

Tree Data Structure

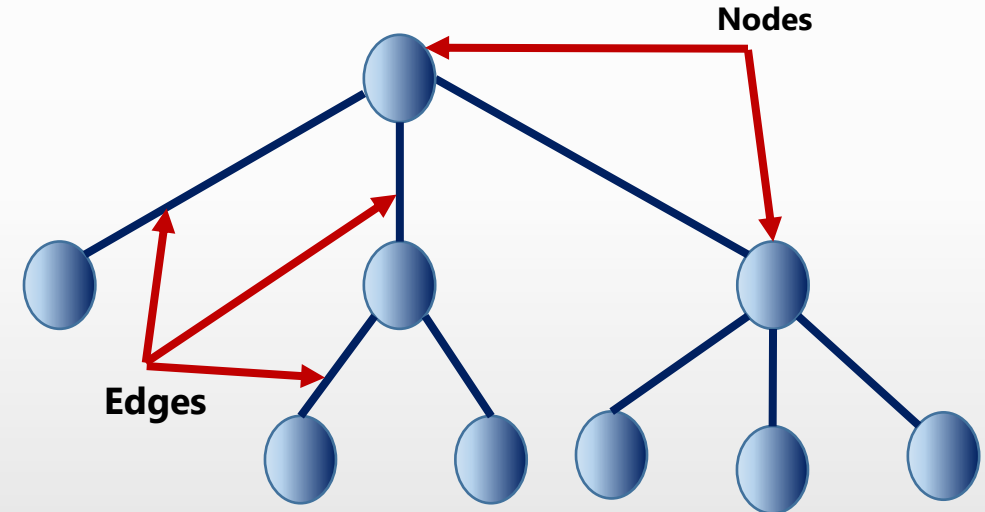
What is a tree?

A tree consist of nodes connected by edges.

In a picture of a tree the nodes are represented as circles and the edges as lines connecting the circles.

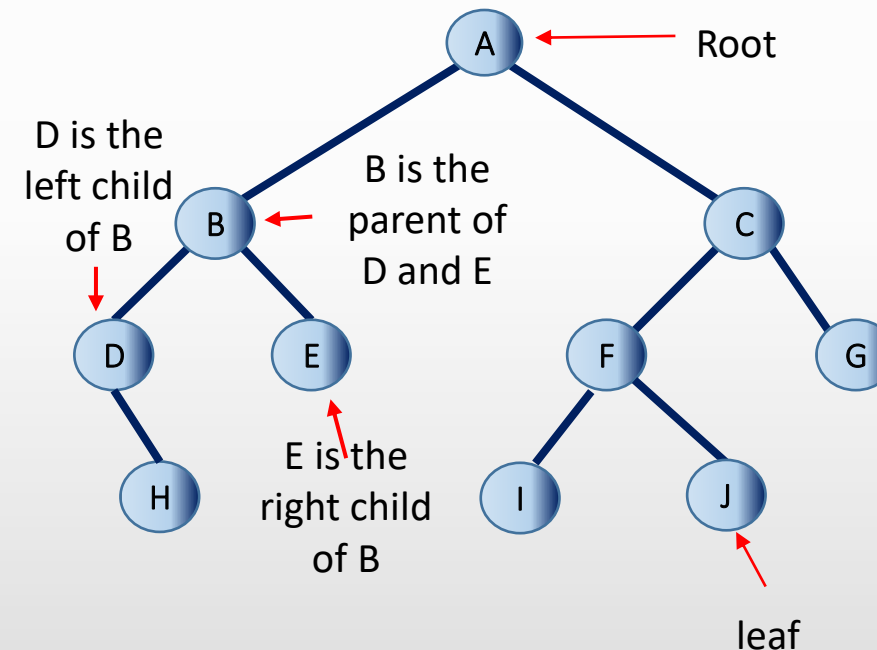
In a tree, the nodes represent the data items and the edges represent the way the nodes are related.

A tree with nodes which has maximum of two children is called a **binary tree**.



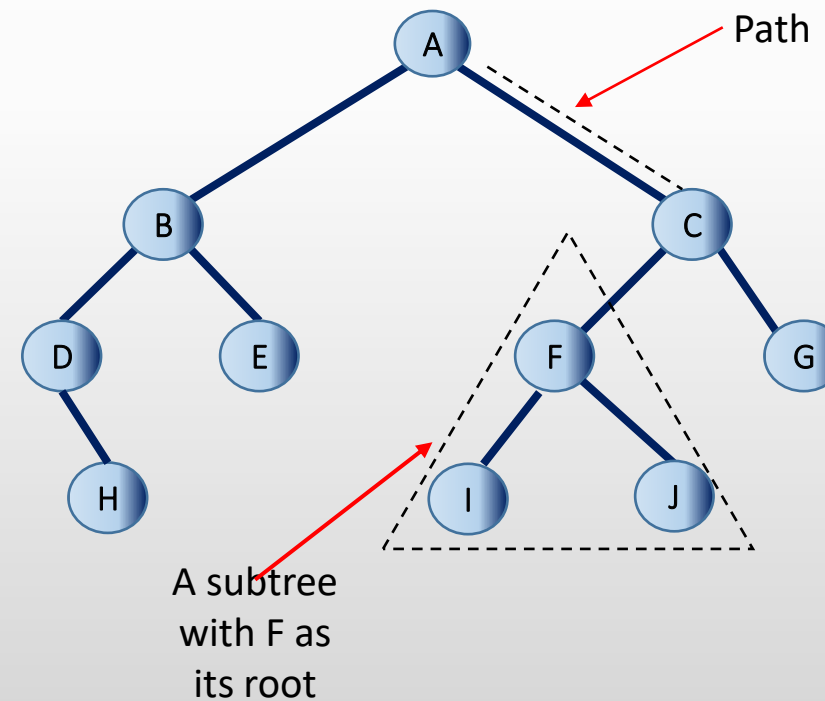
What is root, parent and children in a tree?

