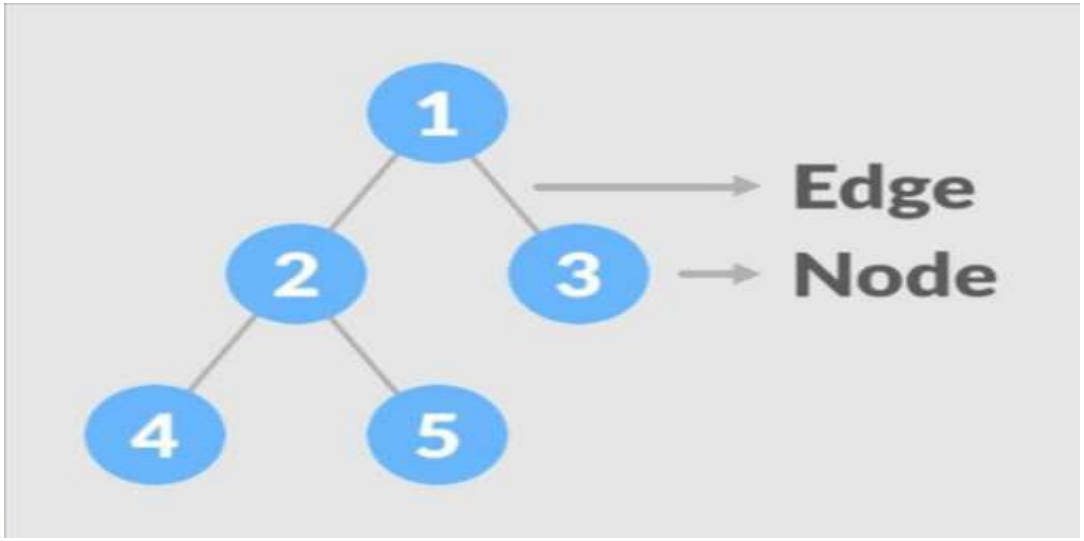
Data Structures and Algorithms(CSE2001)

Module 3

Non-linear Data Structures - Trees and Graph

Introduction to Trees

- A tree is a collection of nodes.
- These nodes are arranged in a hierarchical structure and linked to each other by edges (not making a cycle).



Important Terminologies In Tree Data Structure

- Node: It is the basic unit of a data structure and contains data that may link to another node. A node may contain a value or condition or represent a separate data structure.
- Root: Root is the topmost node of the tree. A tree can have only one root.
- Edge: It is the connecting link between any two nodes. For "n" nodes, you can have "n-1" edges.
- Path: It is a sequence of nodes along the edges or corners of a tree.

- Parent: In the tree data structure, a node that is a predecessor of any node is the parent node.
- Child: A node that branches from a particular node is the child of that node. In the data structure, every node except the root node is a child node and a parent node can have any number of child nodes.
- Sibling: Siblings are the nodes that belong to the same parent.
- Leaf: Leaf is a node having no child node. In the tree data structure, the leaf is the terminal or external node.
- Internal node: Internal nodes are nodes having at least one child node. All nodes other than leaf are internal nodes and when a tree has over one node, the root node becomes the internal node of that tree.the

- Degree: Degree represents the total number of children of a node. Nodes with zero degrees are leaf or terminal nodes.
- Sub-tree: These are the descendants of a node.
- Levels: In the tree data structure, the root node is at level zero, the children of the root node are at level one and the children of level one node are at level two.

Basic Operation of Tree Data Structure

- Insertion
- Deletion
- Searching
- Traversing
- Updating

Advantages of Trees in Data Structure

- **Efficient searching:** Trees are particularly efficient for searching and retrieving data. The time complexity of searching in a tree is typically O(log n), which means that it is very fast even for very large data sets.
- **Flexible size:** Trees can grow or shrink dynamically depending on the number of nodes that are added or removed. This makes them particularly useful for applications where the data size may change over time.
- **Easy to traverse:** Traversing a tree is a simple operation, and it can be done in several different ways depending on the requirements of the application. This makes it easy to retrieve and process data from a tree structure.

- **Easy to maintain:** Trees are easy to maintain because they enforce a strict hierarchy and relationship between nodes. This makes it easy to add, remove, or modify nodes without affecting the rest of the tree structure.
- **Fast insertion and deletion**: Inserting and deleting nodes in a tree can be done in O(log n) time, which means that it is very fast even for very large trees.

Disadvantages of Tree:

- **Memory overhead**: Trees can require a significant amount of memory to store, especially if they are very large. This can be a problem for applications that have limited memory resources.
- **Imbalanced trees:** If a tree is not balanced, it can result in uneven search times. This can be a problem in applications where speed is critical.
- **Complexity**: Trees can be complex data structures, and they can be difficult to understand and implement correctly. This can be a problem for developers who are not familiar with them.

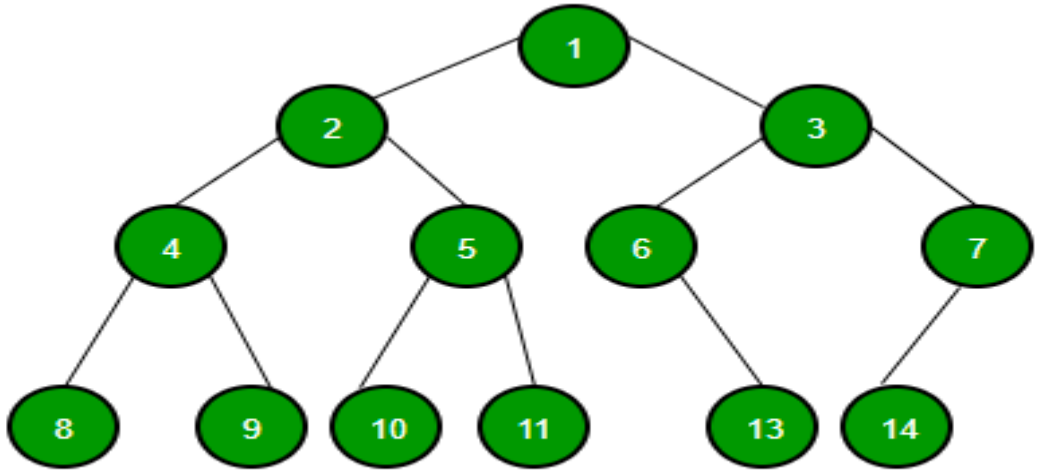
- **Limited flexibility**: While trees are flexible in terms of size and structure, they are not as flexible as other data structures like hash tables. This can be a problem in applications where the data size may change frequently.
- **Inefficient for certain operations:** While trees are very efficient for searching and retrieving data, they are not as efficient for other operations like sorting or grouping. For these types of operations, other data structures may be more appropriate.

