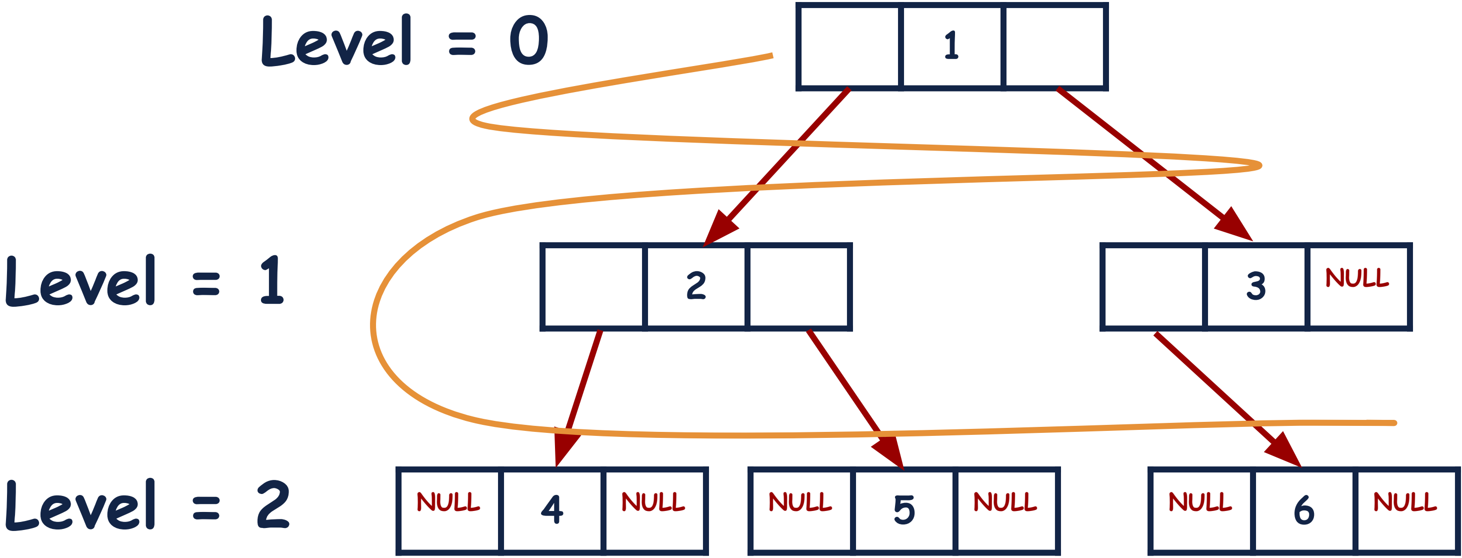
Tree Traversal

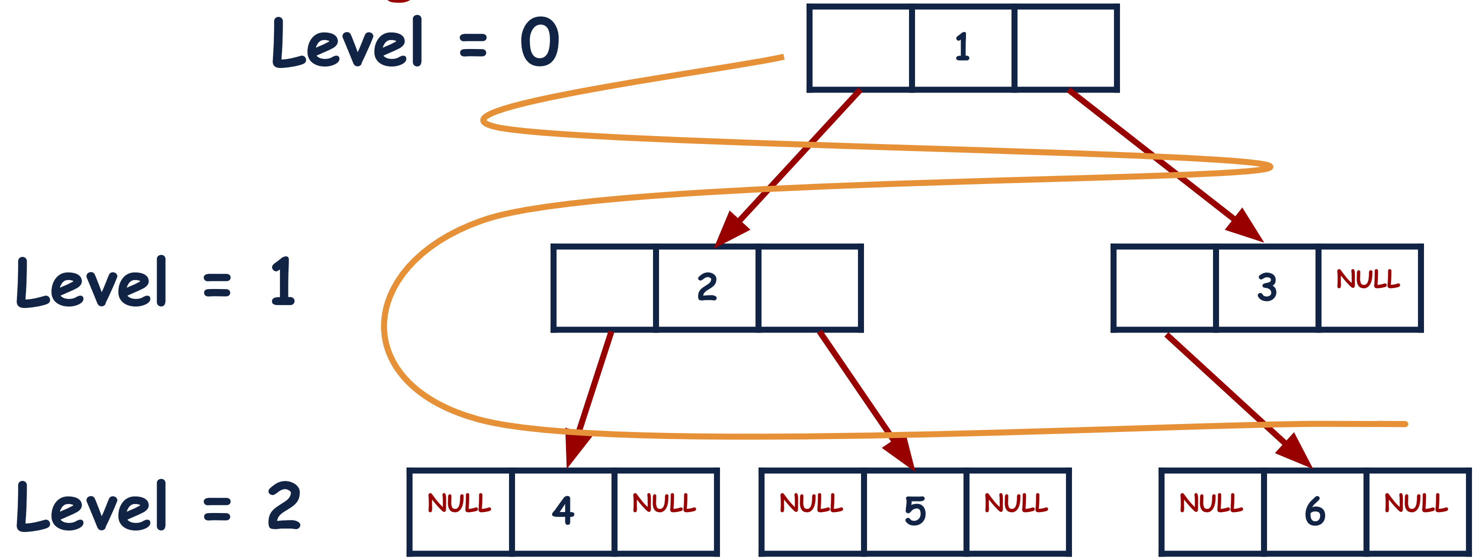
Traversal (Breadth First): Review

Previously we implemented Breadth First Traversal or Level Order Traversal.



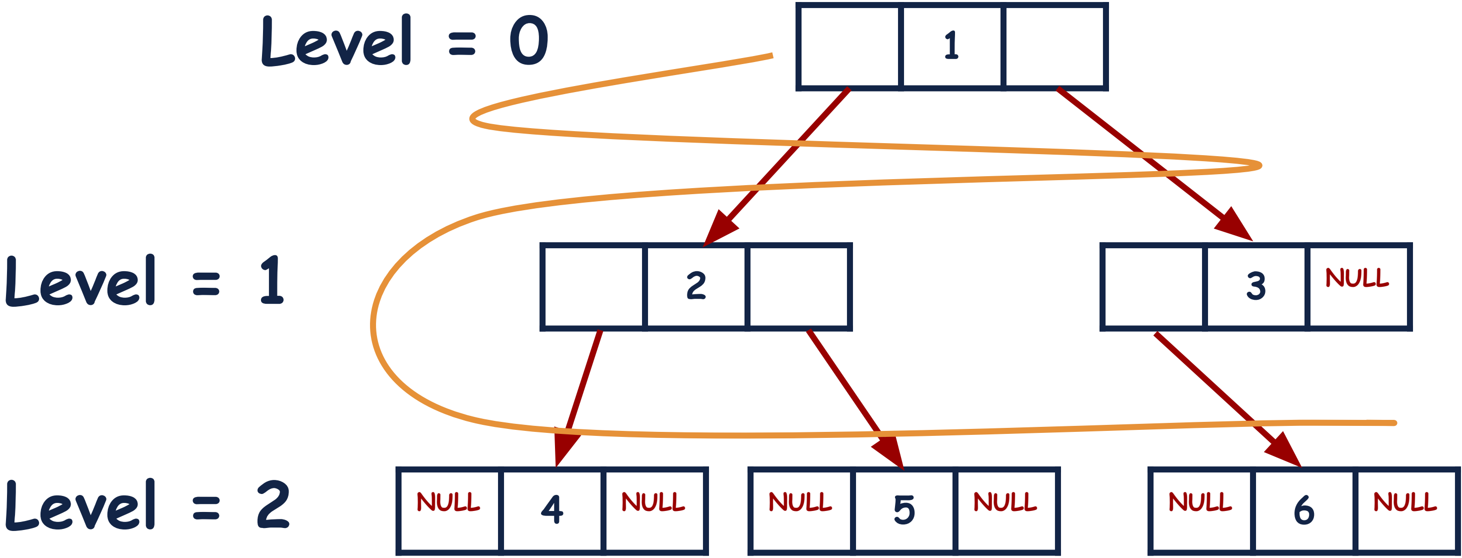
Traversal (Breadth First): Review

In this, We printed all the nodes of depth 0, then depth 1 from left to right, and so on.



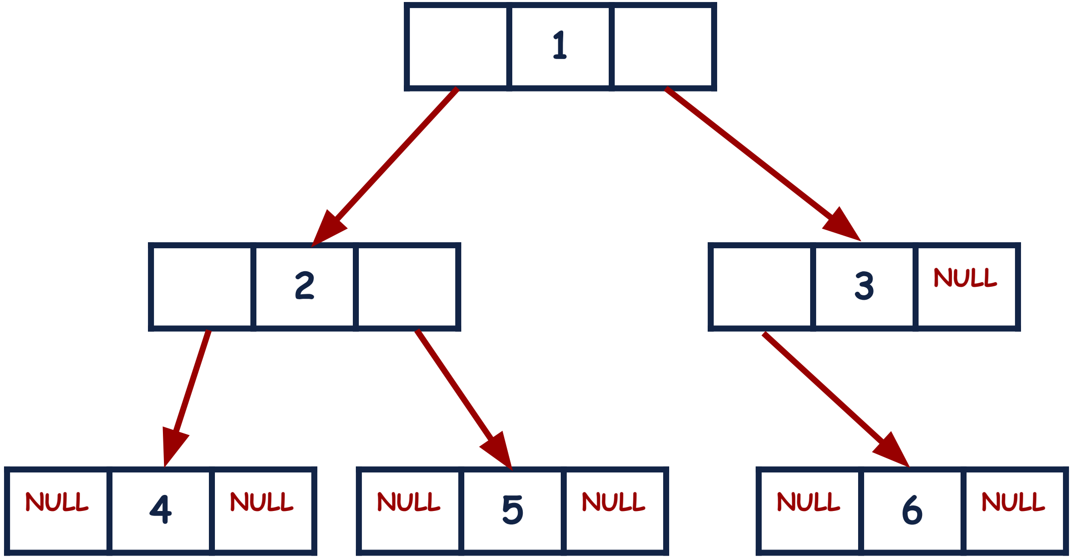
Traversal (Breadth First): Review

For the Breadth first traversal of the Tree, we used **Queue** data structure.