- The node at the top of the tree is called root.
- Any node which has exactly one edge running upwards to other node is called a child
- The two children of each node in a binary tree is called left child and right child.
- Any node which has one or more lines running downwards to other nodes is called a parent
- A node that has no children is called a leaf node or leaf



What is path and subtree in a tree

- Sequence of nodes from one node to another along the edges is called a path.
- Any node which consist of its children and it's children's children and so on is called a sub tree



Key value of a node

- Each node in a tree stores objects containing information.
- Therefore one data item is usually designated as a key value.
- The key value is used to search for a item or to perform other operations on it.
- eg: person object – social security number
car parts object– part number

Binary Search Tree

- Binary Search Tree

- is a tree that has at most two children.

- a node's left child must have a key less than its parent and node's right child must have a key greater than or equal to its parent.

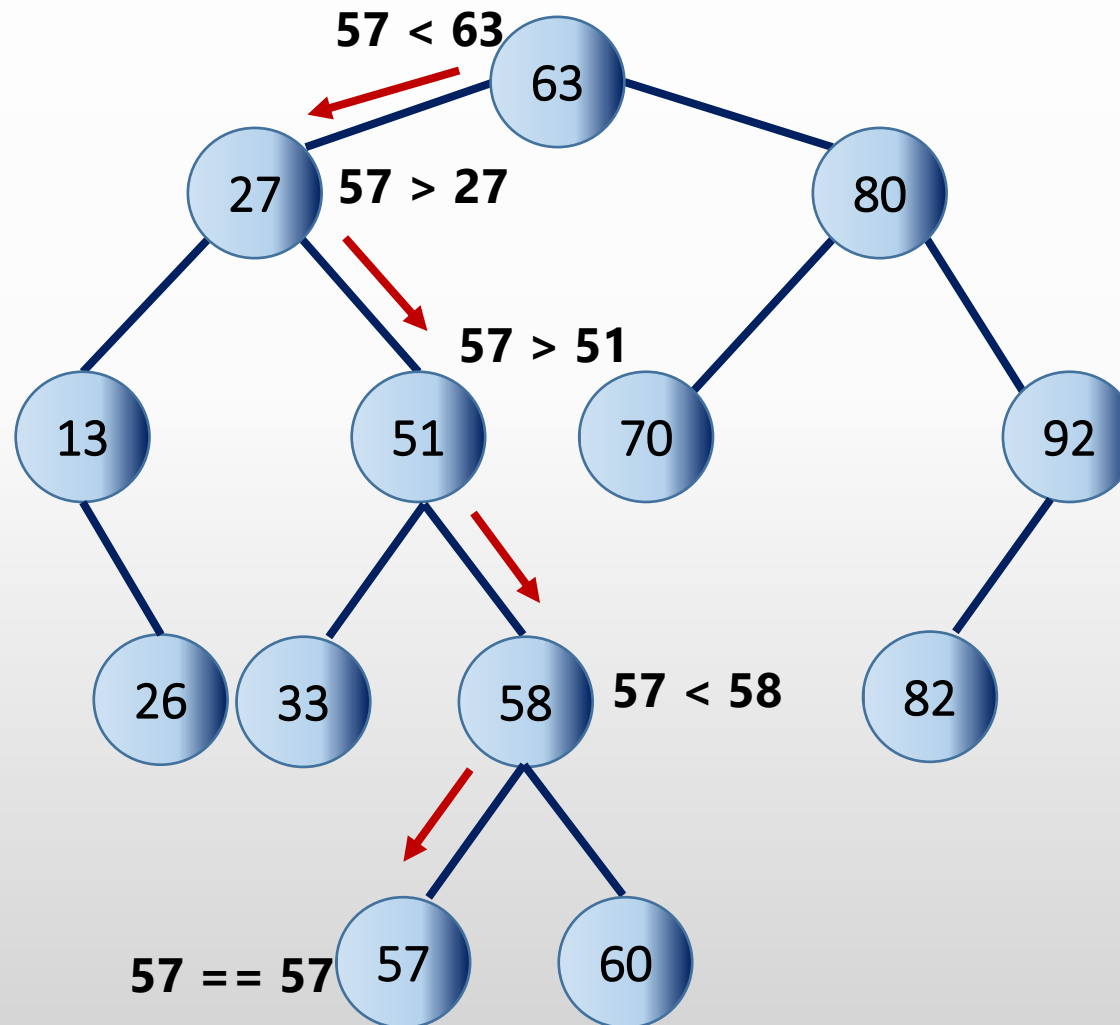
Operations of Binary Search Tree

- There are four main operations perform in a binary search tree
 - Find – find a node with a given key
 - Insert – insert a new node
 - Delete – delete a node
 - Traverse – visit all the nodes

Operations - Find

- Find always start at the root.
- Compare the key value with the value at root.
- If the key value is less, then compare with the value at the left child of root.
- If the key value is higher, then compare with the value at the right child of root
- Repeat this, until the key value is found or reach to a leaf node.

