Binary Tree and its operations

Getting to know Binary Tree:)

Outline

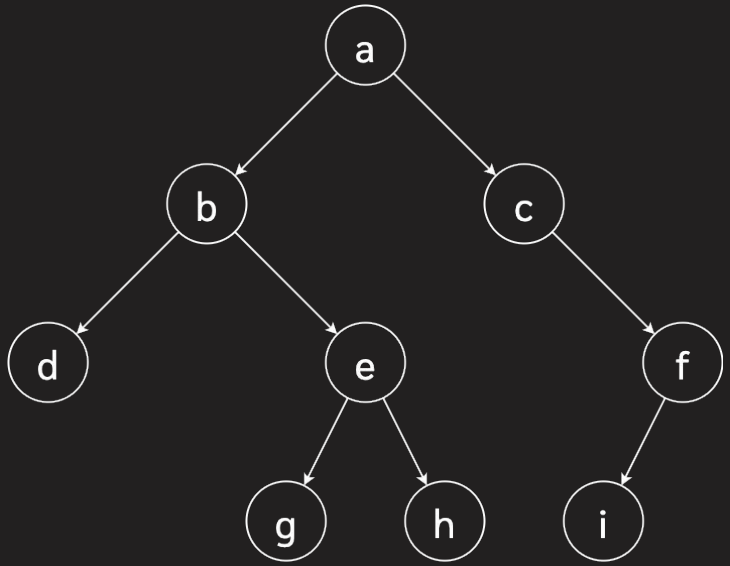
1. Binary Tree
2. Breadth First Traversal (Level Order Traversal)
3. Depth First Traversals
    a. Inorder Traversal
    b. Preorder Traversal
    c. Postorder Traversal
4. Insertion
5. Deletion

Open in Google slides click here

Binary Tree

Binary Tree

- Each node in Tree has at most 2 children call
  - Left child
  - Right child

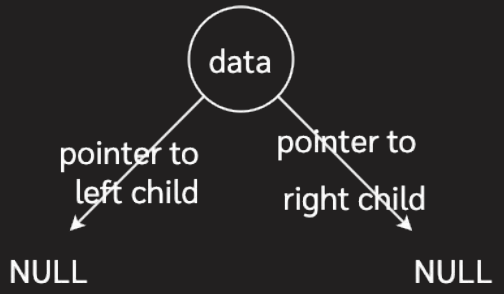


Create Tree in C/C++

- Represent by a pointer to topmost node(root) in Tree
- If tree is empty, value of root is NULL
- A Tree node contains following parts
  - Data
  - Pointer to left child
  - Pointer to right child

Example 1

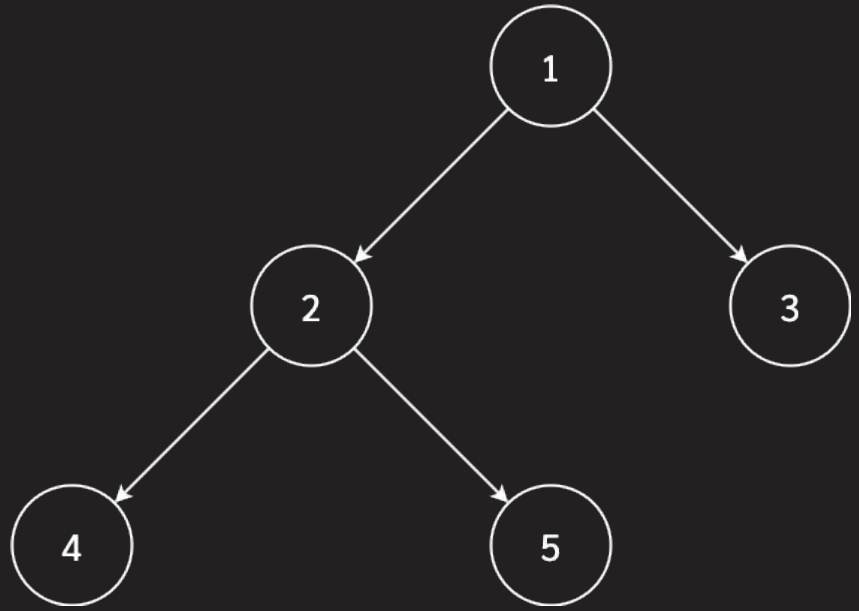
```
struct Node
{
    int data;
    struct Node* left, *right;
    Node(int data)
    {
        this->data = data;
        left = right = NULL;
    }
};
```