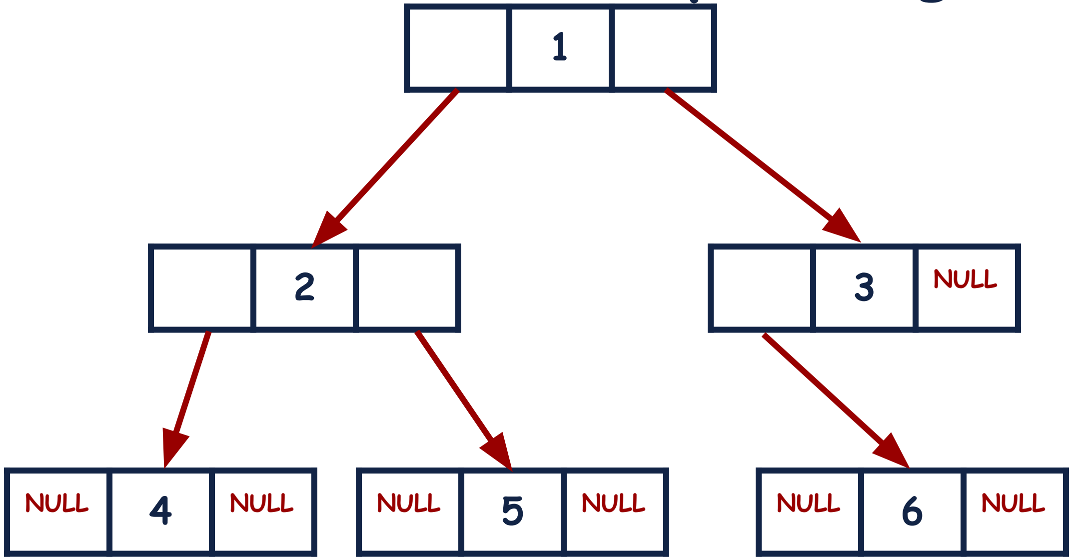
Traversal: Binary Trees

There are **multiple options** in which we can traverse the binary tree as it is a non-linear data structure.



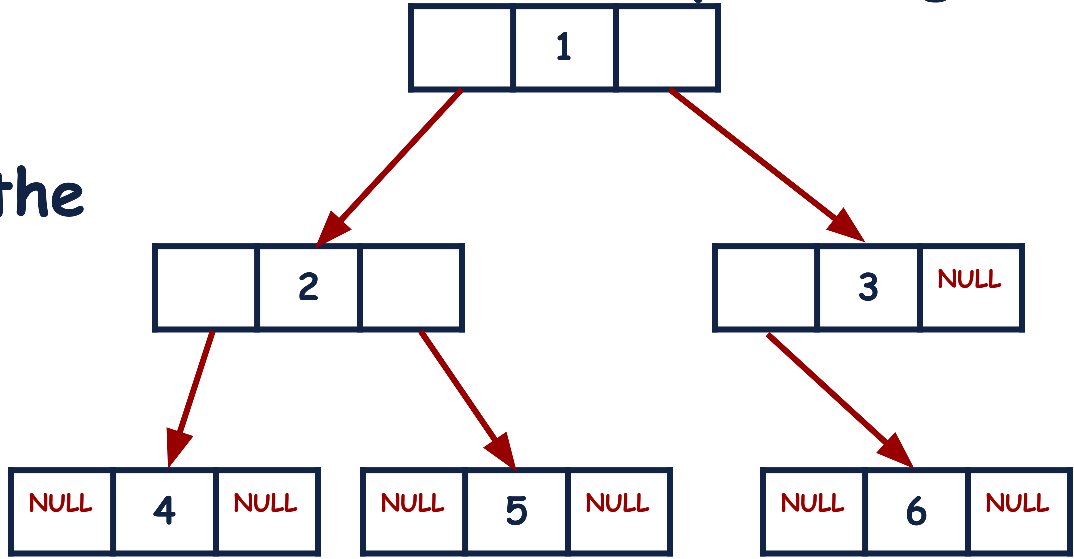
Traversal: Binary Trees

Now, instead of the Queue data structure, lets use **Stack** data structure and see what is the output we get.



Traversal: Binary Trees

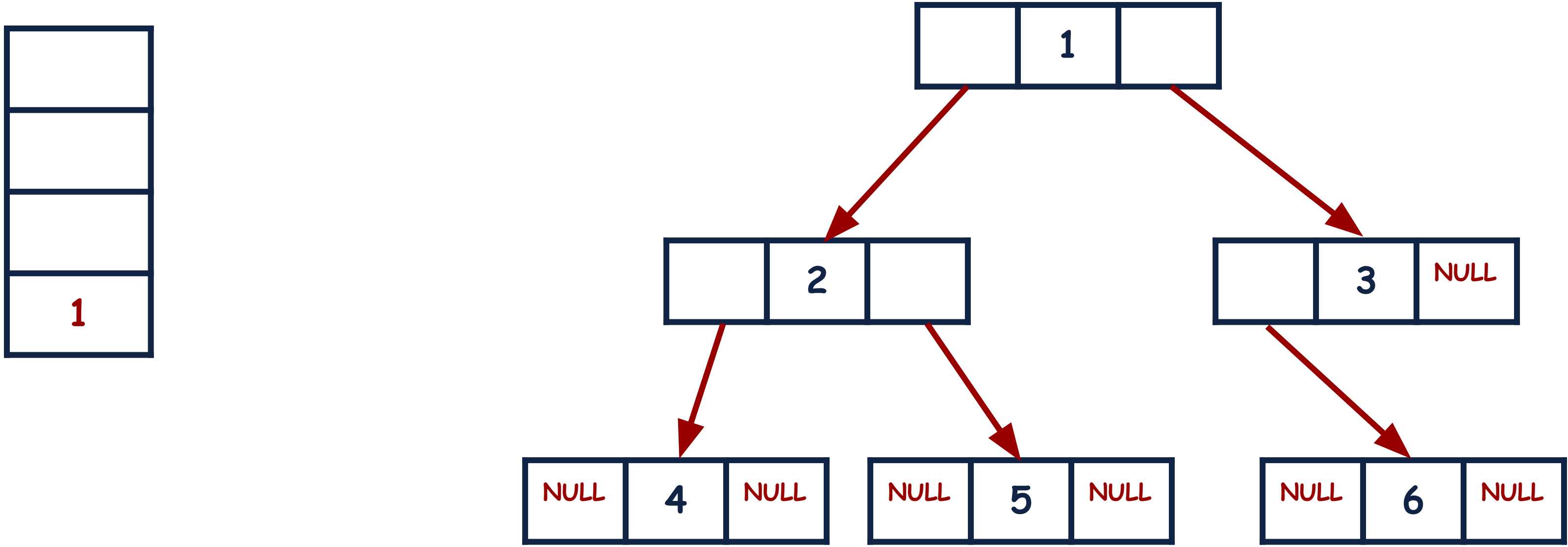
Now, instead of the Queue data structure, lets use **Stack** data structure and see what is the output we get.



But instead of pushing the left node first on the stack, we will push the right node first.

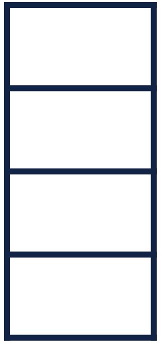
Traversal: Binary Trees

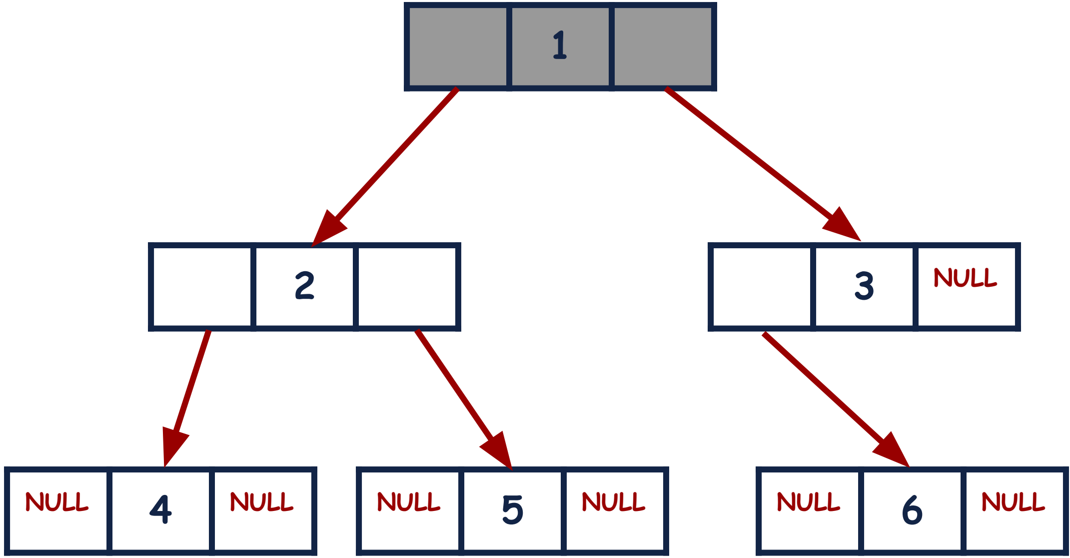
Let's push the root node on to the Stack.



Traversal: Binary Trees

Pop the top node from the stack and print its value.



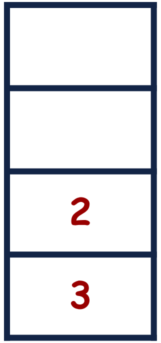


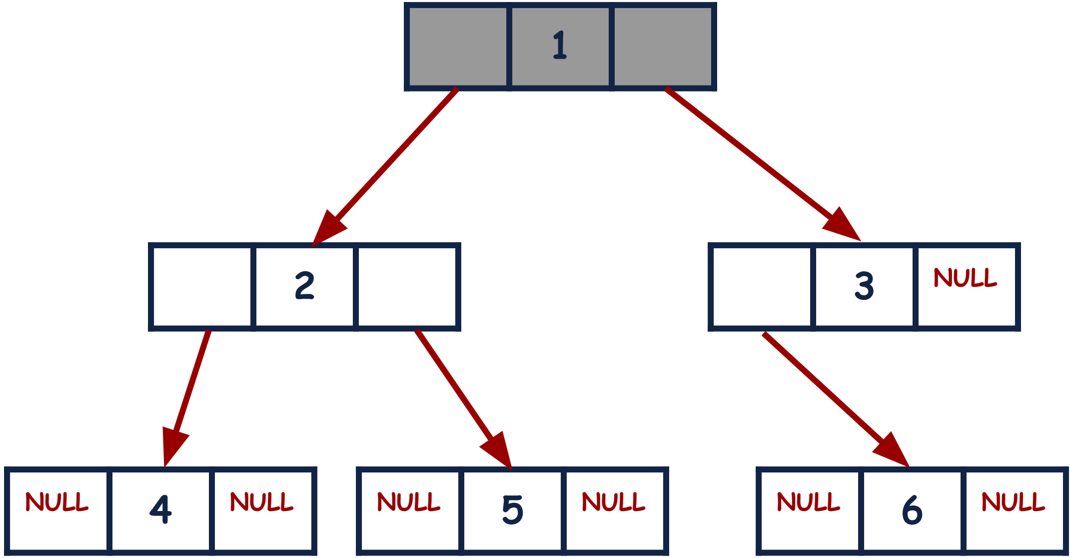
Output:

1,

Traversal: Binary Trees

Push the right node and then the left node onto the stack if they are not NULL.