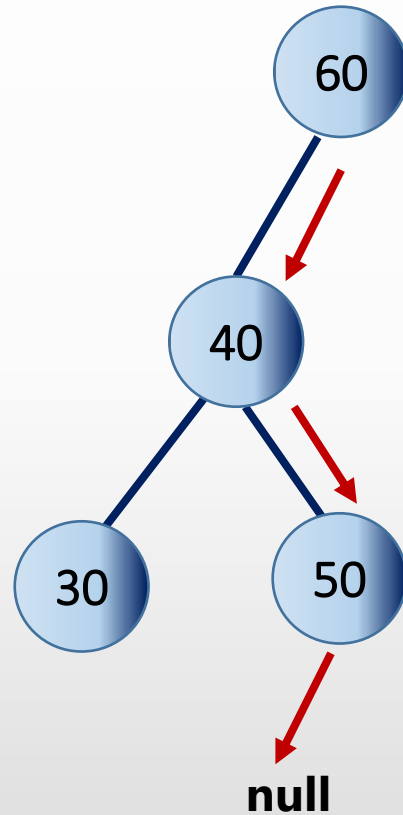
Find value 57



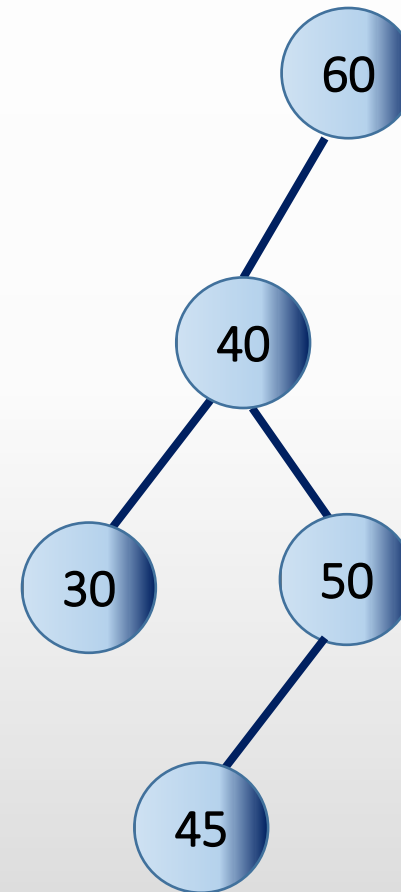
Operations - Insert

- Create a new node.
- Find the place (parent) to insert a new node.
- When the parent is found, the new node is connected as its left or right child, depending on whether the new node's key is less than or greater than that of the parent.

Insert value 45



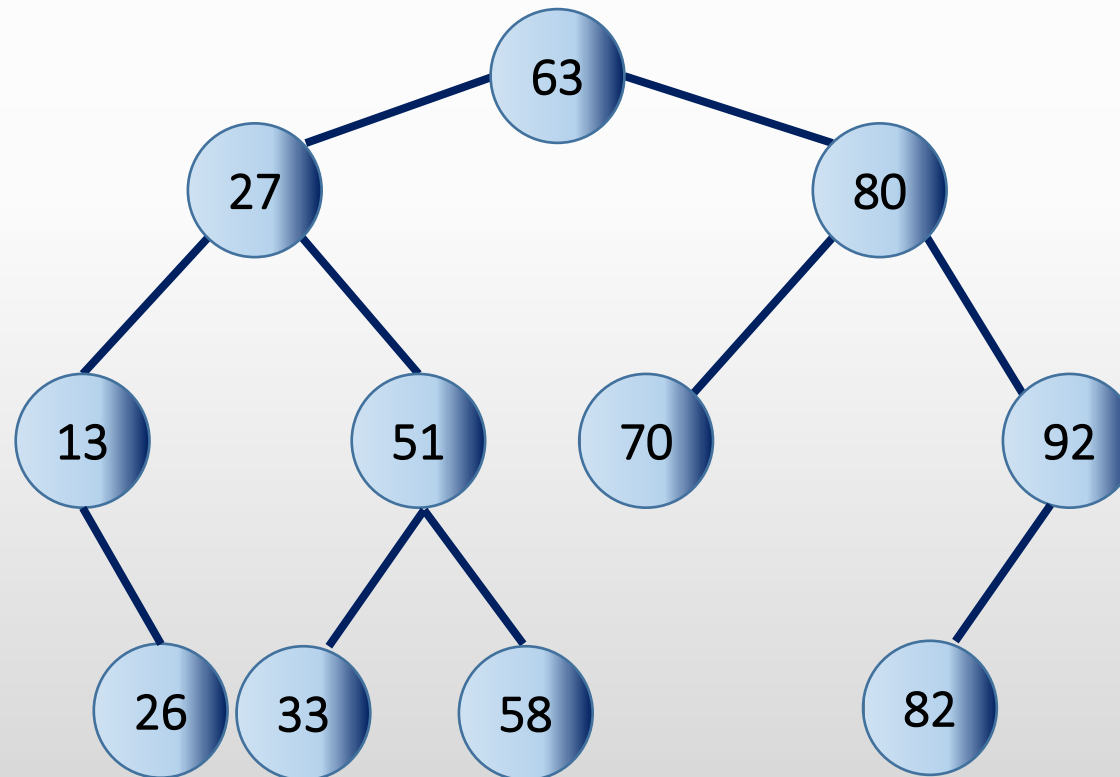
a) Before insertion



b) After insertion

Question 1

- Draw a tree after inserting number 8



BFS vs DFS for Binary Tree

A Tree is typically traversed in two ways:

- Breadth First Traversal (Or Level Order Traversal)
- Depth First Traversals
 1. Inorder Traversal (Left-Root-Right)
 2. Preorder Traversal (Root-Left-Right)
 3. Postorder Traversal (Left-Right-Root)

BFS and DFSs of above Tree

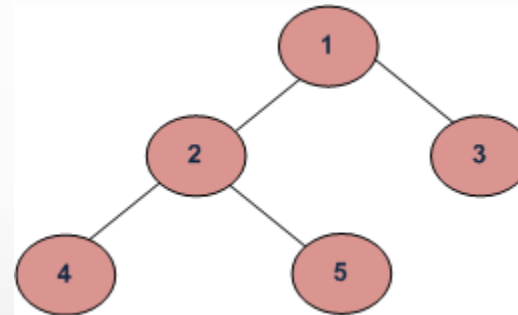
Breadth First Traversal : 1 2 3 4 5

Depth First Traversals:

Preorder Traversal : 1 2 4 5 3

Inorder Traversal : 4 2 5 1 3

Postorder Traversal : 4 5 2 3 1



Traversing

- Traverse a tree means to visit all the nodes in some specified order.
- There are three common ways to traverse a tree
 - Pre order
 - In order
 - Post order

InOrder(root) visits nodes in the following order:

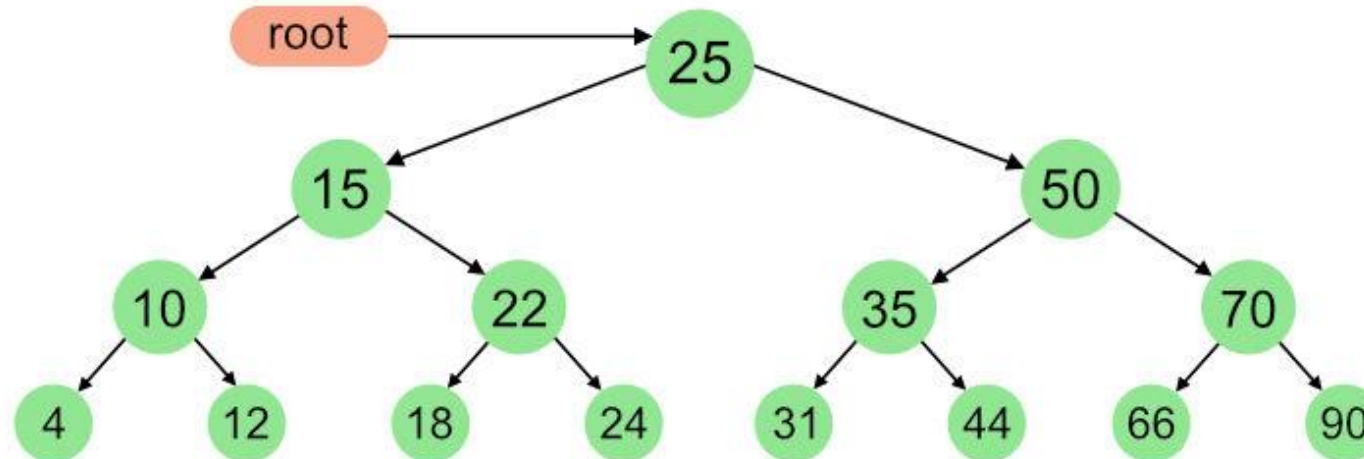
4, 10, 12, 15, 18, 22, 24, 25, 31, 35, 44, 50, 66, 70, 90

A Pre-order traversal visits nodes in the following order:

25, 15, 10, 4, 12, 22, 18, 24, 50, 35, 31, 44, 70, 66, 90, 25

A Post-order traversal visits nodes in the following order:

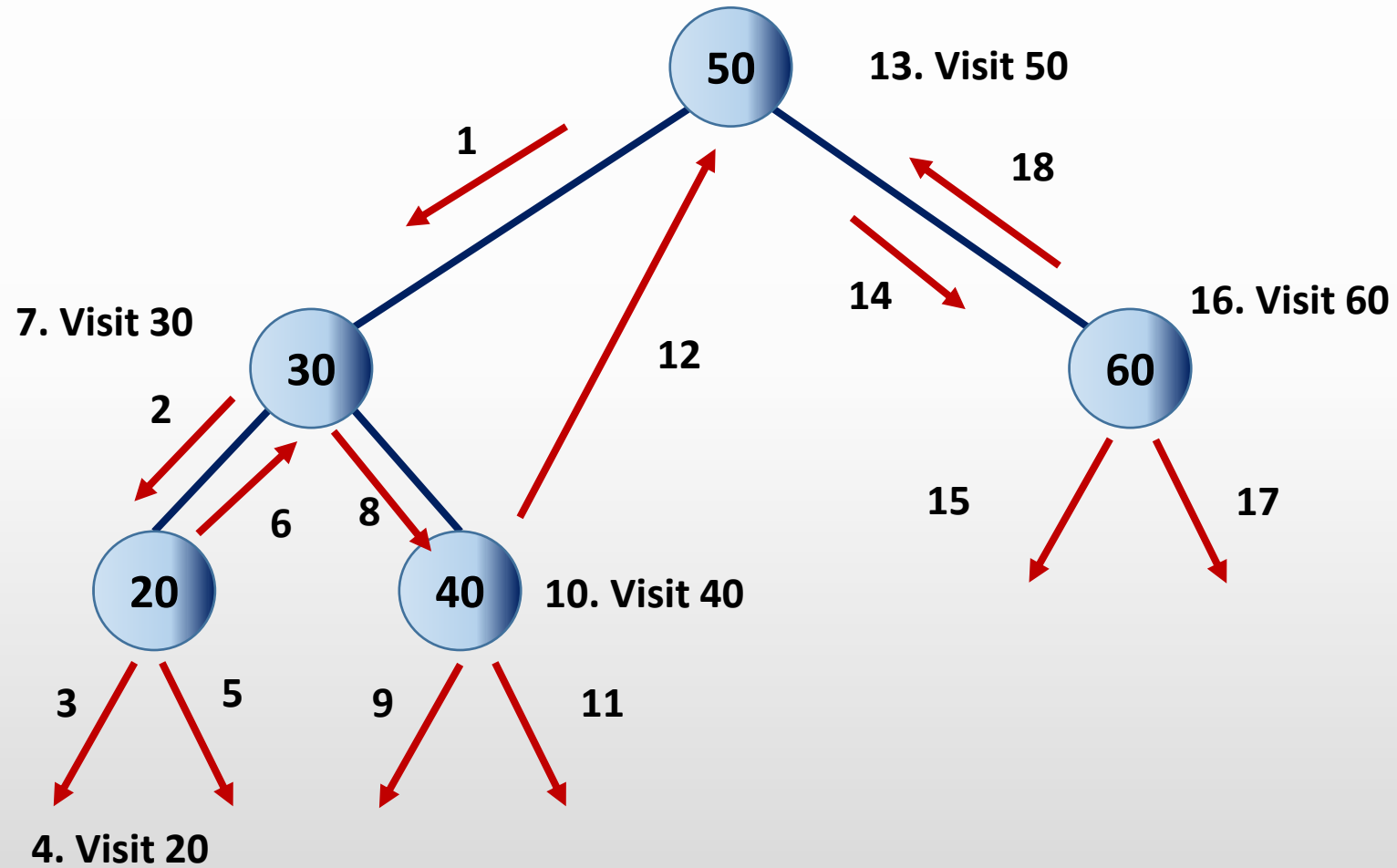
4, 12, 10, 18, 24, 22, 15, 31, 44, 35, 66, 90, 70, 50, 25