Example 1

```
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);

    return 0;
}
```

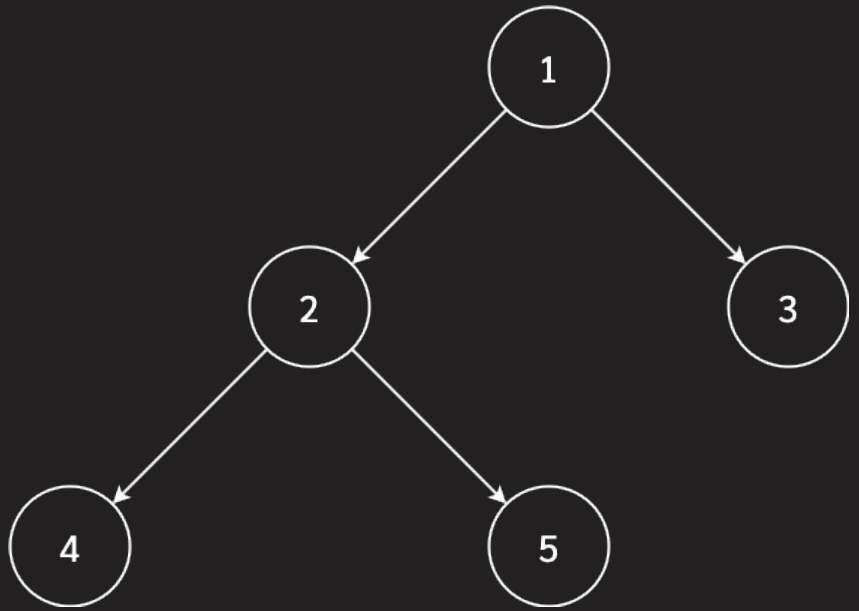


Breadth First Traversal

(Level Order Traversal)

Breadth First Traversal

Given:



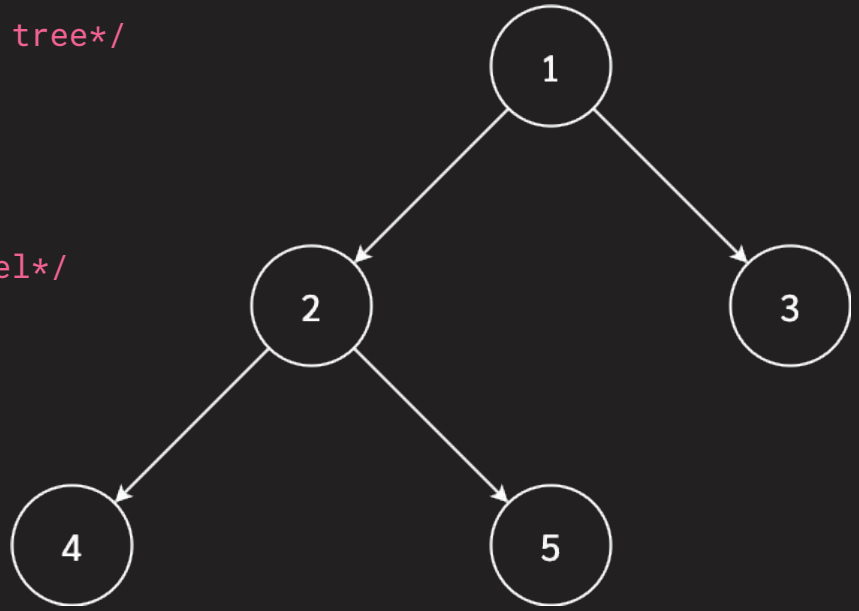
Traversal Output:

1 -> 2 -> 3 -> 4 -> 5

Print a given level

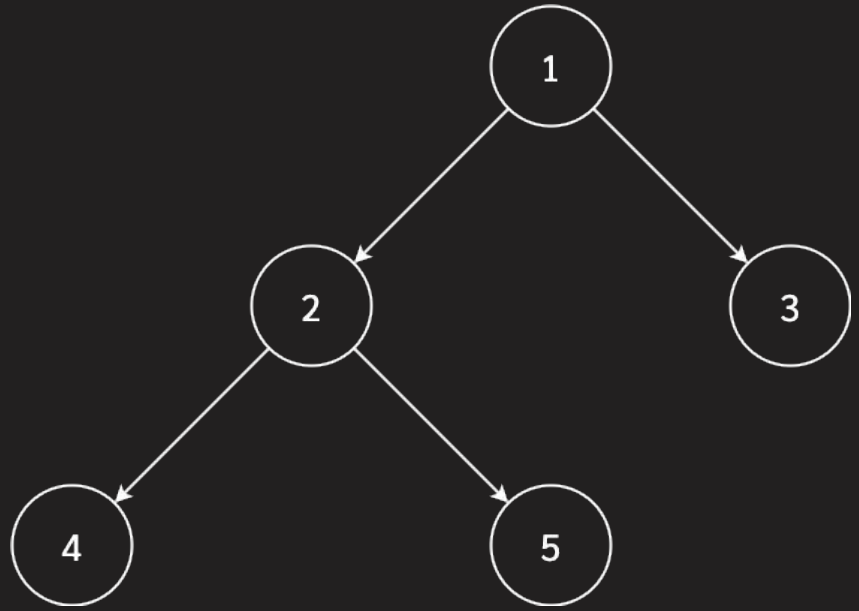
Algorithm

```
/*Function to print level order traversal of tree*/
printLevelorder(tree){
    for d = 0 to height(tree)
        printGivenLevel(tree, d);}
/*Function to print all nodes at a given level*/
printGivenLevel(tree, level){
    if tree is NULL then return;
    if level is 0, then
     print(tree->data);
    else if level greater than 0, then
     printGivenLevel(tree->left, level-1);
     printGivenLevel(tree->right, level-1);}
```



Using queue

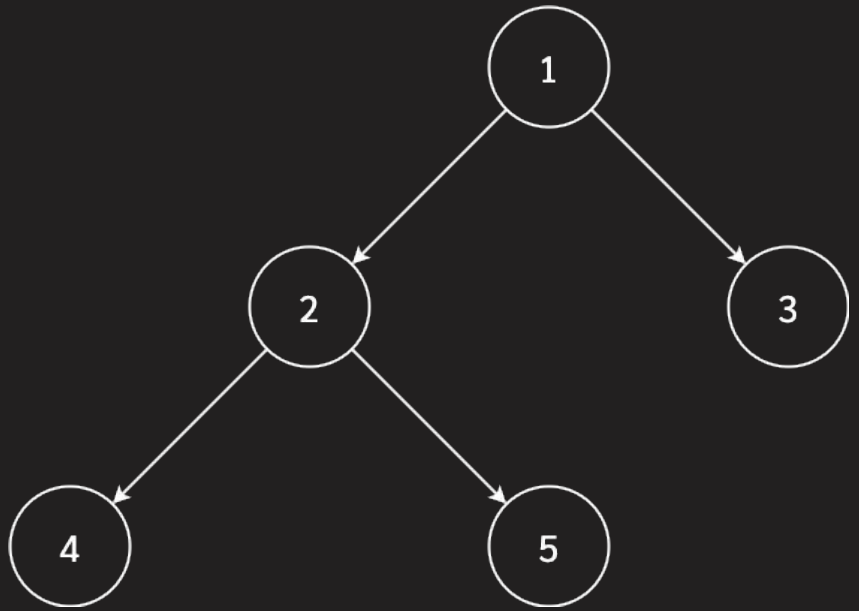
```
printLevelorder(tree)
1) Create an empty queue q
2) temp_node = root /*start from root*/
3) Loop while temp_node is not NULL
    a) print temp_node->data.
    b) Enqueue temp_node's children
(first left then right children) to q
    c) Dequeue a node from q and assign
it's value to temp_node
```