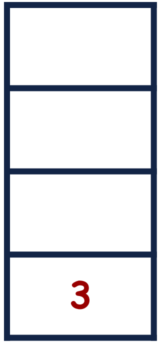


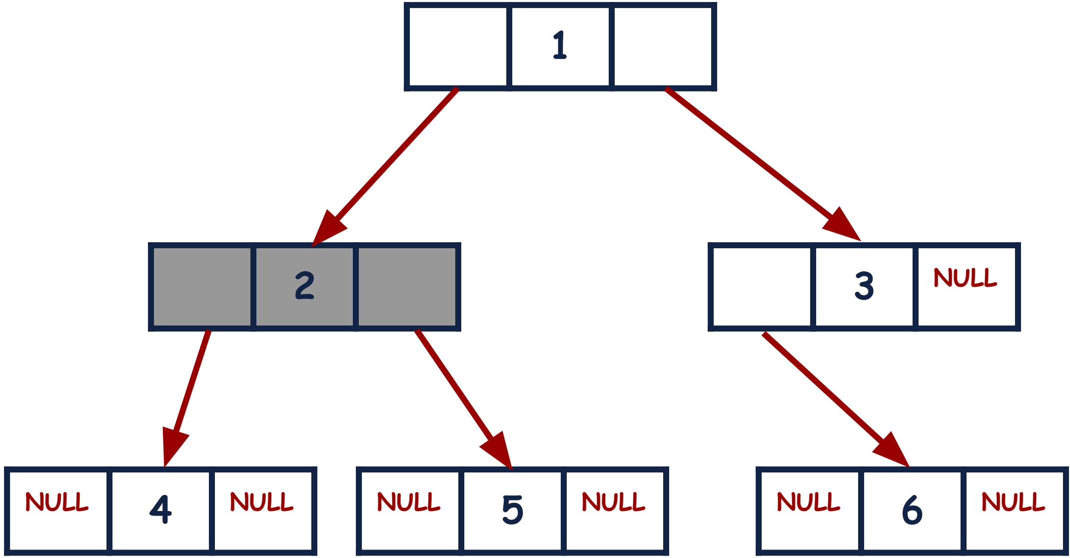
Output:

1,

Traversal: Binary Trees

Pop the top node from the stack and print its value.

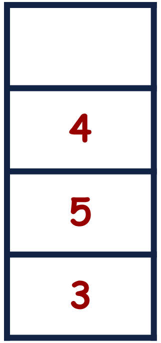


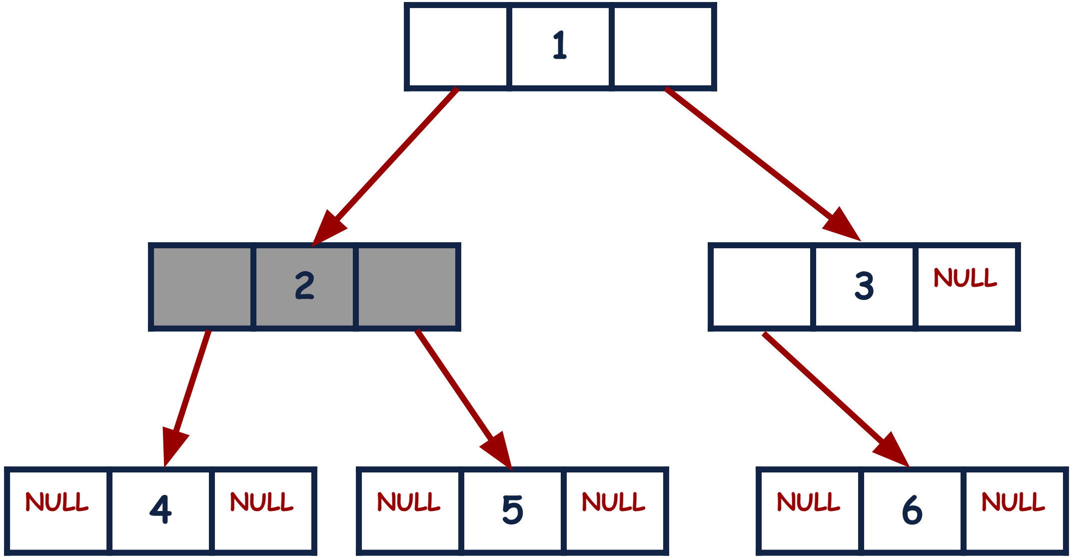


Output:
1, 2,

Traversal: Binary Trees

Push the right node and then the left node onto the stack if they are not NULL.

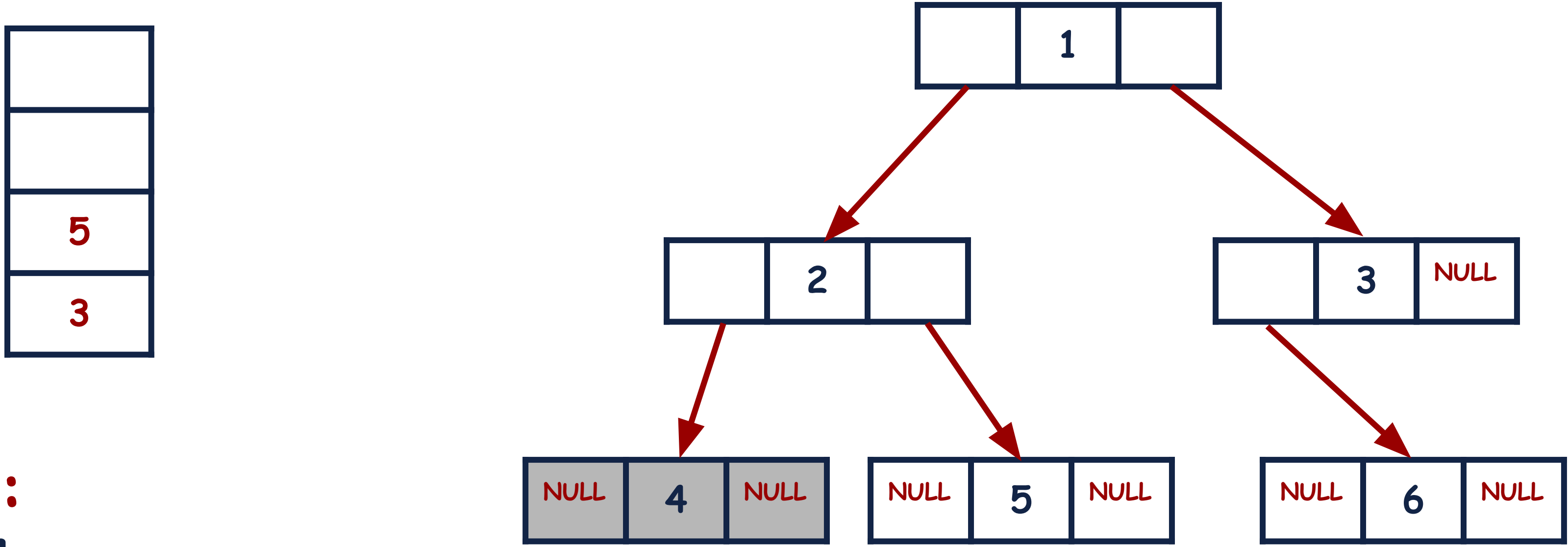




Output:
1, 2,

Traversal: Binary Trees

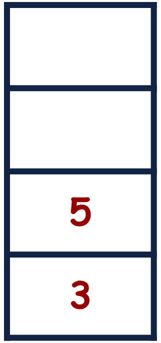
Pop the top node from the stack and print its value.

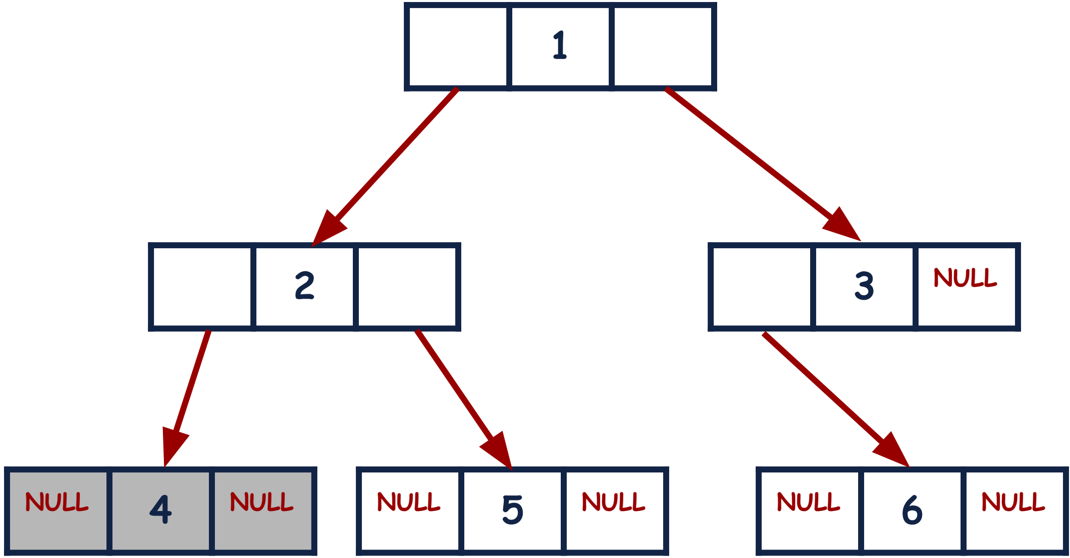


Output:
1, 2, 4

Traversal: Binary Trees

Push the right node and then the left node onto the stack if they are not NULL.

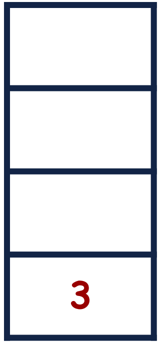


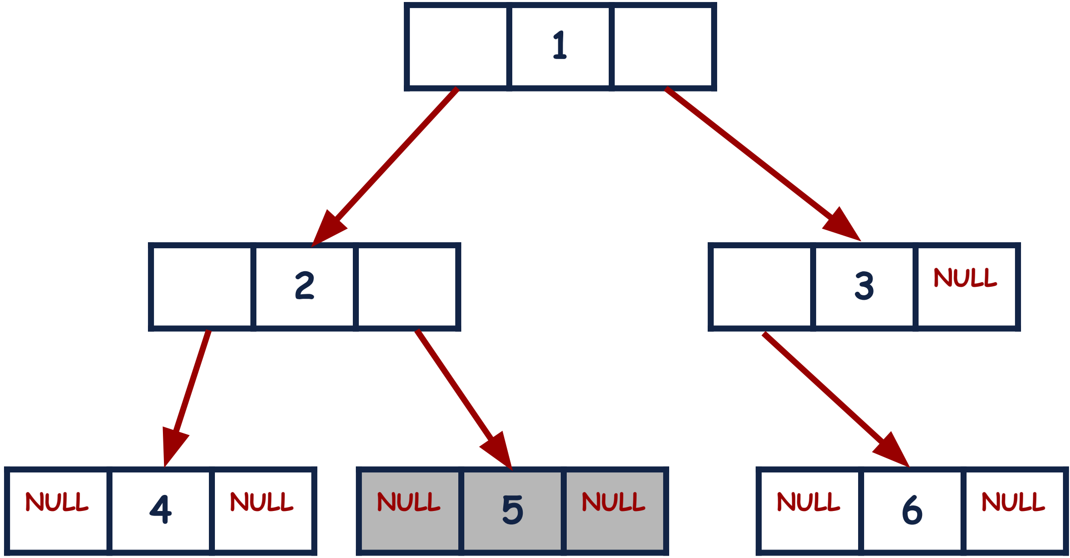


Output:
1, 2, 4

Traversal: Binary Trees

Pop the top node from the stack and print its value.