Inorder traversing

- Call itself to traverse the node's left subtree
- Visit the node
- Call itself to traverse the node's right subtree

Inorder traversing cont...

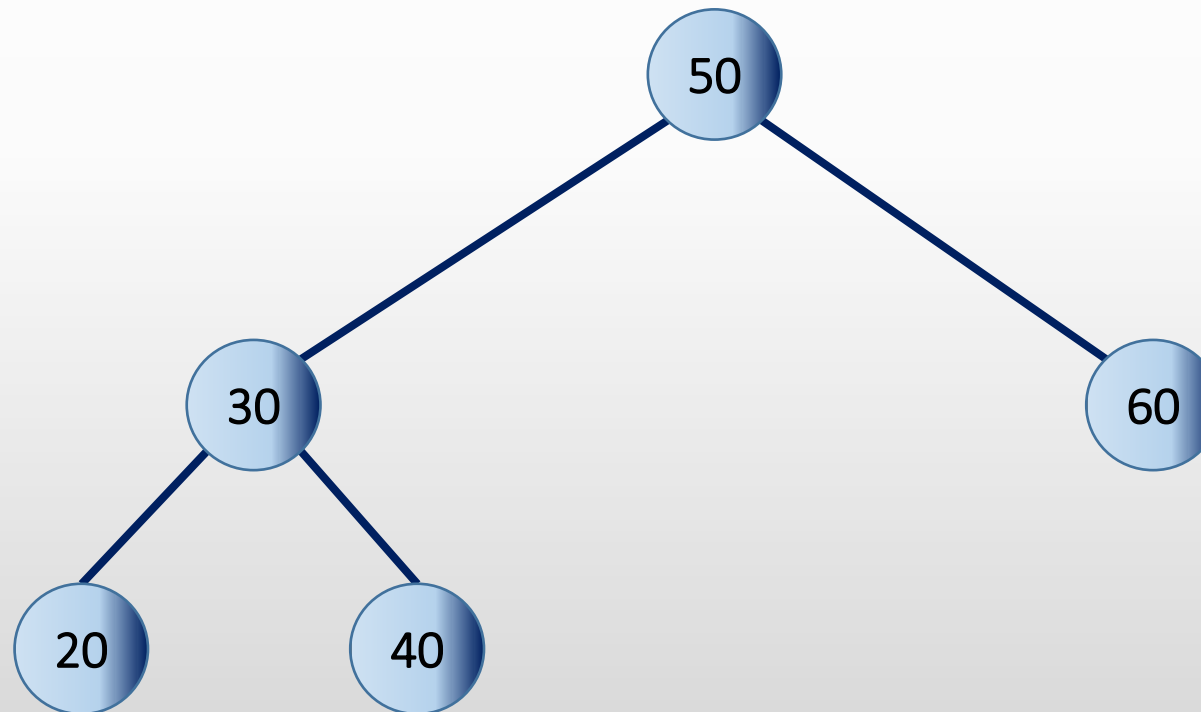


Preorder traversing

- Visit the node
- Call itself to traverse the node's left subtree
- Call itself to traverse the node's right subtree

Question 2

- Write the output, if the following tree is traverse in preorder.