Types of Trees

- General tree
- Binary tree
- Binary search tree
- AVL tree
- Red-black tree
- Splay tree
- Treap
- B-tree

Binary Tree Data Structure

A tree whose elements have at most 2 children is called a binary tree. Since each element in a binary tree can have only 2 children, we typically name them the left and right children.



Basic Operations On Binary Tree:

- Inserting an element.
- Removing an element.
- Searching for an element.
- Traversing an element.

There are four (mainly three) types of traversals in a binary tree which will be discussed ahead.

Auxiliary Operations On Binary Tree:

- Finding the height of the tree
- Find the level of the tree
- Finding the size of the entire tree.

Binary Tree Traversals:

Tree Traversal algorithms can be classified broadly into two categories:

- Depth-First Search (DFS) Algorithms
- Breadth-First Search (BFS) Algorithms

Tree Traversal using Depth-First Search (DFS) algorithm can be further classified into three categories:

- Preorder Traversal (current-left-right)
- Inorder Traversal (left-current-right)
- Postorder Traversal (left-right-current)

Preorder Traversal (current-left-right): Visit the current node before visiting any nodes inside the left or right subtrees. Here, the traversal is root – left child – right child. It means that the root node is traversed first then its left child and finally the right child.

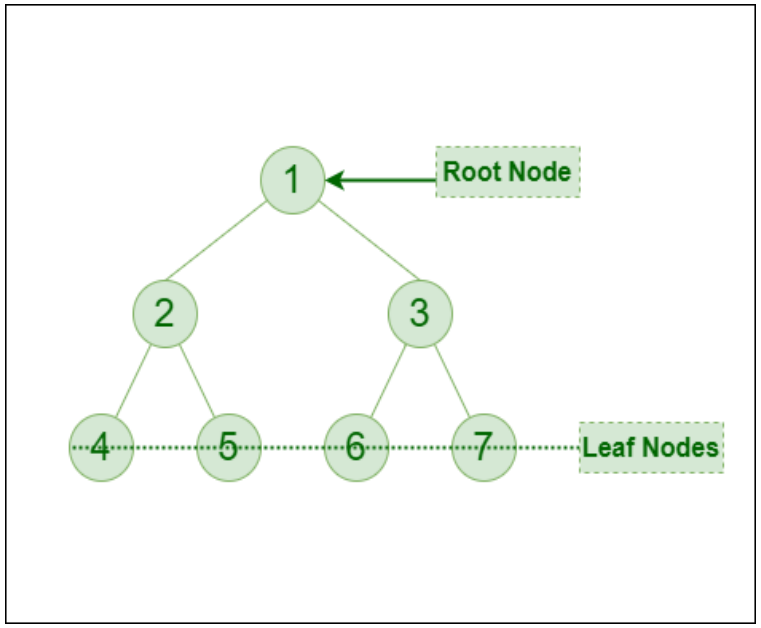
Inorder Traversal (left-current-right): Visit the current node after visiting all nodes inside the left subtree but before visiting any node within the right subtree. Here, the traversal is left child – root – right child. It means that the left child is traversed first then its root node and finally the right child.

Postorder Traversal (left-right-current): Visit the current node after visiting all the nodes of the left and right subtrees. Here, the traversal is left child – right child – root. It means that the left child has traversed first then the right child and finally its root node.

Tree Traversal using Breadth-First Search (BFS) algorithm can be further classified into one category:

Level Order Traversal: Visit nodes level-by-level and left-to-right fashion at the same level. Here, the traversal is level-wise. It means that the most left child has traversed first and then the other children of the same level from left to right have traversed. The

Let us traverse the following tree with all four traversal methods:



Pre-order Traversal : 1-2-4-5-3-6-7

In-order Traversal : 4-2-5-1-6-3-7

Post-orderTraversal:4-5-2-6-7-3-1

Level-order Traversal : 1-2-3-4-5-6-7

Binary tree implementation

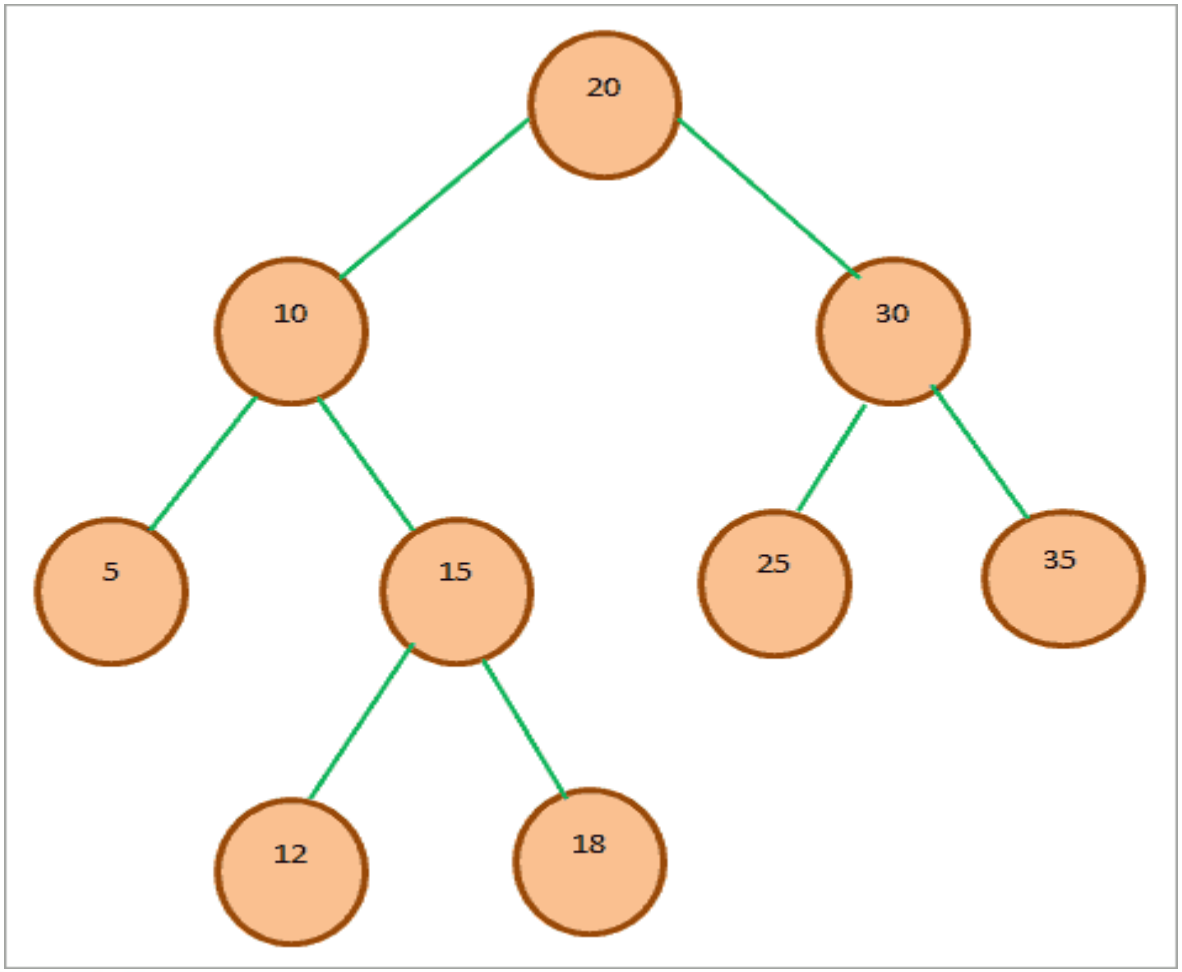
```
class Node
{
    int key;
    Node left, right;
    public Node(int item)
    {
        key = item;
        left = right = null;
    }
}
public class BinaryTree
{
    Node root;
    // Traverse tree
    public void traverseTree(Node node)
    {
        if (node != null)
        {
            traverseTree(node.left);
```

```
System.out.print(" " + node.key);
traverseTree(node.right);
}
}
public static void main(String[] args) {


// create an object of BinaryTree
BinaryTree tree = new BinaryTree();


// create nodes of the tree
tree.root = new Node(1);
tree.root.left = new Node(2);
tree.root.right = new Node(3);
tree.root.left.left = new Node(4);
System.out.print("\nBinary Tree: ");
tree.traverseTree(tree.root);
}
}
```

Binary Search Tree

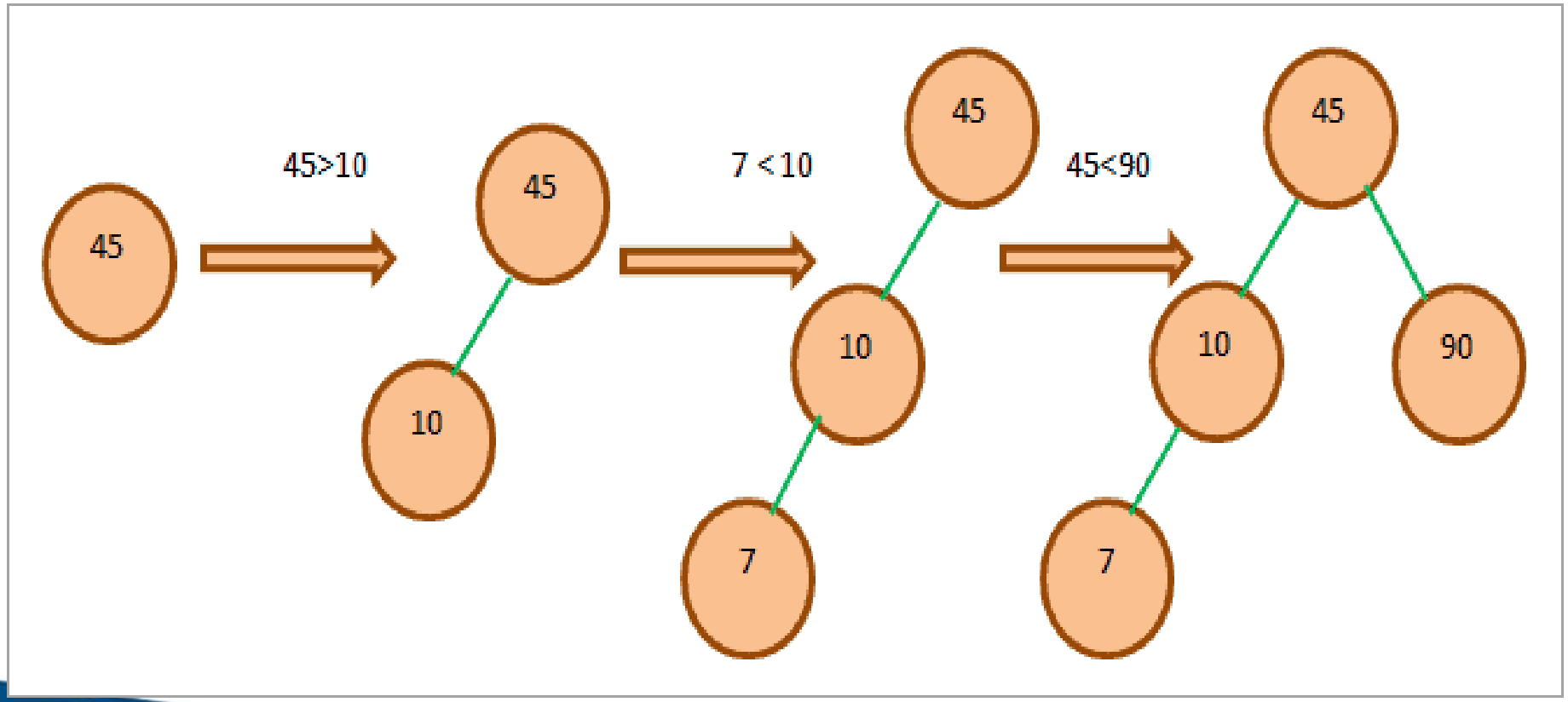
- A Binary search tree (referred to as BST hereafter) is a type of binary tree.
- It can also be defined as a node-based binary tree or ‘Ordered Binary Tree’.
- In BST, all the nodes in the left subtree have values that are less than the value of the root node.
- All the nodes of the right subtree of the BST have values that are greater than the value of the root node.
- This ordering of nodes has to be true for respective subtrees as well.