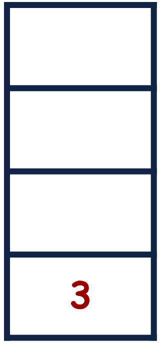


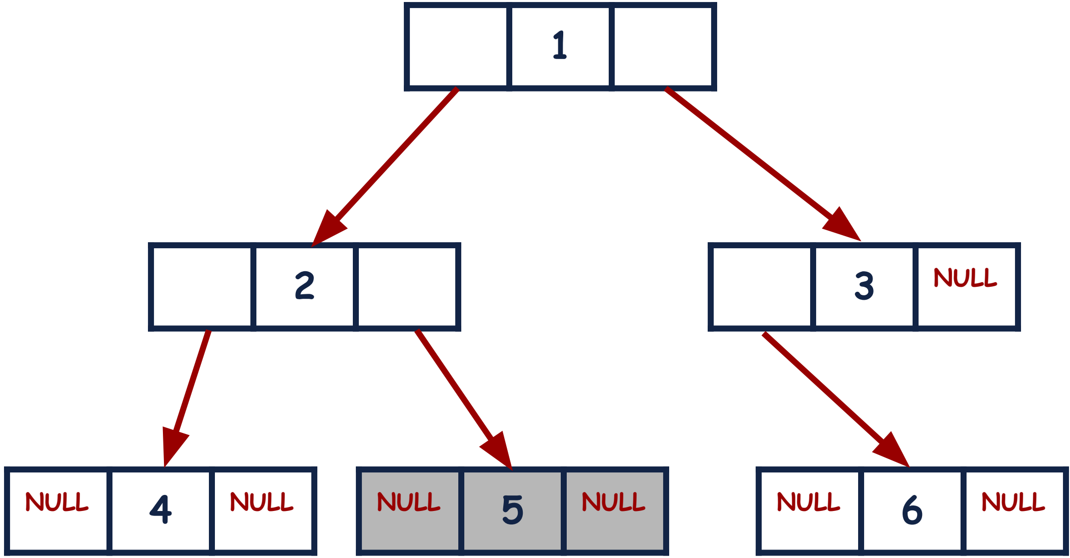
Output:

1, 2, 4, 5,

Traversal: Binary Trees

Push the right node and then the left node onto the stack if they are not NULL.



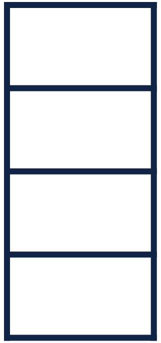


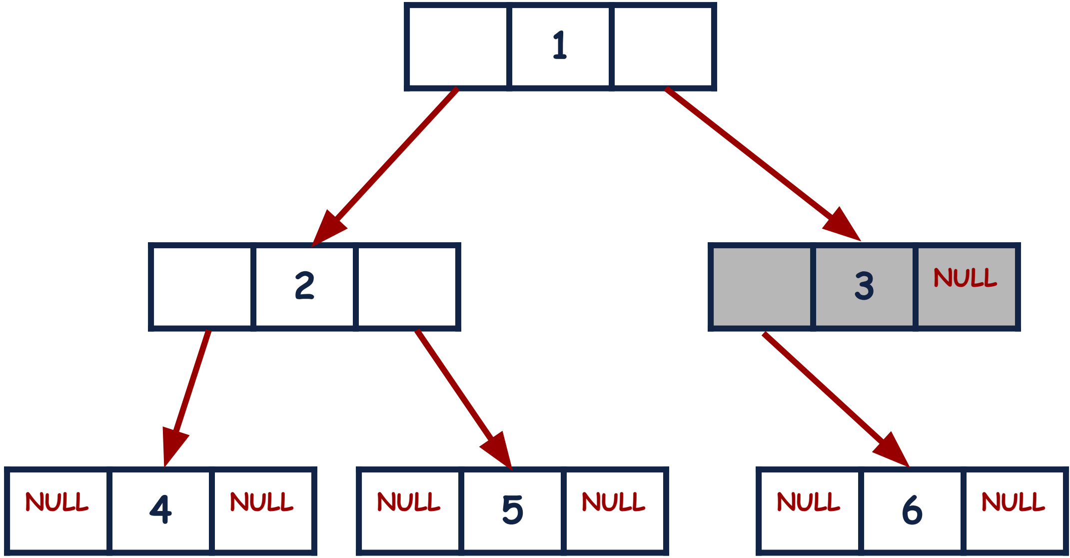
Output:

1, 2, 4, 5,

Traversal: Binary Trees

Pop the top node from the stack and print its value.



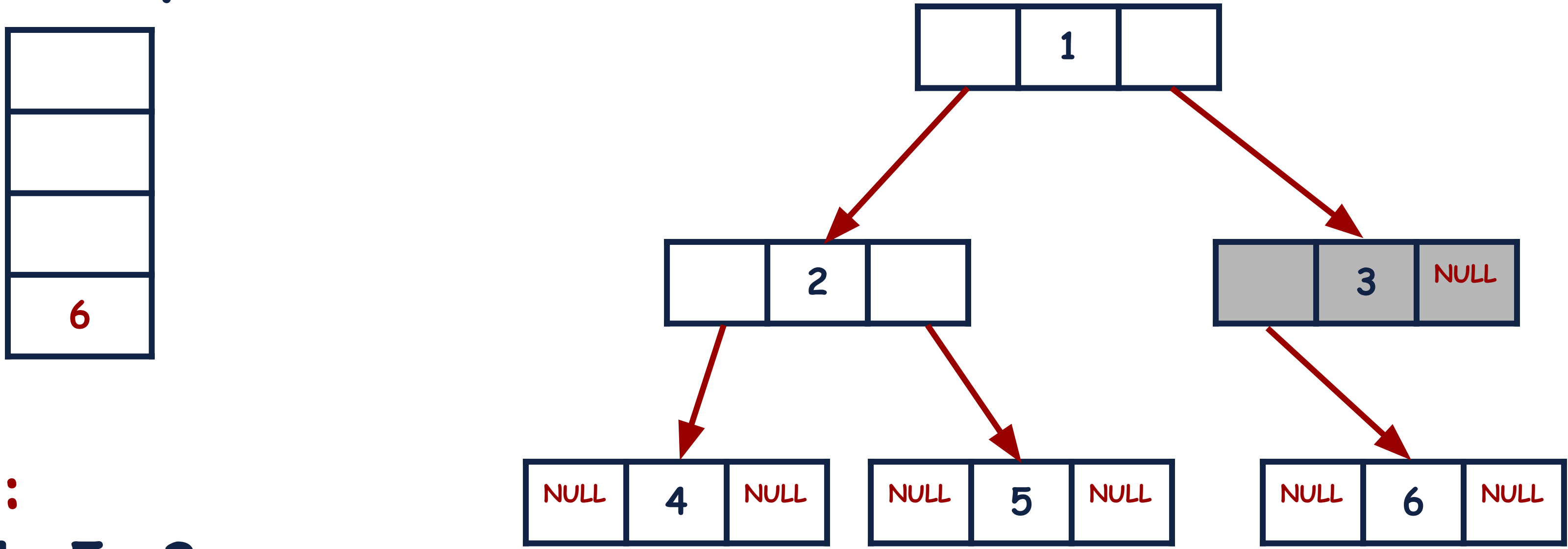


Output:

1, 2, 4, 5, 3,

Traversal: Binary Trees

Push the right node and then the left node onto the stack if they are not NULL.

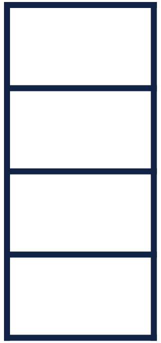


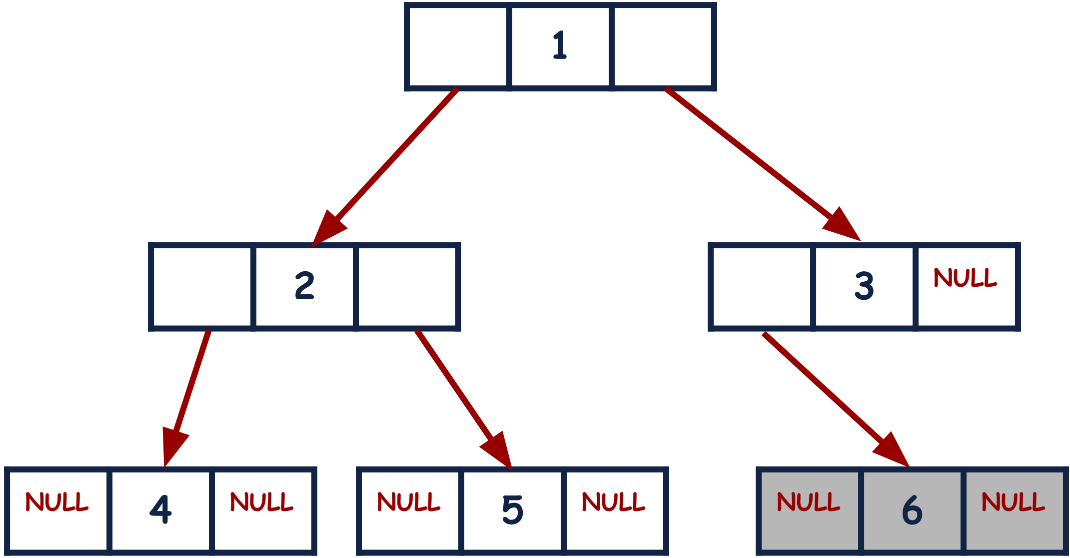
Output:

1, 2, 4, 5, 3,

Traversal: Binary Trees

Pop the top node from the stack and print its value.