Depth First Traversal

Inorder Traversal

Given:



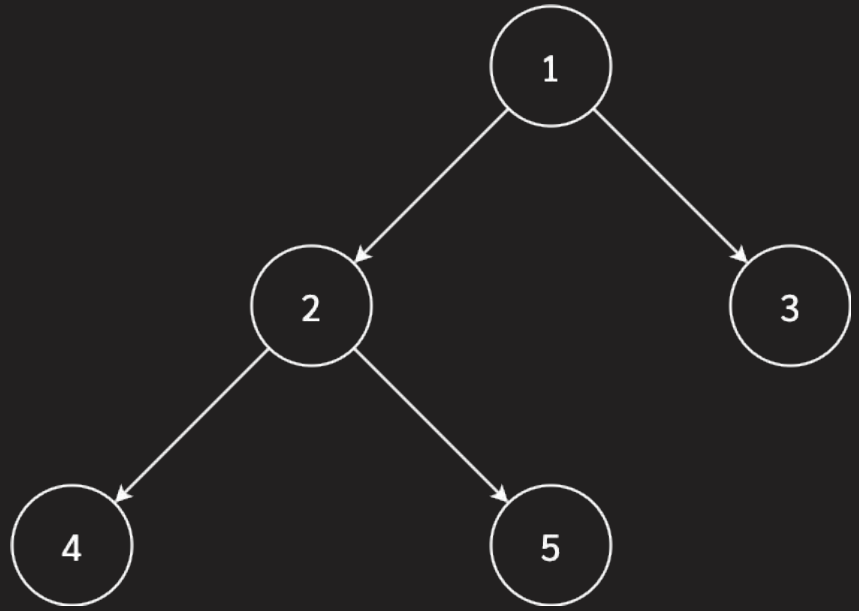
Traversal Output:

4 -> 2 -> 5 -> 1 -> 3

Left, Root, Right

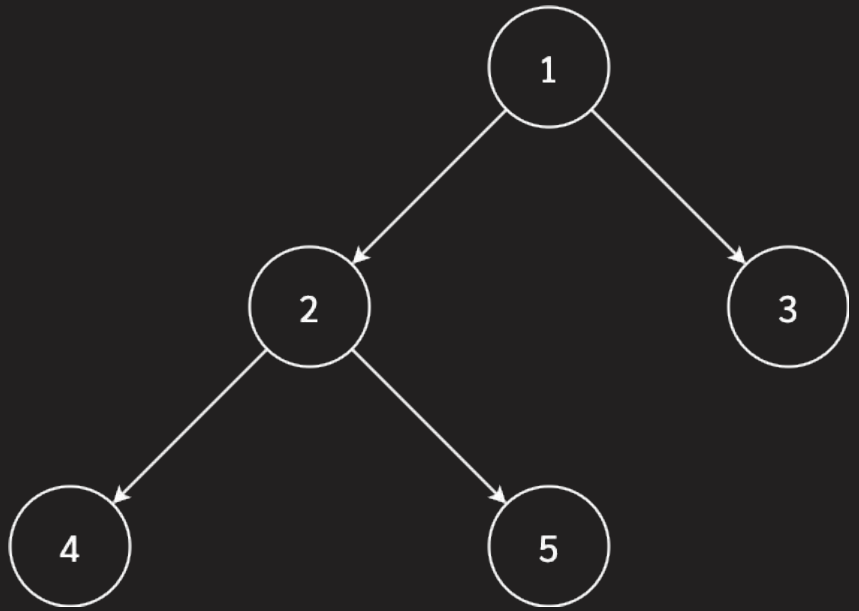
Algorithm

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



Preorder Traversal

Given:



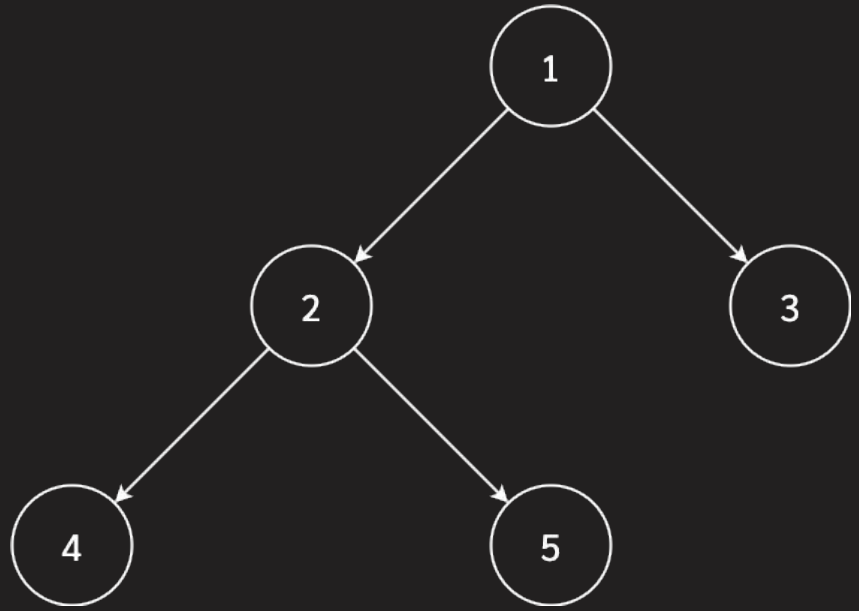
Traversal Output:

1 -> 2 -> 4 -> 5 -> 3

Root, Left, Right

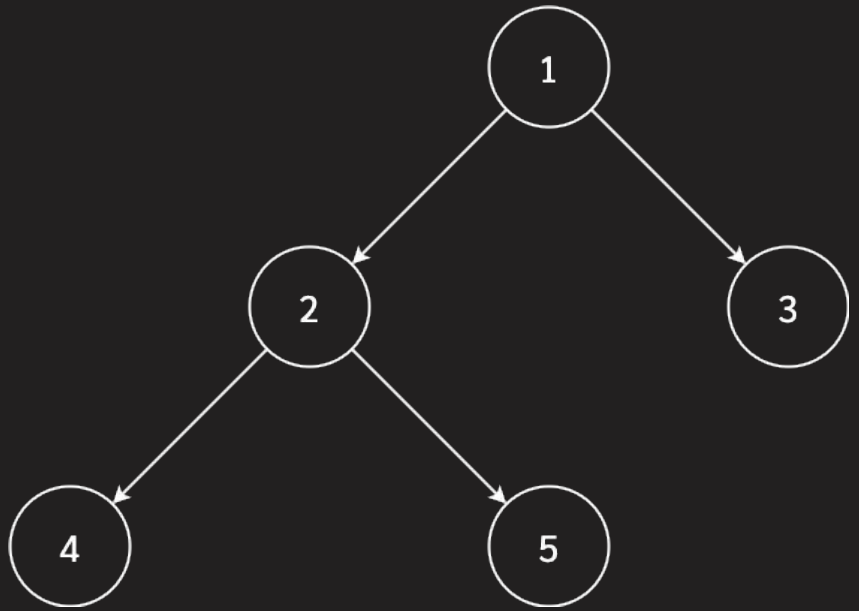
Algorithm

```
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)
```



