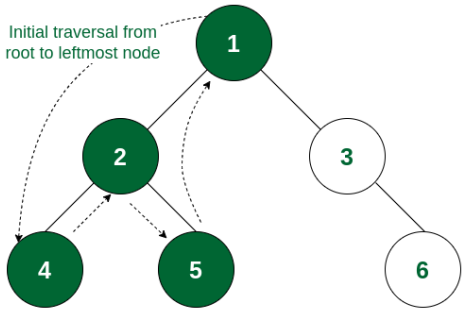

**Left subtree of 2 is fully traversed. So 2 is visited next**

- ***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.*



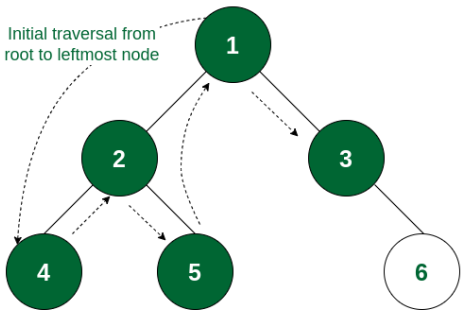

**Right subtree of 2 (i.e., 5) is traversed**

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



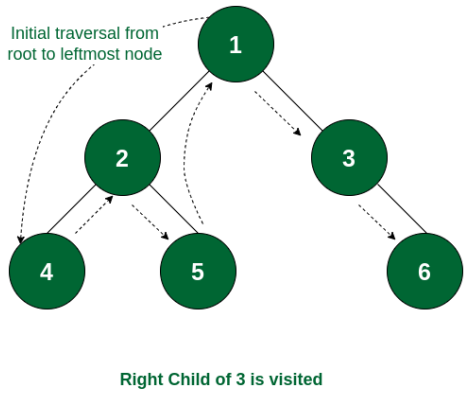

**Left subtree of 1 is fully traversed. So 1 is visited next**

✓ ***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.*




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