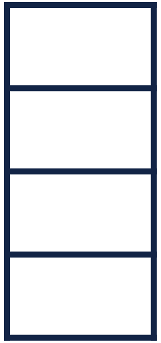


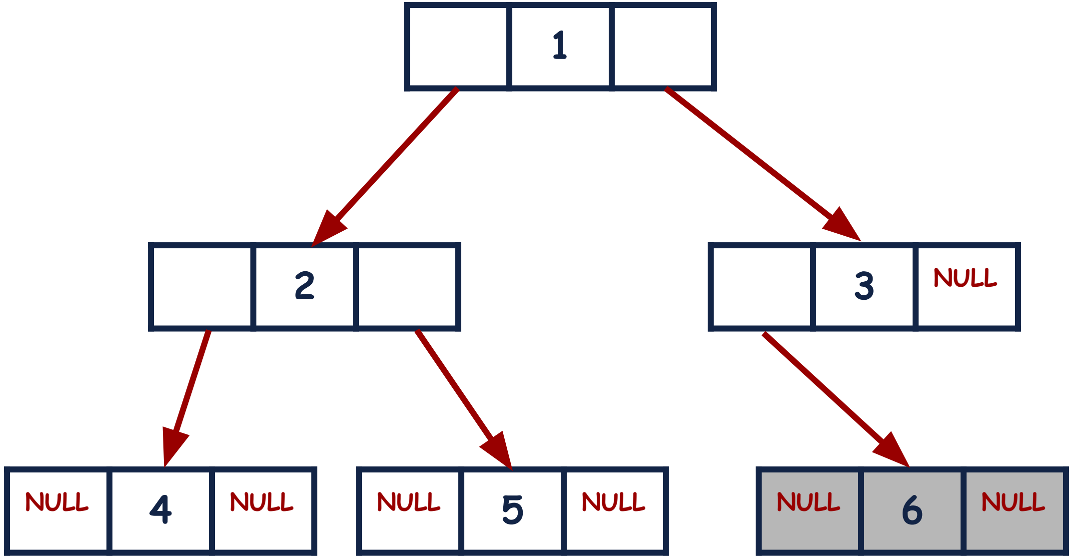
Output:

1, 2, 4, 5, 3, 6

Traversal: Binary Trees

Stop if the Stack is empty.

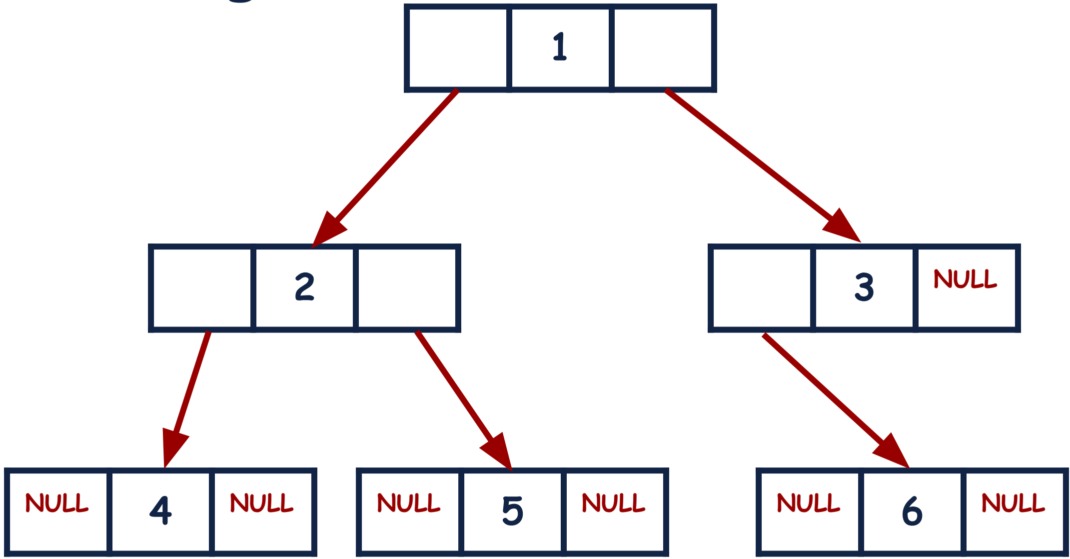




Output:
1, 2, 4, 5, 3, 6

Traversal: Binary Trees

In this Traversal, we are traversing from the root to the left subtree then to the right subtree.



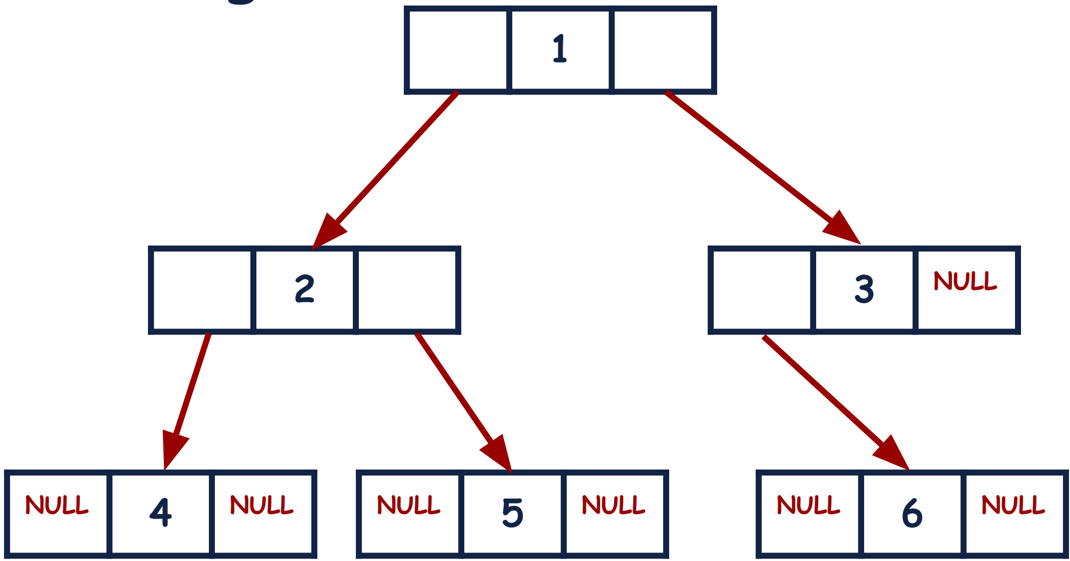
Output:

1, 2, 4, 5, 3, 6

Traversal: Binary Trees

In this Traversal, we are traversing from the root to the left subtree then to the right subtree.

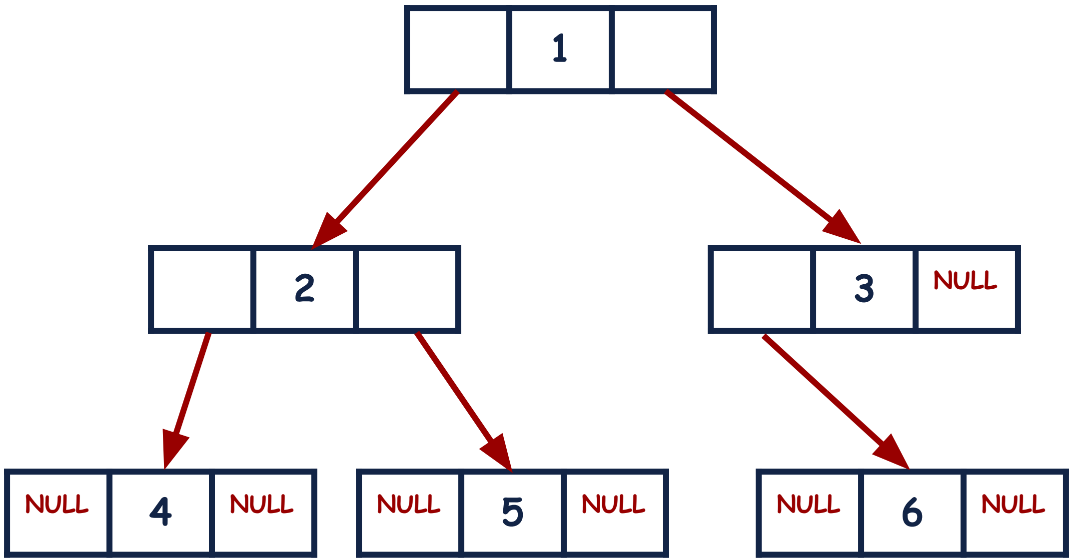
Root, Left, Right (DLR)



Pre-Order Traversal: Binary Trees

In this Traversal, root node is displayed before the left and the right subtrees. Therefore, it is called **Pre-Order Traversal**.

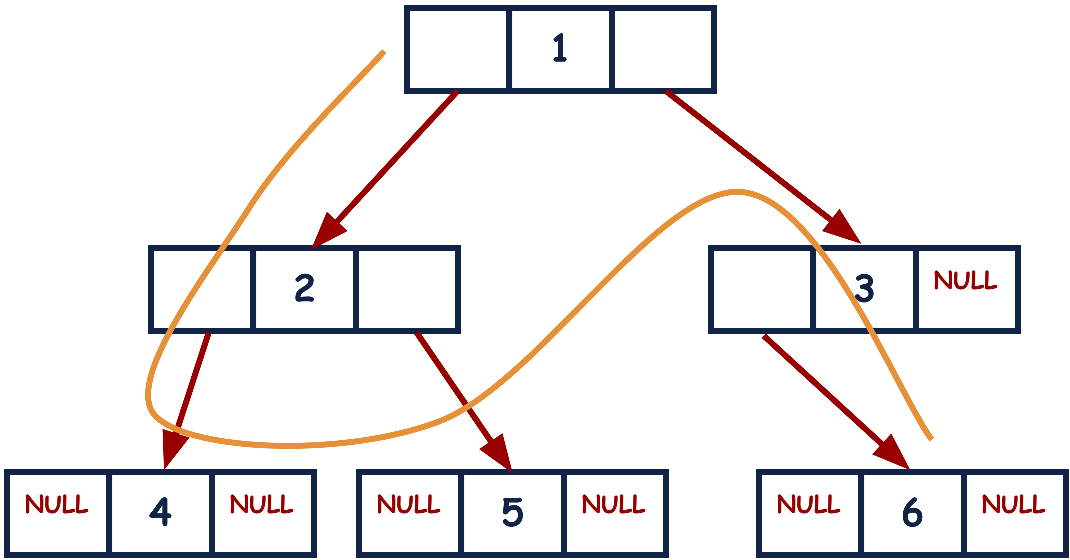
Root, Left, Right (DLR)



Pre-Order Traversal: Binary Trees

In this Traversal, root node is displayed before the left and the right subtrees. Therefore, it is called **Pre-Order Traversal**.