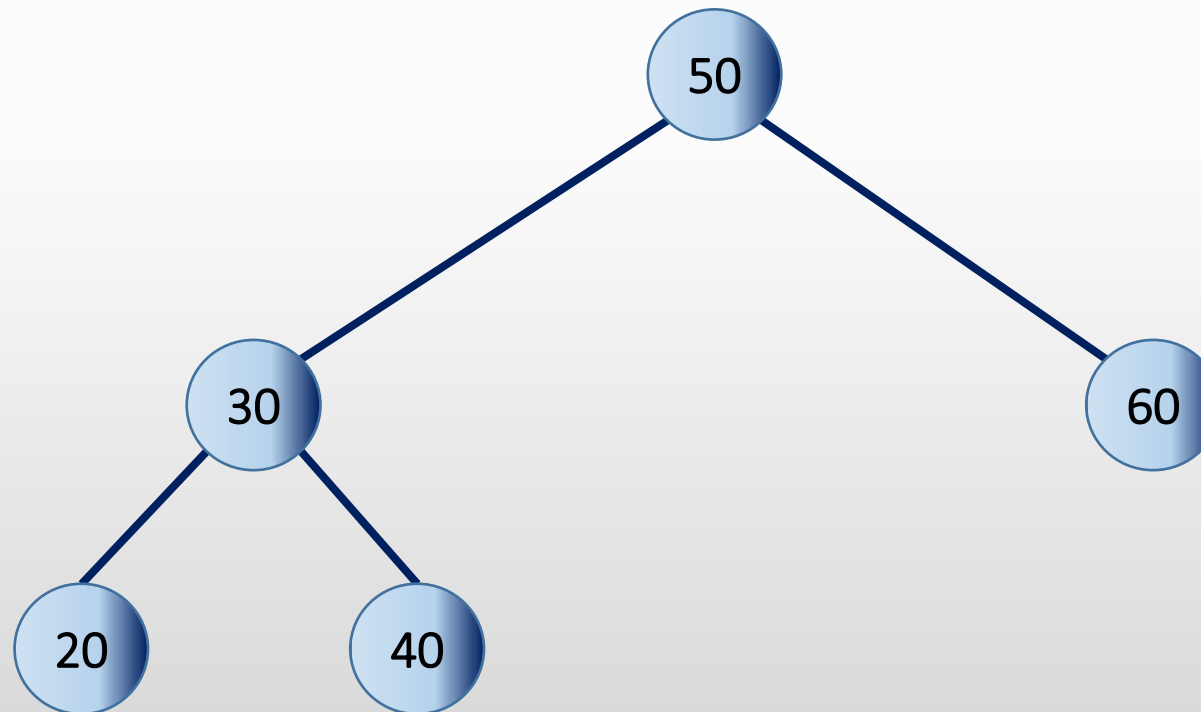


Postorder traversing

- Call itself to traverse the node's left subtree
- Call itself to traverse the node's right subtree
- Visit the node

Question 3

- Write the output, if the following tree is traverse in postorder.



Binary search tree - Implementation

Node Class

Node contains the information about an object.

Each node should have a key, data and reference to left and right child.

```
class Node
{
    public int iData; // data item (used as key value)
    public double dData; //other data
    public Node leftChild; // this node's left child
    public Node rightChild; //this node's right child

    public void displayNode( ){
        System.out.print("{");
        System.out.print(iData);
        System.out.print(", ");
        System.out.print(dData);
        System.out.print( " }");
    }
}
```

Binary search tree - Implementation

Tree Class

```
class Tree
{
    private Node root; // first node of tree

    public Tree(){
        root = null;
    }
    public void insert (int id, double dd){
    }
    public boolean delete (int id){
    }
    public Node find (int key){
    }
}
```


Binary search tree - Implementation

Tree Class – Find Method

```
class Tree
{
    private Node root;
    public Tree(){
        root = null;
    }
    public void insert (int id, double dd){
    }
    public void delete (int id){
    }
    public Node find(int key){
        Node current = root;
        while (current.iData != key)
        {
            if(key < current.iData)
                current = current.leftChild;
            else
                current = current.rightChild;
            if (current == null)
                return null;
        }
        return current;
    }
}
```

Binary search tree - Implementation

```

class Tree{
    private Node root;
    .....
    .....

    public void insert ( int id , double dd){
        Node newNode = new Node();
        newNode.iData = id;
        newNode.dData = dd;
        if (root == null) // no node in root
            root = newNode;
        else // root occupied
        {
            Node current = root; //start at root
            Node parent;
            while (true)
            {
                parent = current;
            }
        }
    }
}
    
```

```

        if (id < current.iData) // go left
        {
            current = current.leftChild;
            if (current == null) {
                parent.leftChild = newNode;
                return;
            }
        }
        else // go right
        {
            current = current.rightChild;
            if (current == null){
                parent.rightChild = newNode;
                return;
            }
        }
    }
}
}
}
    
```

Binary search tree - Implementation

Tree Class – Inorder Traversing Method

```
private void inOrder(Node localRoot)
{
    if (localRoot != null)
    {
        inOrder(localRoot.leftChild);
        localRoot.displayNode();
        inOrder(localRoot.rightChild);
    }
}
```

Binary search tree - Implementation

Tree Class – Preorder Traversing Method

```
private void preOrder(Node localRoot)
{
    if (localRoot != null)
    {
        localRoot.displayNode();
        preOrder(localRoot.leftChild);
        preOrder(localRoot.rightChild);
    }
}
```

Binary search tree - Implementation

Tree Class – Postorder Traversing Method

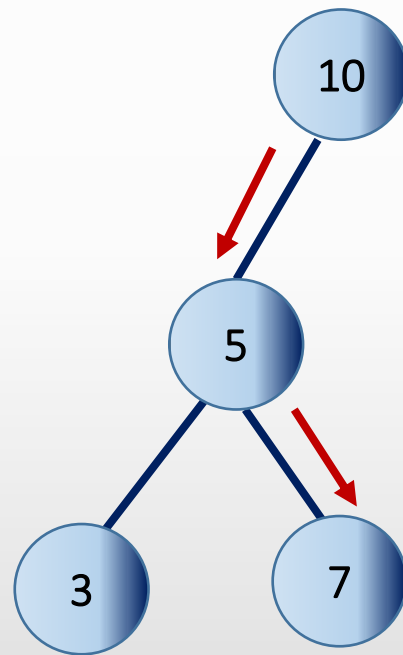
```
private void postOrder(Node localRoot)
{
    if (localRoot != null)
    {
        postOrder(localRoot.leftChild);
        postOrder(localRoot.rightChild);
        localRoot.displayNode();
    }
}
```

