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: (5-3)

PRE-ORDER TRAVERSAL

Program to Implement Preorder Traversal of Binary Tree

Below is the code implementation of the preorder traversal:

```
// 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

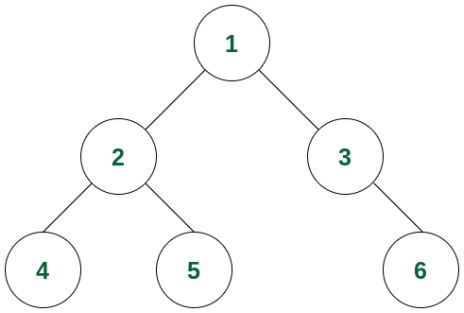
```
Preorder traversal of binary tree is:
1 2 4 5 3 6
```

• Complexity Analysis:

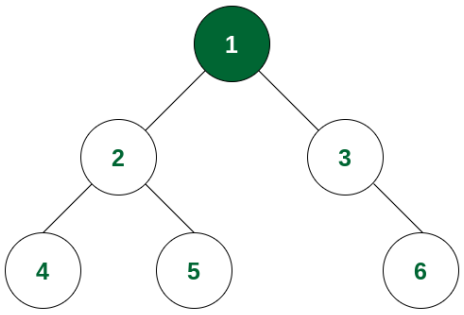
- **Time Complexity**: O(N) where N is the total number of nodes. Because it traverses all the nodes at least once.
- Auxiliary Space:
  - **O(1)** if no recursion stack space is considered.
  - Otherwise, **O(h)** where h is the height of the tree
  - In the worst case, **h** can be the same as **N**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

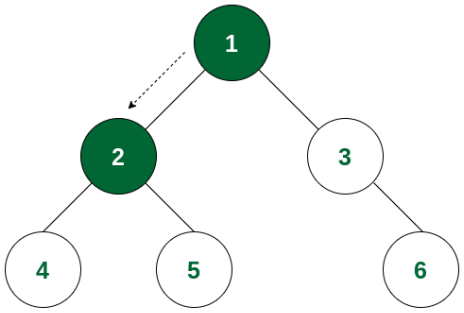


✓ **Step 1:** At first the root will be visited, i.e. node 1.



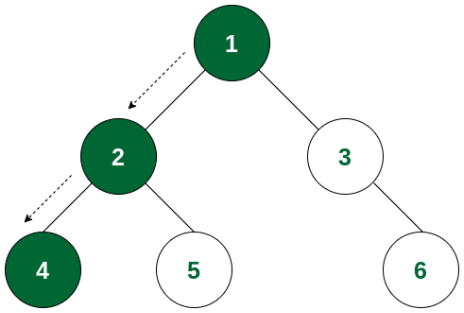
Root of the tree (i.e., 1) is visted

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



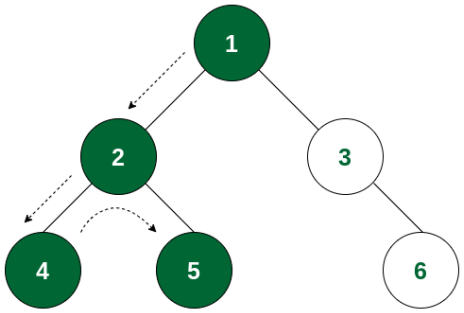
Root of left subtree of 1 (i.e., 2) is visited

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



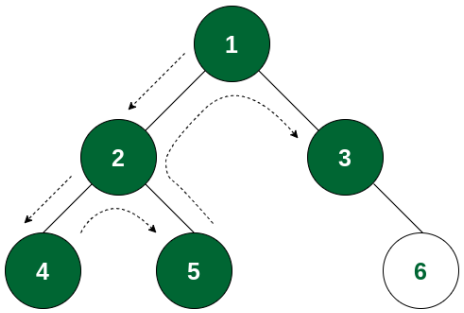
Left child of 2 (i.e., 4) is visited

- ✓ **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.



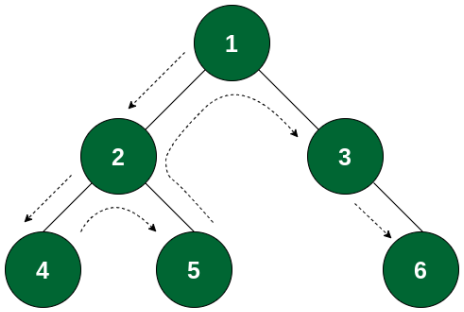
Right child of 2 (i.e., 5) is visited

- ✓ **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**.