Root, Left, Right (DLR)



Pre-Order Traversal: Pseudocode

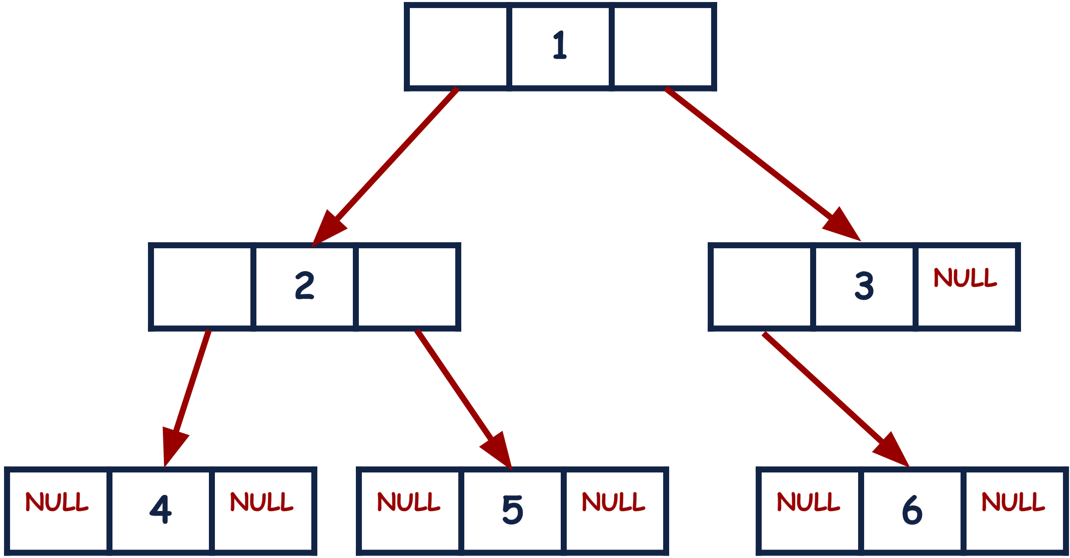
1. Declare the Stack
2. Push the root node
3. while(Stack is not empty)
    a. Pop the node
    b. Print the value
    c. if(right node is not NULL)
        i. Push the right node
    d. if(left node is not NULL)
        i. Enqueue the left node

Pre-Order Traversal: Implementation

```
void preOrder()
    {
        stack<node *> stack;
        stack.push(root);
        while (!stack.empty())
        {
                node * curr = stack.top();
                stack.pop();
                cout << curr->data << " ";
                if(curr->right != NULL)
                    stack.push(curr->right);
                if(curr->left != NULL)
                    stack.push(curr->left);
        }
    }
```

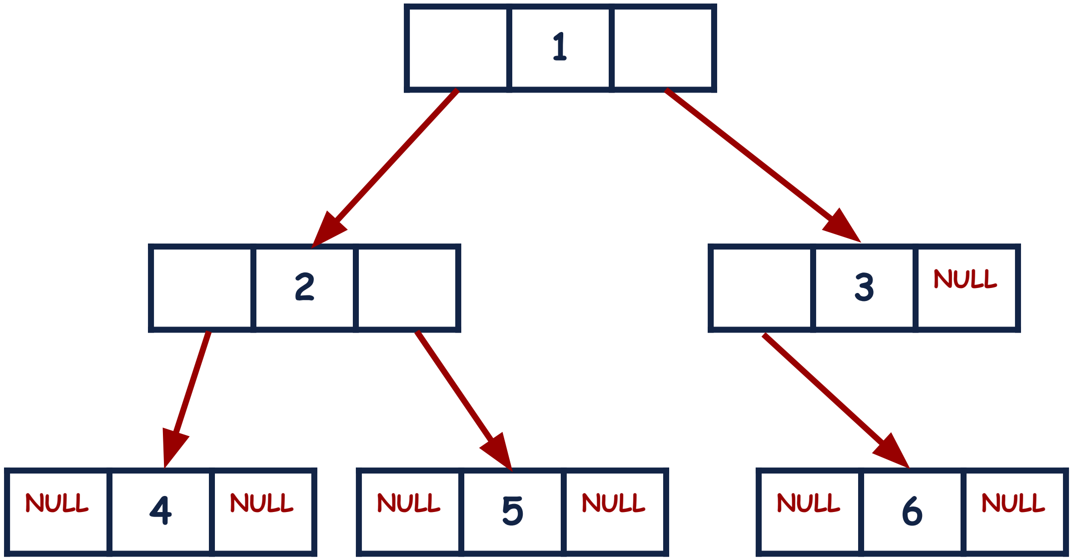
Traversal: Binary Trees

Now, instead of pushing the right node onto the stack, lets push the left node first.



Traversal: Binary Trees

Now, instead of pushing the right node onto the stack, lets push the left node first.



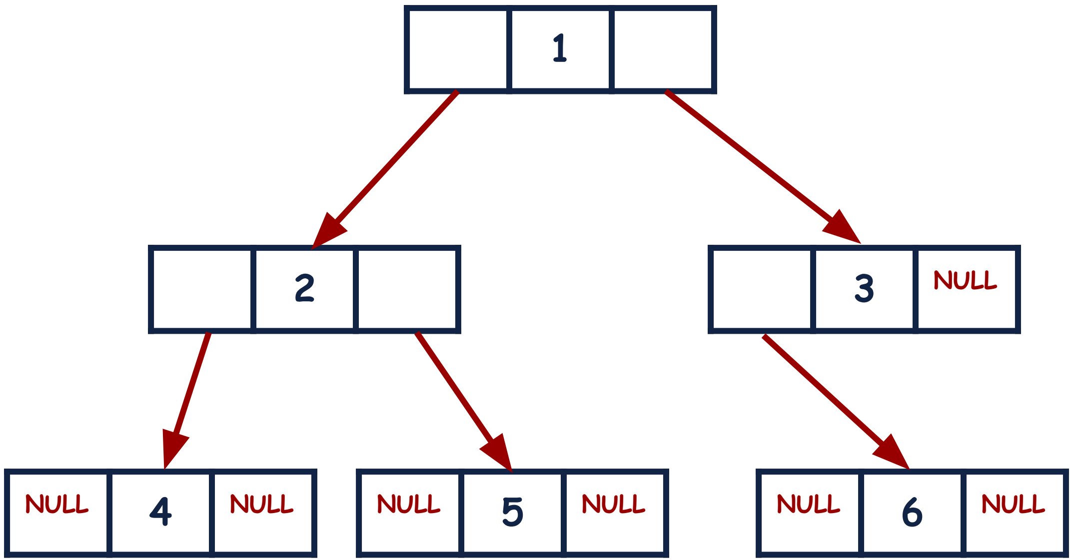
Output:
1, 3, 6, 2, 5, 4

Traversal: Binary Trees

Now, lets reverse this output.

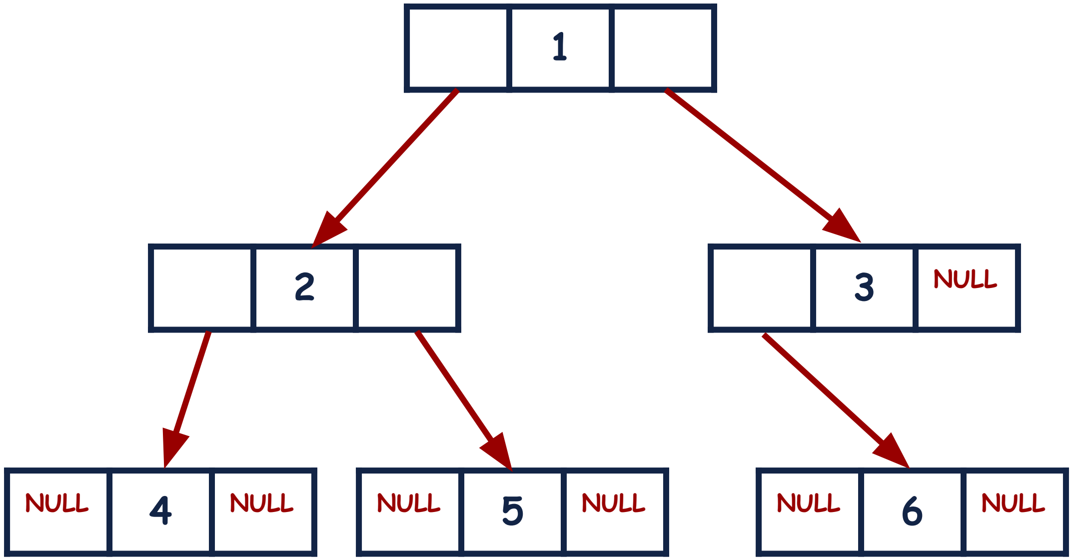
Output:
1, 3, 6, 2, 5, 4

Reversed Output:
4, 5, 2, 6, 3, 1



Traversal: Binary Trees

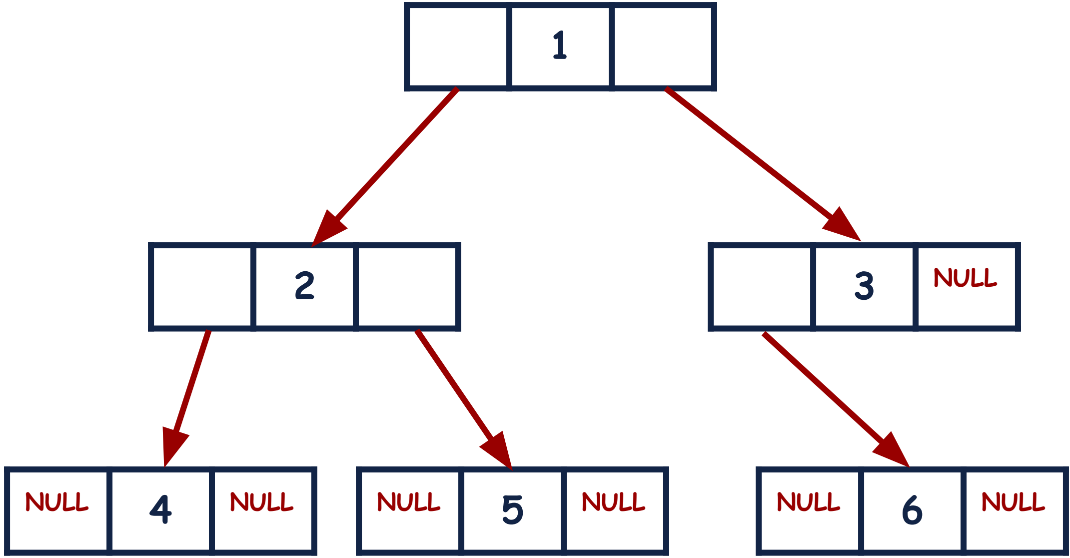
Now, instead of displaying the root node first, we are displaying the left node, then the root node and then the right node.