Postorder Traversal

Given:



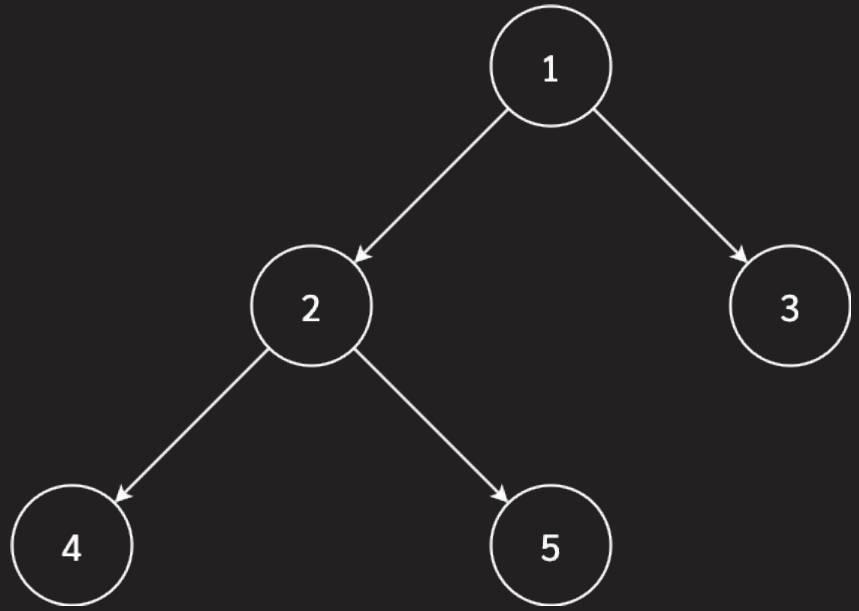
Traversal Output:

4 -> 5 -> 2 -> 3 -> 1

Left, Right, Root

Algorithm

```
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.
```



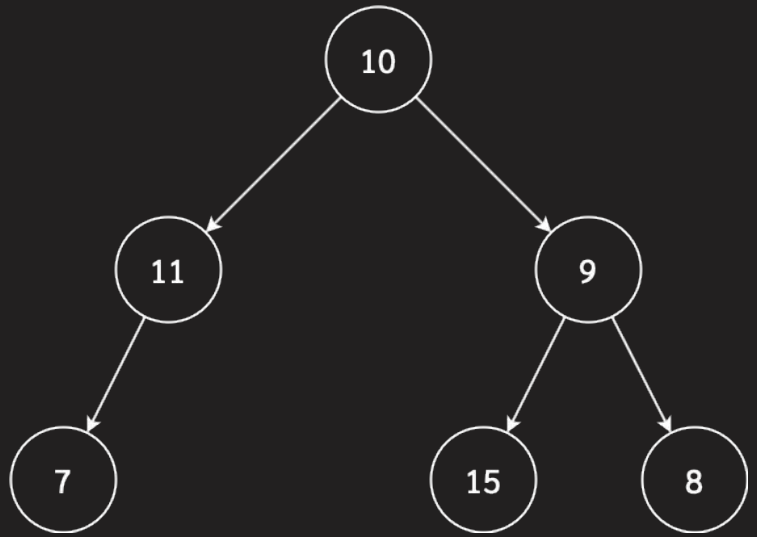
Quiz3: Binary Tree Traversals

Attempt quiz

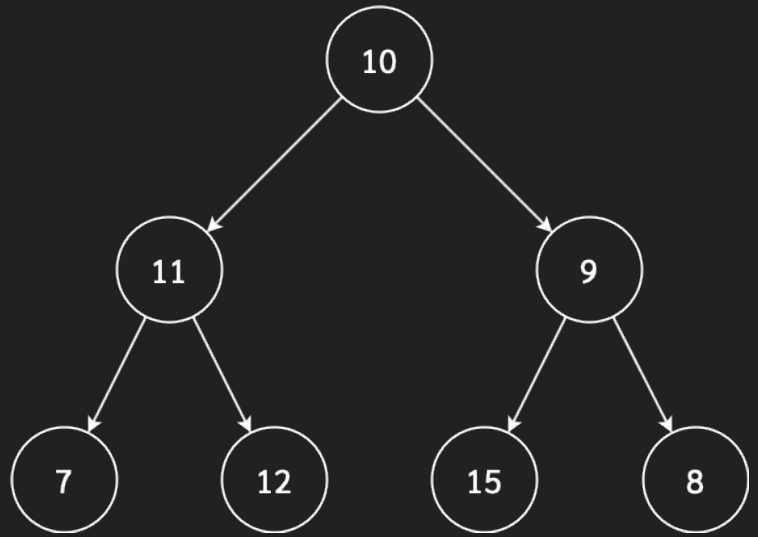
Insertion

Insert the first available position

Given: A graph and want to insert 12



Output:



Insert the first available position

Algorithm

```
printLevelorder(tree)
1) Create an empty queue q
2) temp_node = root /*start from root*/
3) Loop while temp_node is not NULL
    a.1) if temp_node's left children is
available then insert the node and break
    a.2) else Enqueue temp_node's left children
    b.1) if temp_node's right children is
available then insert the node and break
    b.2) else Enqueue temp_node's right
children
```

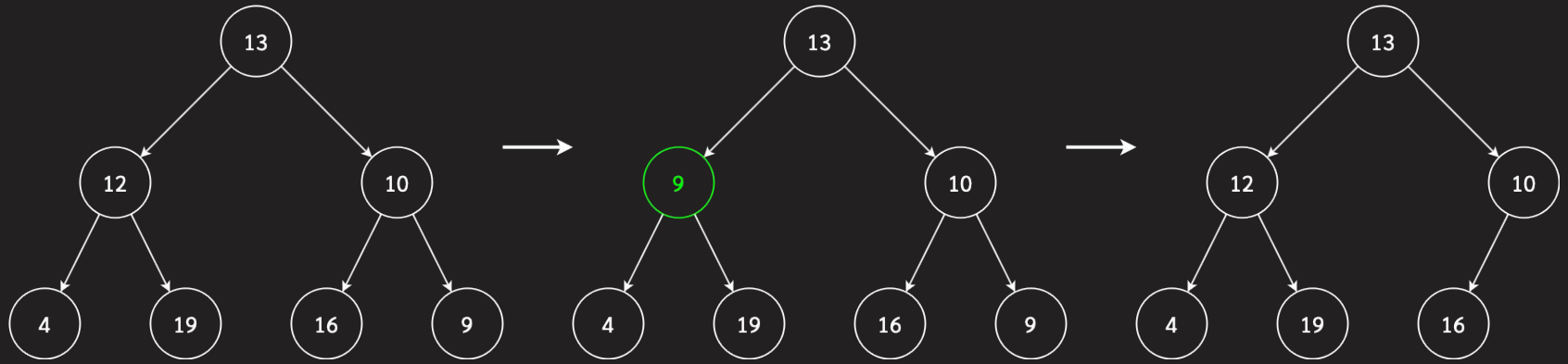
Result

```
Inorder traversal before insertion:
7 11 10 15 9 8
Inorder traversal after insertion:
7 11 12 10 15 9 8
```

Deletion

Maintain completion of Binary Tree

- To make sure that Tree will be shrinked from the bottom
- Given: A graph and want to delete 12



Want to delete 12

Swap with last node (bottom rightmost)

Delete last node

Maintain completion of Binary Tree

Algorithm

Algorithm

1. Starting at root, find the deepest and rightmost node in binary tree and node which we want to delete.
2. Replace the deepest rightmost node's data with node to be deleted.
3. Then delete the deepest rightmost node.

Quiz4: Insertion and Deletion

Attempt Quiz