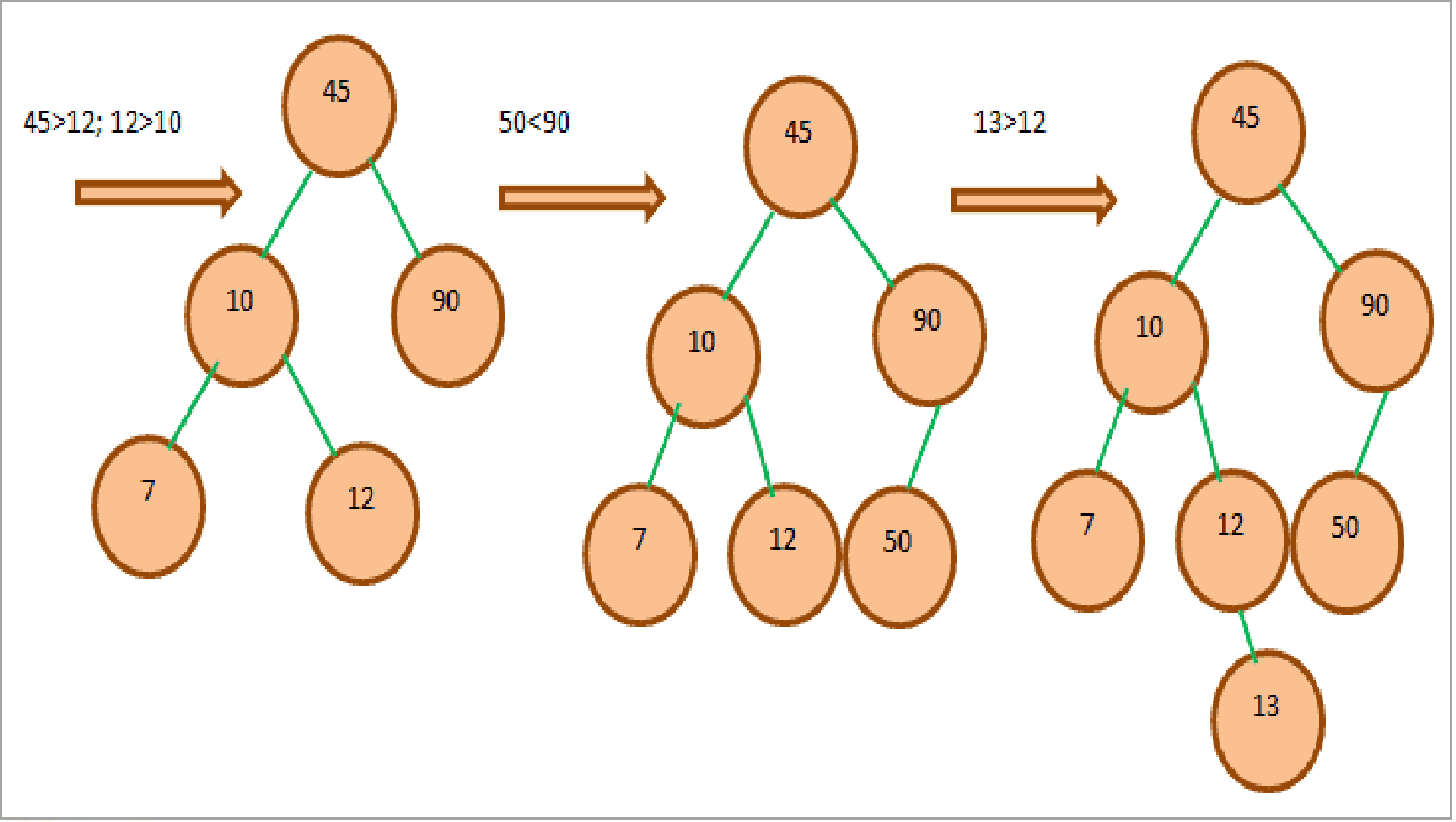
- A BST does not allow duplicate nodes.