Traversal: Binary Trees

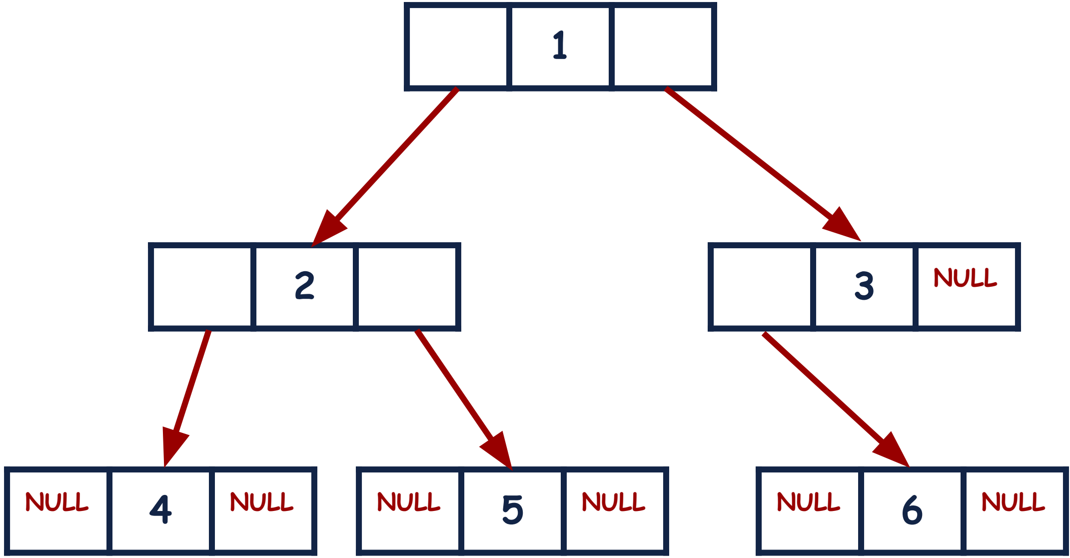
In this Traversal, we are traversing the left subtree first, then the right subtree and then the root node.

Left, Right, Root (LRD)



Post-Order Traversal: Binary Trees

In this Traversal, root node is displayed at the end therefore, it is called Post-Order Traversal.

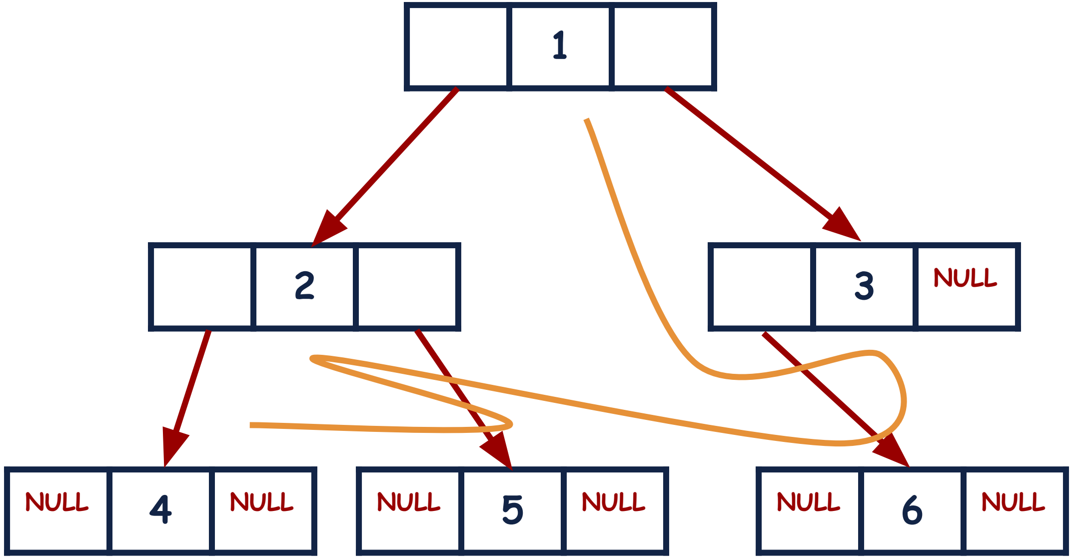


Left, Right, Root (LRD)

Post-Order Traversal: Binary Trees

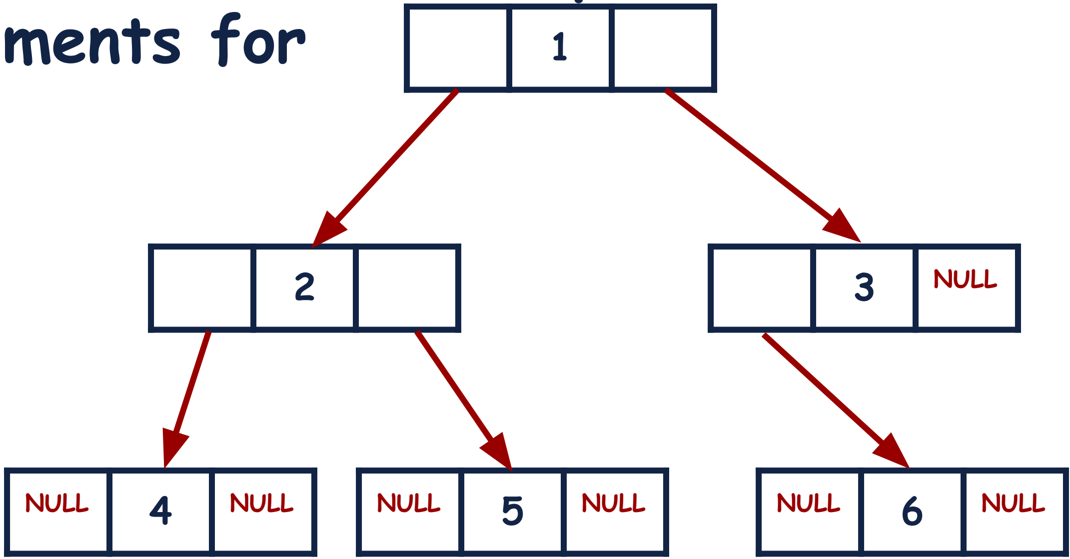
In this Traversal, root node is displayed at the end therefore, it is called **Post-Order Traversal**.

Left, Right, **Root** (LRD)



Post-Order Traversal: Binary Trees

First traverse the tree in Pre-order with right subtree first than the left subtree, store the output in a stack and then pop all the elements for Post Order traversal.



Post-Order:

4, 5, 2, 6, 3, 1

Post-Order Traversal: Pseudocode

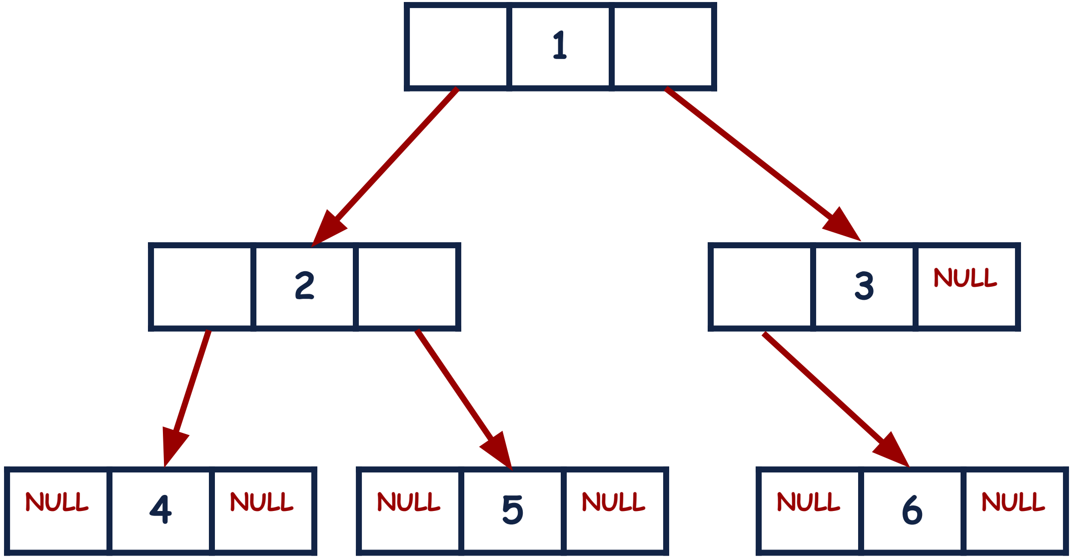
1. Declare 2 Stacks
2. Push the root node onto the stack 1.
3. while(Stack 1 is not empty)
   a. Pop the top most element from the Stack 1.
   b. Push that element onto the Stack 2
   c. if(the left node is not NULL)
      i. Push the left node onto the stack 1.
   d. if(the right node is not NULL)
      i. Push the right node onto the stack 1.
4. while(Stack 2 is not empty)
      i. Pop and print the top most element from the stack 2

Post-Order Traversal: Implementation

```
void postOrder()
    {
        stack<node *> s1, s2;
        s1.push(root);
        while(!s1.empty())
        {
            node * curr = s1.top();
            s2.push(curr);
            s1.pop();
            if(curr->left != NULL)
                s1.push(curr->left);
            if(curr->right != NULL)
                s1.push(curr->right);
        }
        while (!s2.empty())
        {
            cout << s2.top()->data << " ";
            s2.pop();
        }
    }
```

Traversal: Binary Trees

Now, instead of displaying the root node first, lets display the left node, then the root node and then the right node.



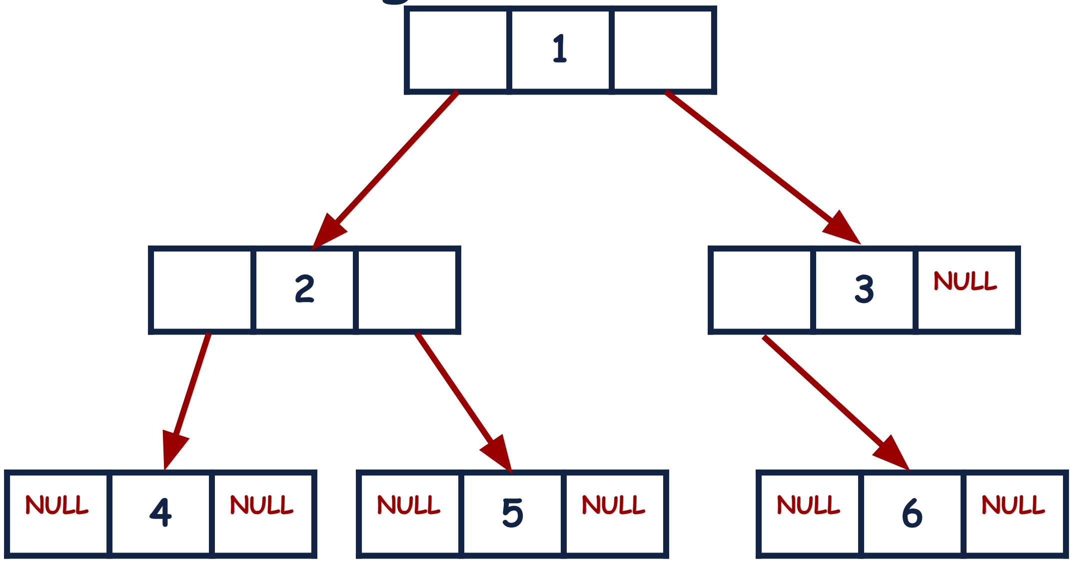
Output:

4, 2, 5, 1, 6, 3

Traversal: Binary Trees

In this Traversal, we are traversing the left subtree first, then the root and then the right subtree.