Operations - Delete

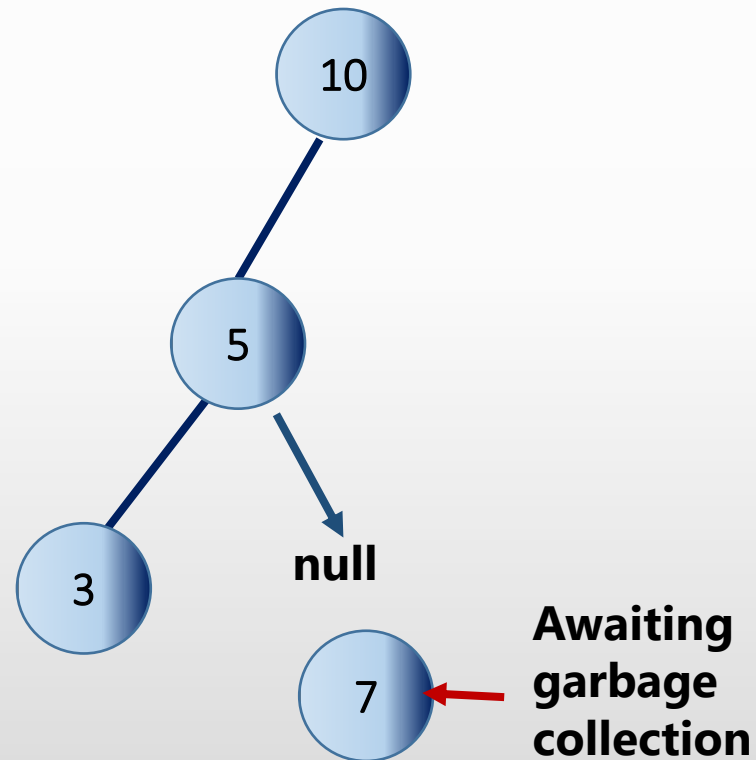
- First find the node to be deleted.
- If the node to be deleted is found there are three cases to be considered.
Whether
 - The node to be deleted is a leaf
 - The node to be deleted has one child
 - The node to be deleted has two children.

Case 1 : The node to be deleted has no children

Delete 7



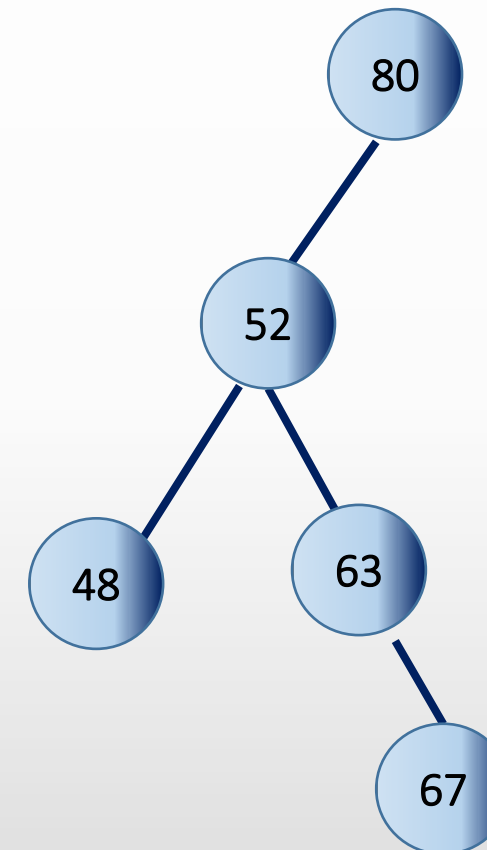
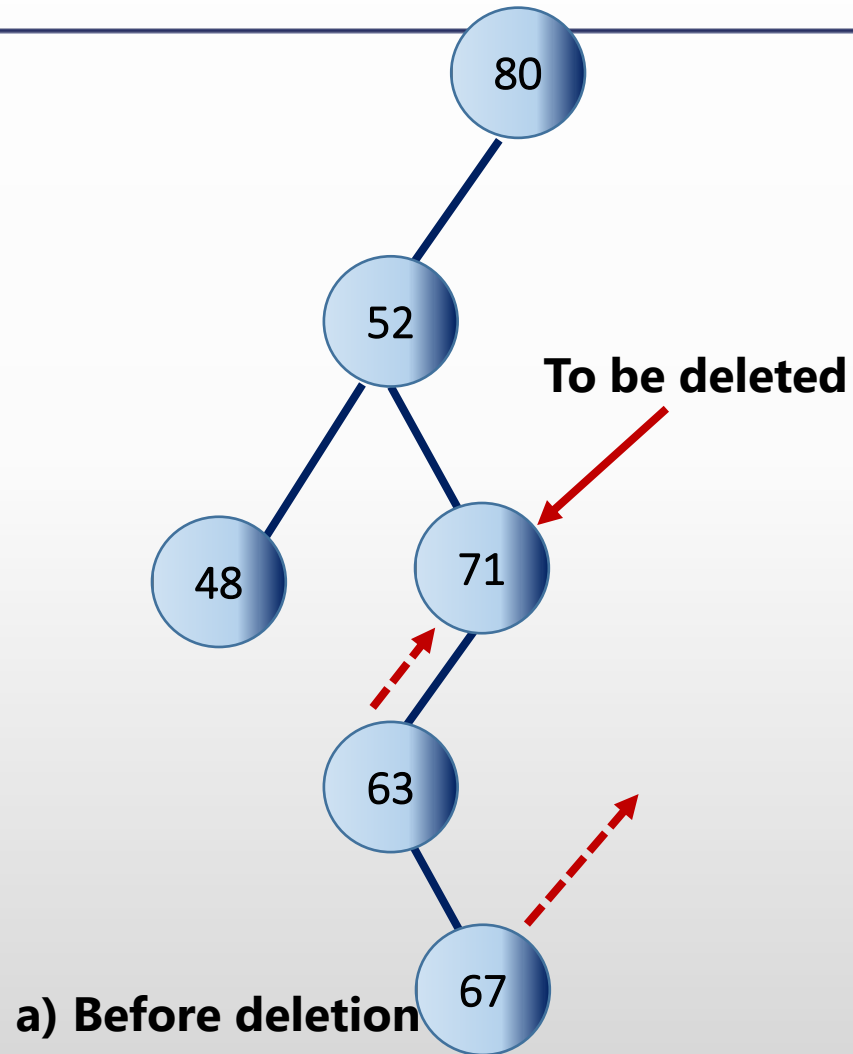
a) Before deletion