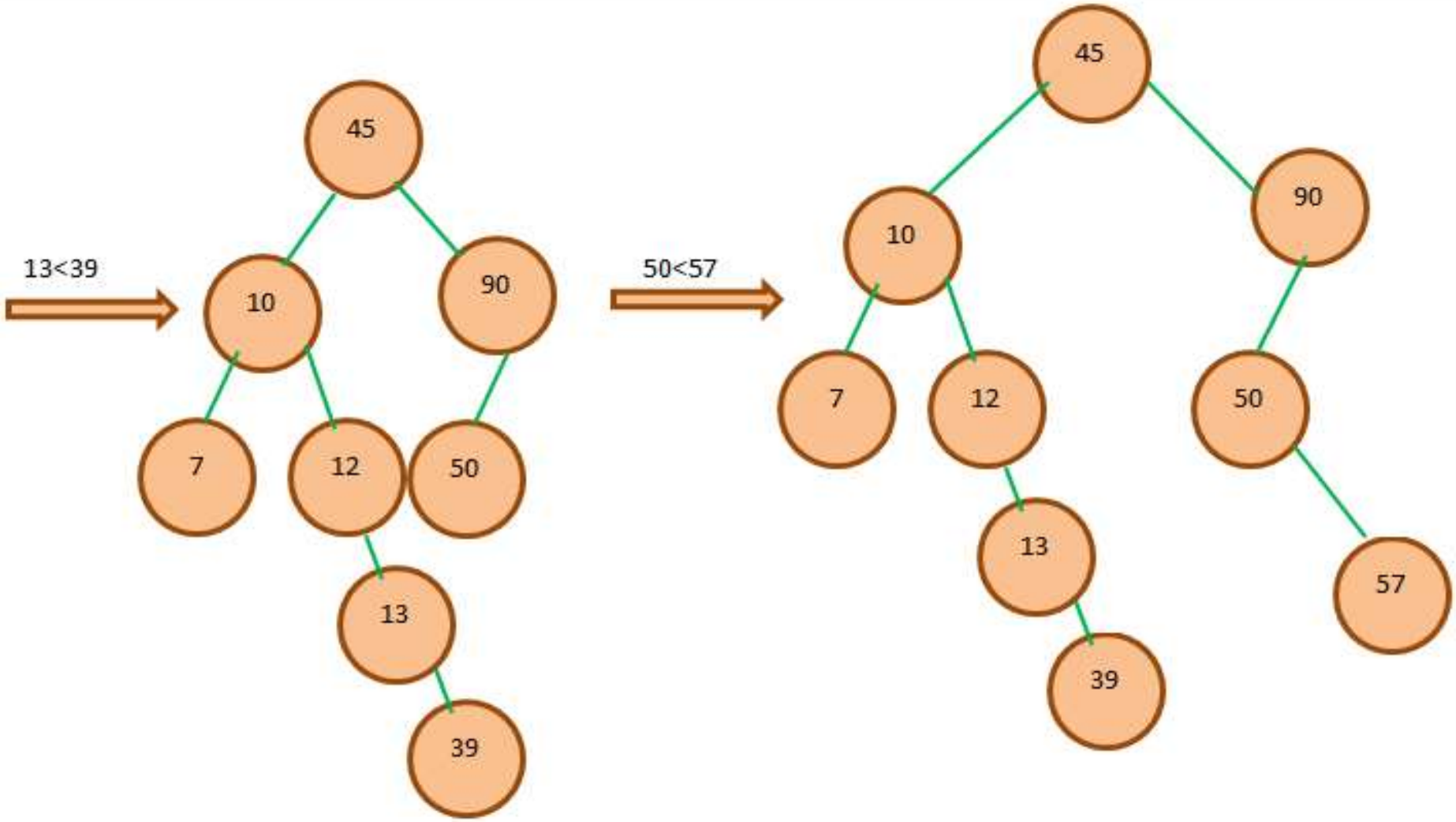


Creating A Binary Search Tree (BST)

Given array: 45, 10, 7, 90, 12, 50, 13, 39, 57




45>12; 12>10
45
10
90
7
12
50<90
45
10
90
7
12
50
13>12
45
10
90
7
12
50
13


13<39
45
10
90
7
12
50
13
39
50<57
45
10
90
7
12
50
13
57
39

Insert An Element In BST

An element is always inserted as a leaf node in BST.

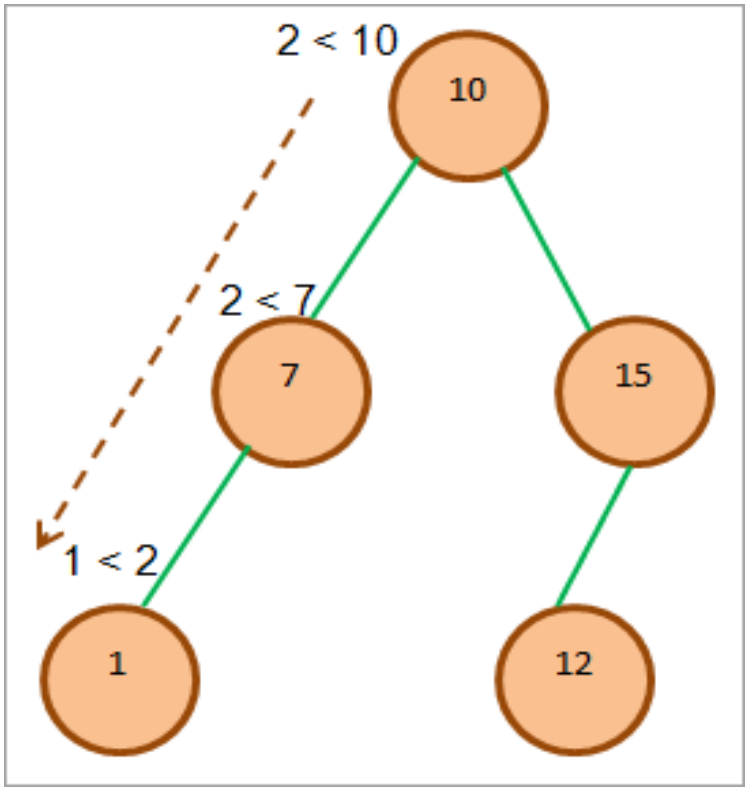
Given below are the steps for inserting an element.

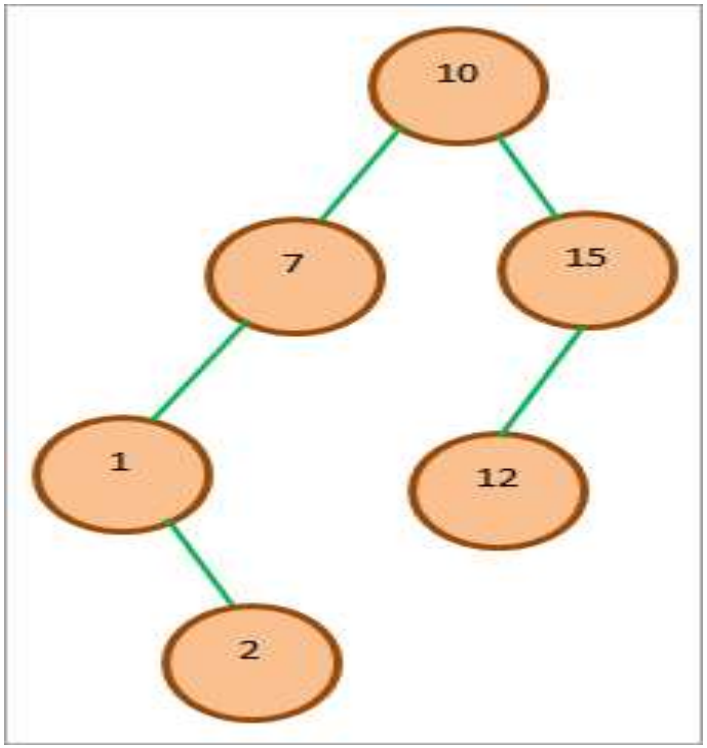
1.Start from the root.