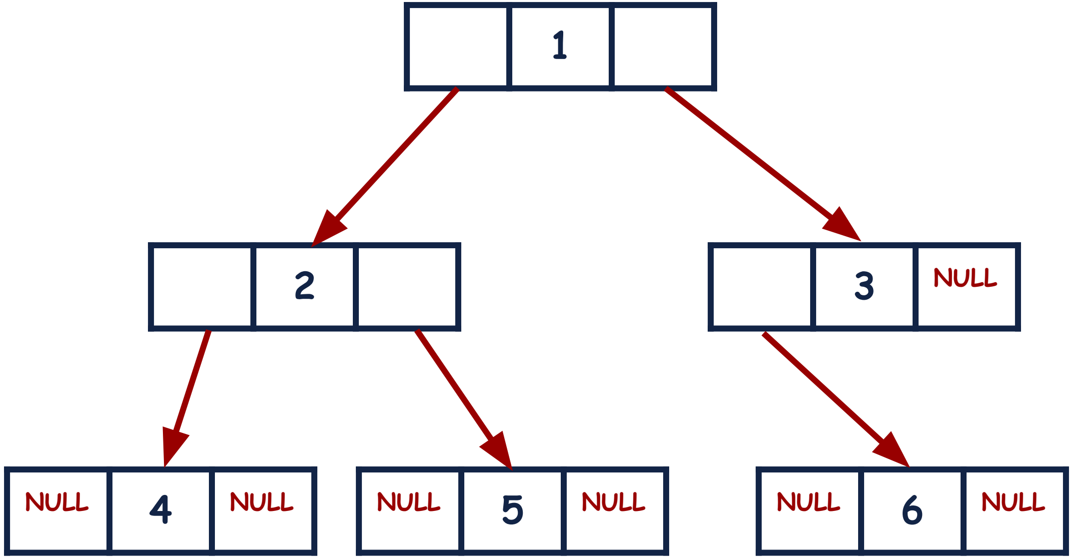
Left, Root, Right (LDR)



Traversal: Binary Trees

In this Traversal, root node is displayed in the middle therefore, it is called **In-Order Traversal**.

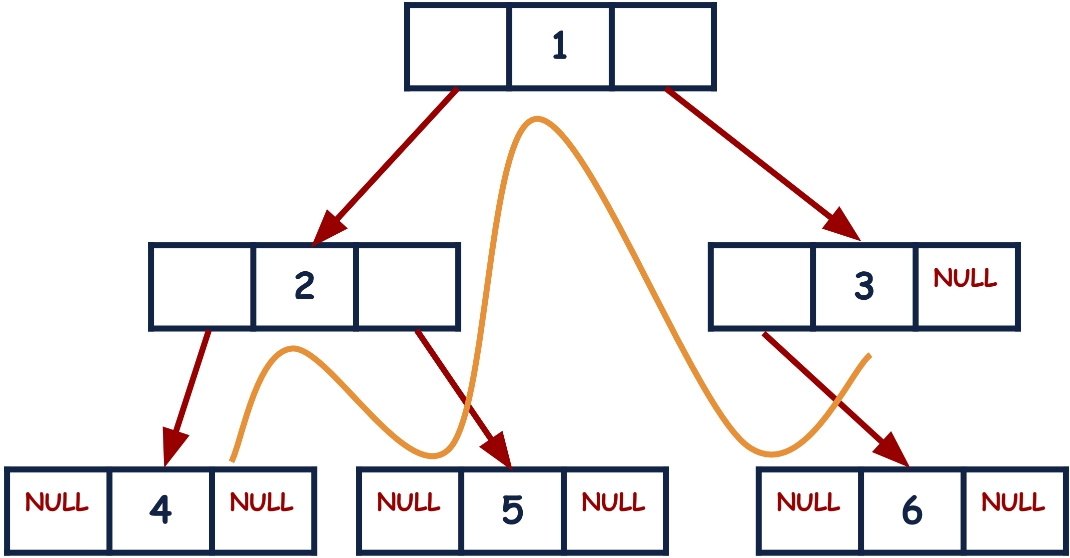
Left, **Root**, Right (LDR)



In-Order Traversal: Binary Trees

In this Traversal, root node is displayed in the middle therefore, it is called **In-Order Traversal**.

Left, **Root**, Right (LDR)



In-Order Traversal: Pseudocode

1. Declare the Stack
2. Initialize the current pointer to root node.
3. while(Stack is not empty || current pointer is not NULL)
   a. if(current is not NULL)
      i. Push the current node to the stack
      ii. Update current to current -> left
   b. else
      i. Set the current to the top most element of stack
      ii. Pop the element from the stack
      iii. Print the value of the current pointer
      iv. Update current to current -> right

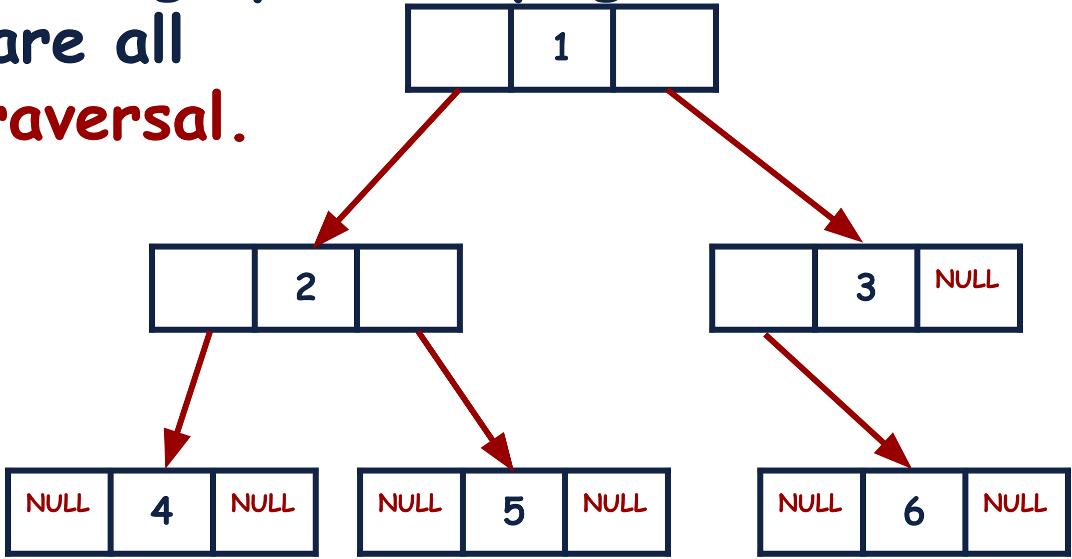
In-Order Traversal: Implementation

```
void inOrder()
    {
        stack<node *> stack;
        node *curr = root;

        while (!stack.empty() || curr != NULL)
        {
            if (curr != NULL)
            {
                stack.push(curr);
                curr = curr->left;
            }
            else
            {
                curr = stack.top();
                stack.pop();
                cout << curr->data << " ";
                curr = curr->right;
            }
        }
    }
```

Depth First Traversal: Binary Trees

In these Traversals, we are going as deep as possible down one path before backing up and trying a different one. Therefore, these are all types of Depth First Traversal.



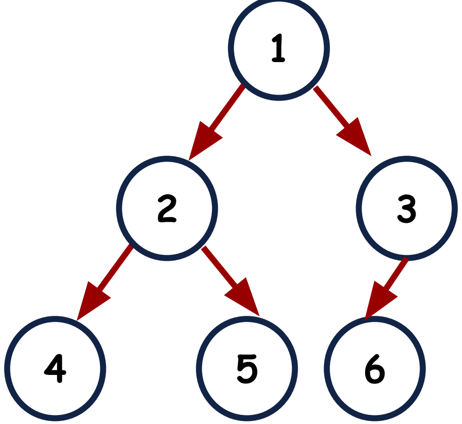
Traversals: Binary Trees

There are 2 types of traversals

1. Breadth First Traversal

2. Depth First Traversal

Traversals: Binary Trees

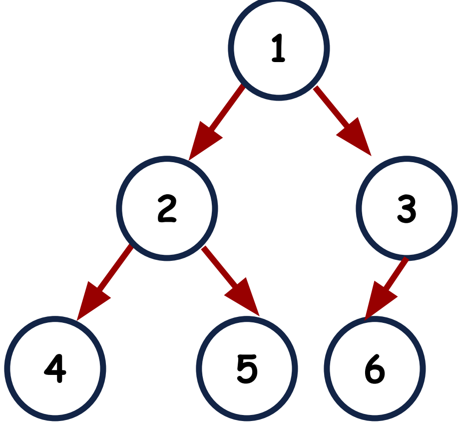


There are 2 types of traversals

1. Breadth First Traversal
    a. Level Order Traversal (1, 2, 3, 4, 5, 6)

2. Depth First Traversal

Traversals: Binary Trees

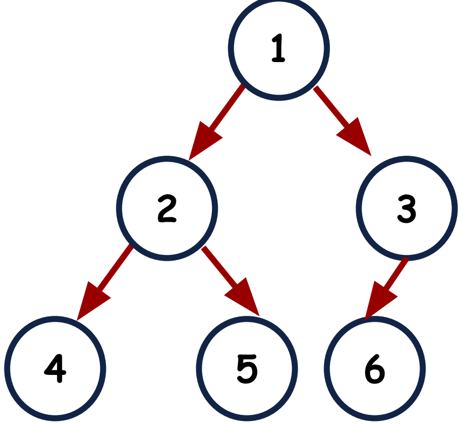


There are **2 types** of traversals

1. Breadth First Traversal
   a. Level Order Traversal (1, 2, 3, 4, 5, 6)

2. Depth First Traversal
   a. Pre-Order Traversal (1, 2, 4, 5, 2, 6)
   b. In-Order Traversal (4, 2, 5, 1, 6, 3)
   c. Post-Order Traversal (4, 5, 2, 6, 3, 1)

Traversals: Binary Trees



There are 2 types of traversals

1. Breadth First Traversal (Queue)
   a. Level Order Traversal (1, 2, 3, 4, 5, 6)

2. Depth First Traversal (Stack)
   a. Pre-Order Traversal (1, 2, 4, 5, 2, 6)
   b. In-Order Traversal (4, 2, 5, 1, 6, 3)
   c. Post-Order Traversal (4, 5, 2, 6, 3, 1)

Learning Objective

Students should be able to traverse the binary Trees in Depth First (Pre, In and Post Order) to solve the problems efficiently.

Self Assessment

1. Implement the Family Tree Application with the following menus
   a. Add a Person
   b. View a Person
   c. Find the Parent of the Person
   d. Find the Children of the Person
   e. View the Family in Breadth First Traversal
   f. View the Family in Depth First Traversal
      i. Pre-Order
      ii. In-Order
      iii. Post-Order

Self Assessment

1. https://leetcode.com/problems/binary-tree-preorder-traversal/
2. https://leetcode.com/problems/binary-tree-inorder-traversal/
3. https://leetcode.com/problems/binary-tree-postorder-traversal/
4. https://leetcode.com/problems/diameter-of-binary-tree/
5. https://leetcode.com/problems/binary-tree-right-side-view/