2.Compare the element to be inserted with the root node. If it is less than root, then traverse the left subtree or traverse the right subtree.

3.Traverse the subtree till the end of the desired subtree. Insert the node in the appropriate subtree as a leaf node.

Consider the following BST and let us insert element 2 in the tree.





Search Operation In BST

Enlisted below are the steps that we have to follow.

1.Compare the element to be searched with the root node.

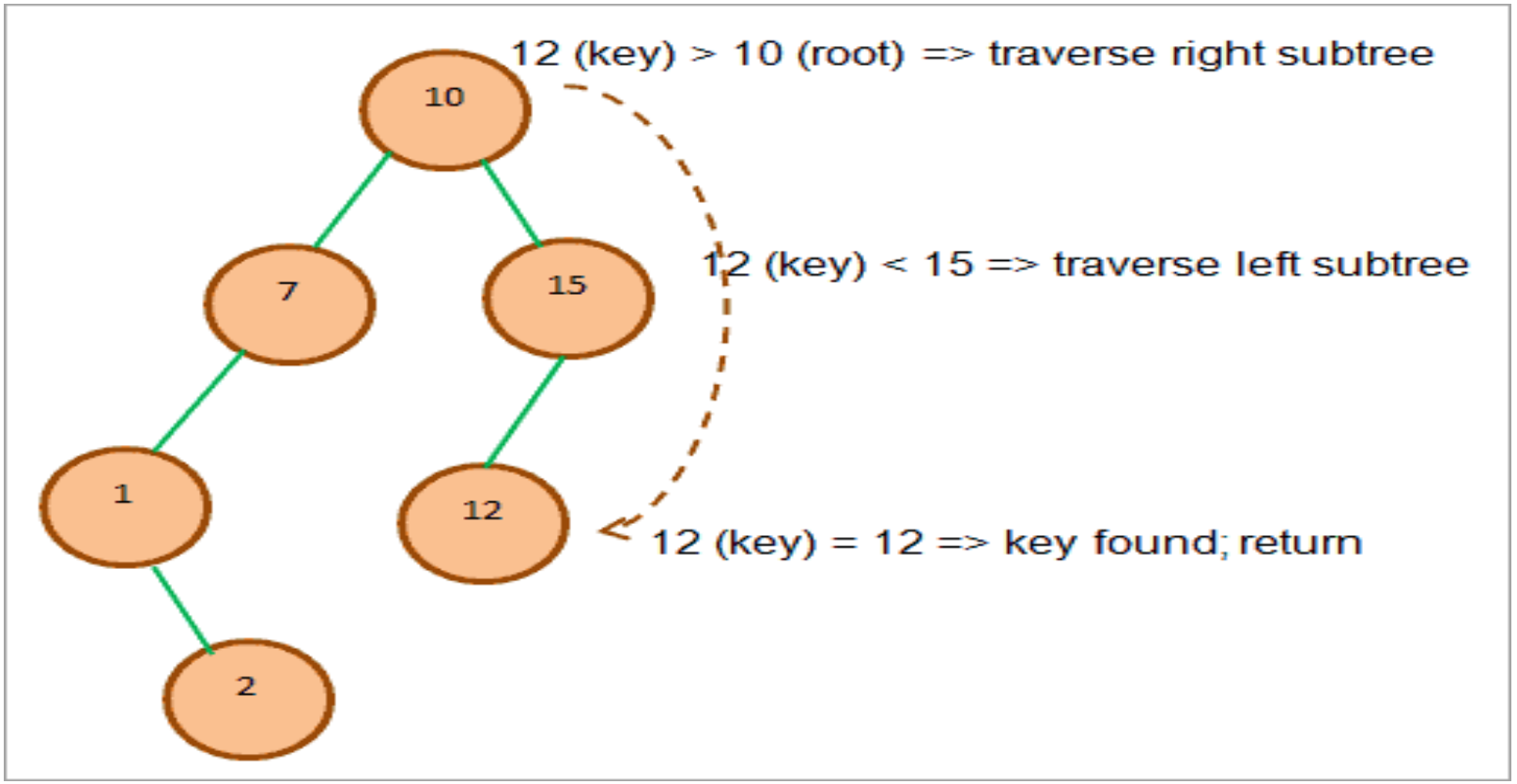
2.If the key (element to be searched) = root, return root node.

3.Else if key < root, traverse the left subtree.

4.Else traverse right subtree.

5.Repetitively compare subtree elements until the key is found or the end of the tree is reached.

Let’s illustrate the search operation with an example. Consider that we have to search the key = 12.

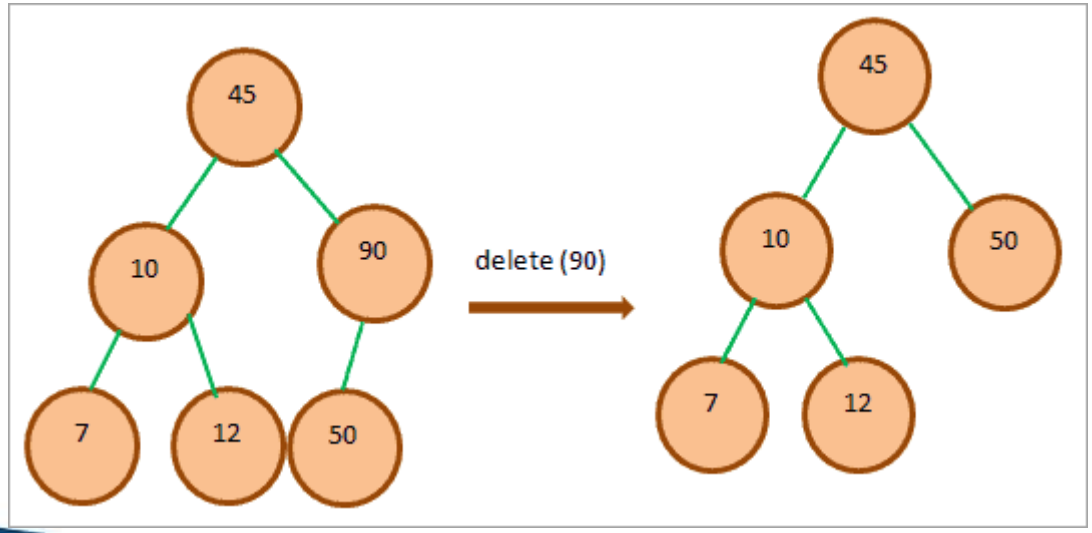


Remove Element From The BST

When we delete a node from the BST, then there are three possibilities

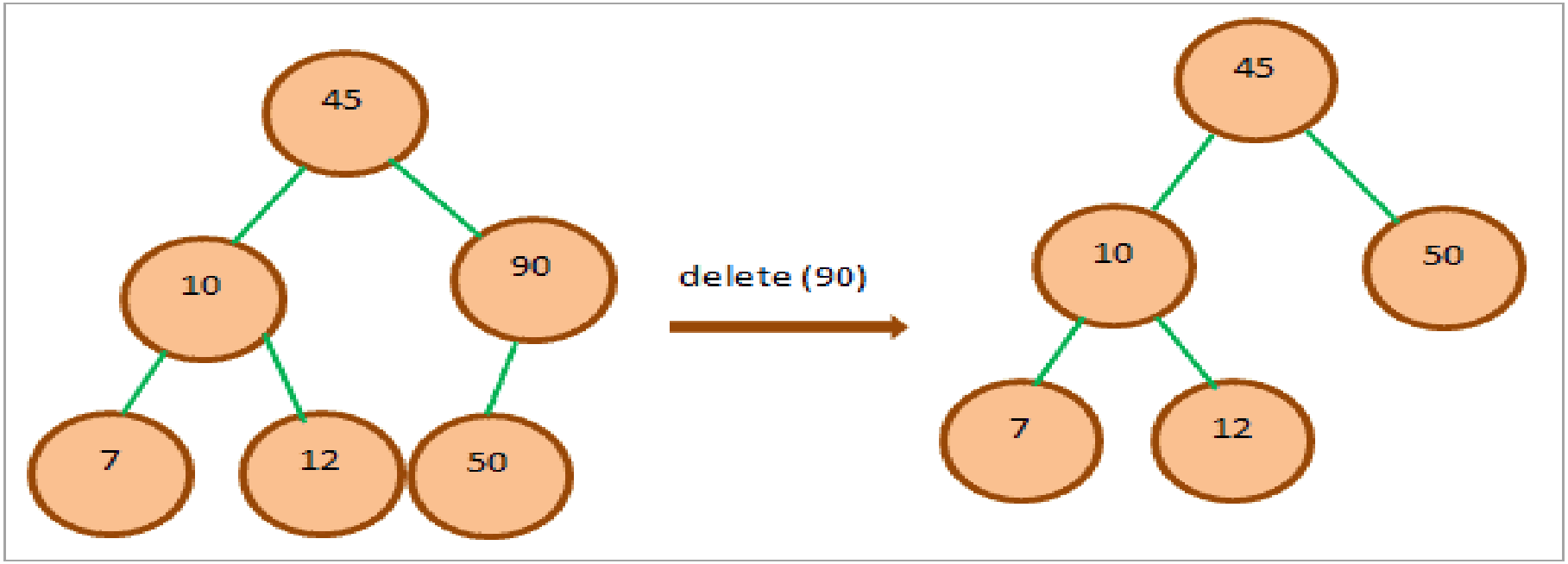
Node Is A Leaf Node

If a node to be deleted is a leaf node, then we can directly delete this node as it has no child nodes. This is shown in the below image.



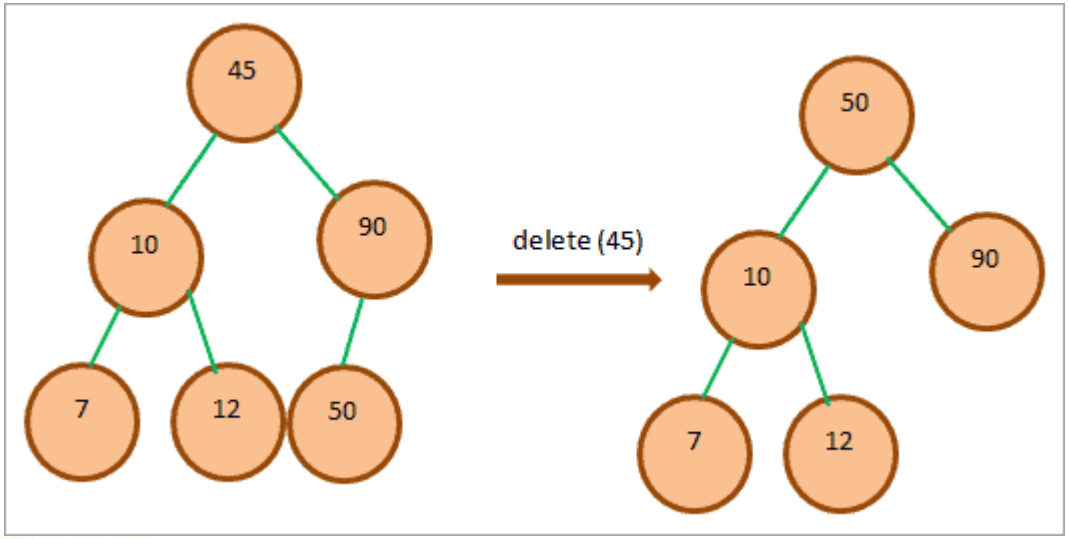
Node Has Only One Child

When we need to delete the node that has one child, then we copy the value of the child in the node and then delete the child.



Node Has Two Children

When a node to be deleted has two children, then we replace the node with the inorder (left-root-right) successor of the node or simply said the minimum node in the right subtree if the right subtree of the node is not empty. We replace the node with this minimum node and delete the node.



Binary Search Tree (BST) Traversal In Java

- Inorder Traversal
- Preorder Traversal
- PostOrder Traversal

All the above traversals use depth-first technique i.e. the tree is traversed depth wise.

Inorder Traversal

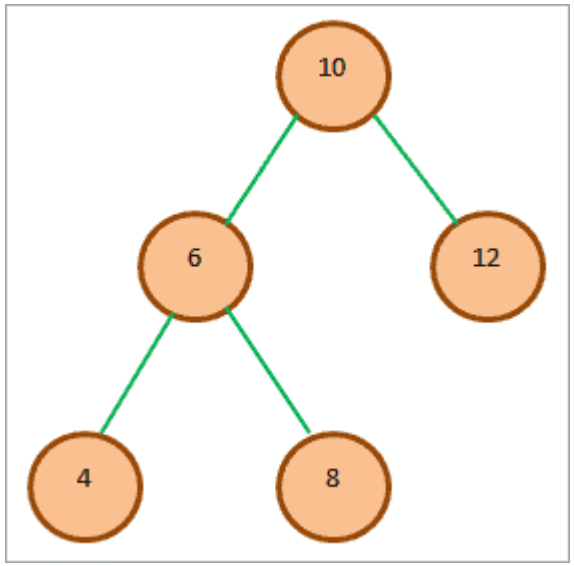
The inorder traversal approach traversed the BST in the order, **Leftsubtree=>RootNode=>Right subtree**.

The algorithm InOrder (bstTree) for InOrder Traversal is given below.

1.Traverse the left subtree using InOrder (left_subtree)

2.Visit the root node.

3.Traverse the right subtree using InOrder (right_subtree)



The inorder traversal of the tree is:

4 6 8 10 12

Preorder Traversal

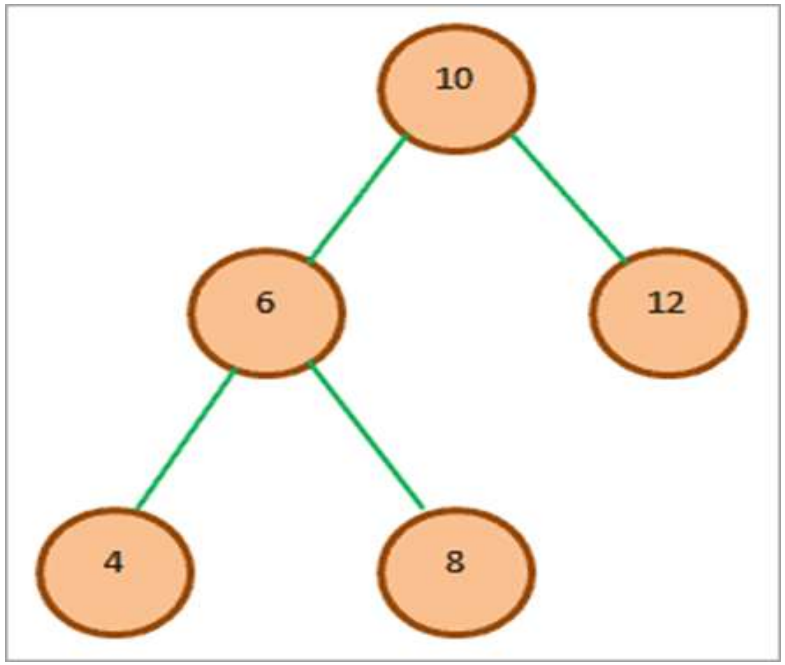
In preorder traversal, the root is visited first followed by the left subtree and right subtree.

The algorithm for PreOrder (bst_tree) traversal is given below:

1.Visit the root node

2.Traverse the left subtree with PreOrder (left_subtree).

3.Traverse the right subtree with PreOrder (right_subtree).



The preorder traversal for the BST given above is:

10 6 4 8 12

PostOrder Traversal

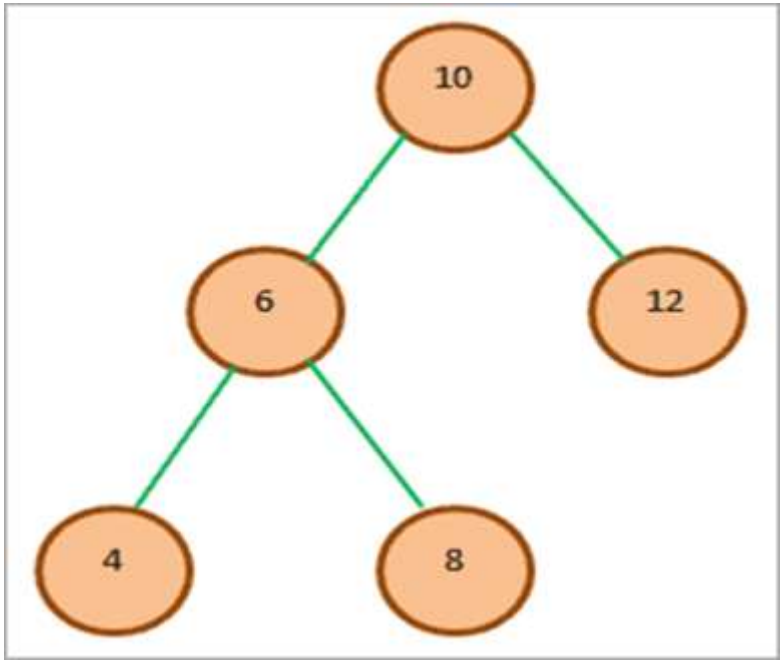
The postOrder traversal traverses the BST in the order: **Left subtree->Right subtree->Root node**.Postorder

The algorithm for postOrder (bst_tree) traversal is as follows:

1.Traverse the left subtree with postOrder (left_subtree).

2.Traverse the right subtree with postOrder (right_subtree).

3.Visit the root node



The postOrder traversal for the above example BST is:

4 8 6 12 10