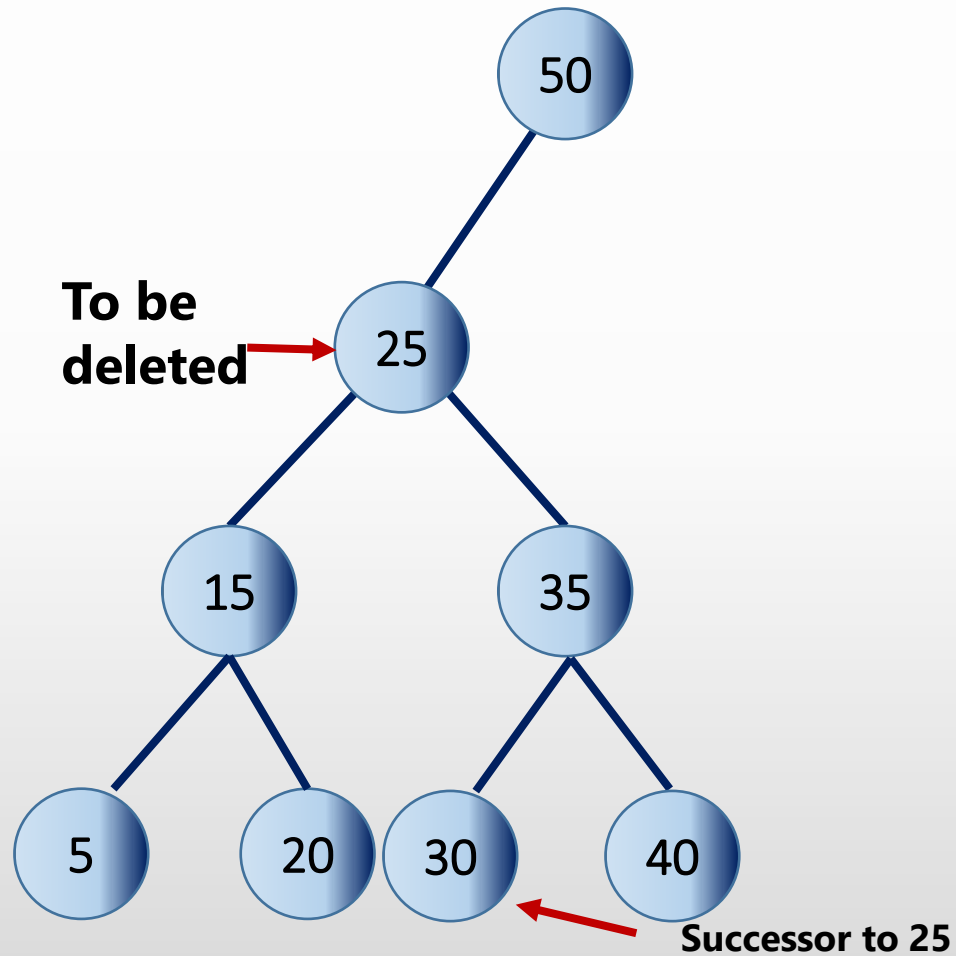
b) After deletion

Awaiting
garbage
collection

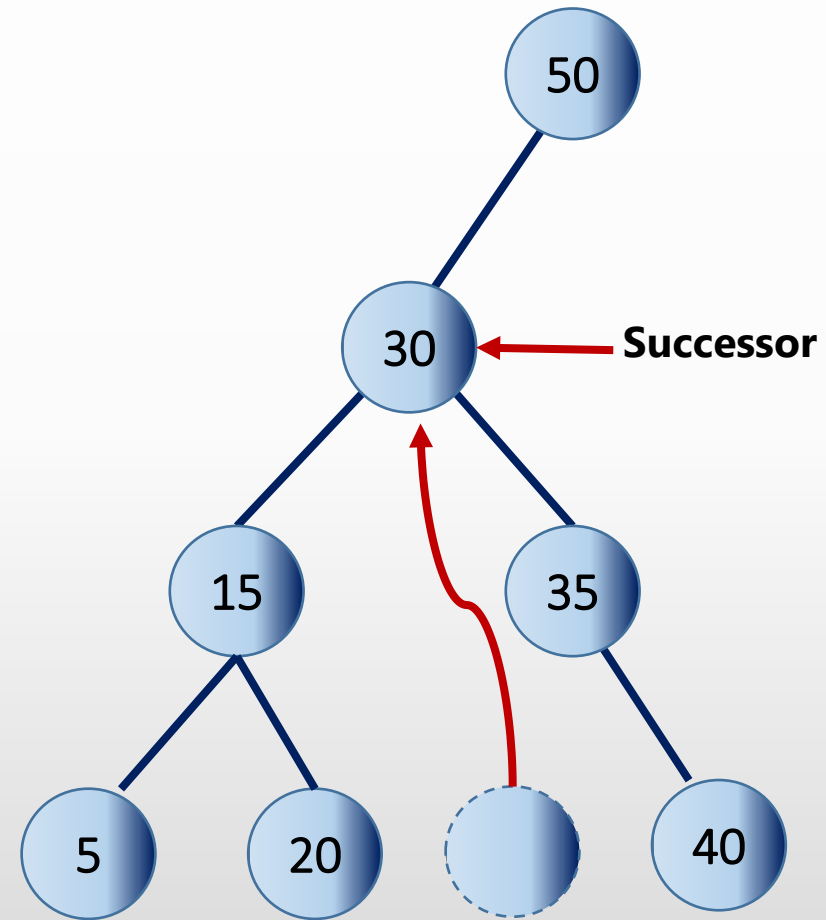
Case 2 : The node to be deleted has one child



Case 3 : The node to be deleted has two children



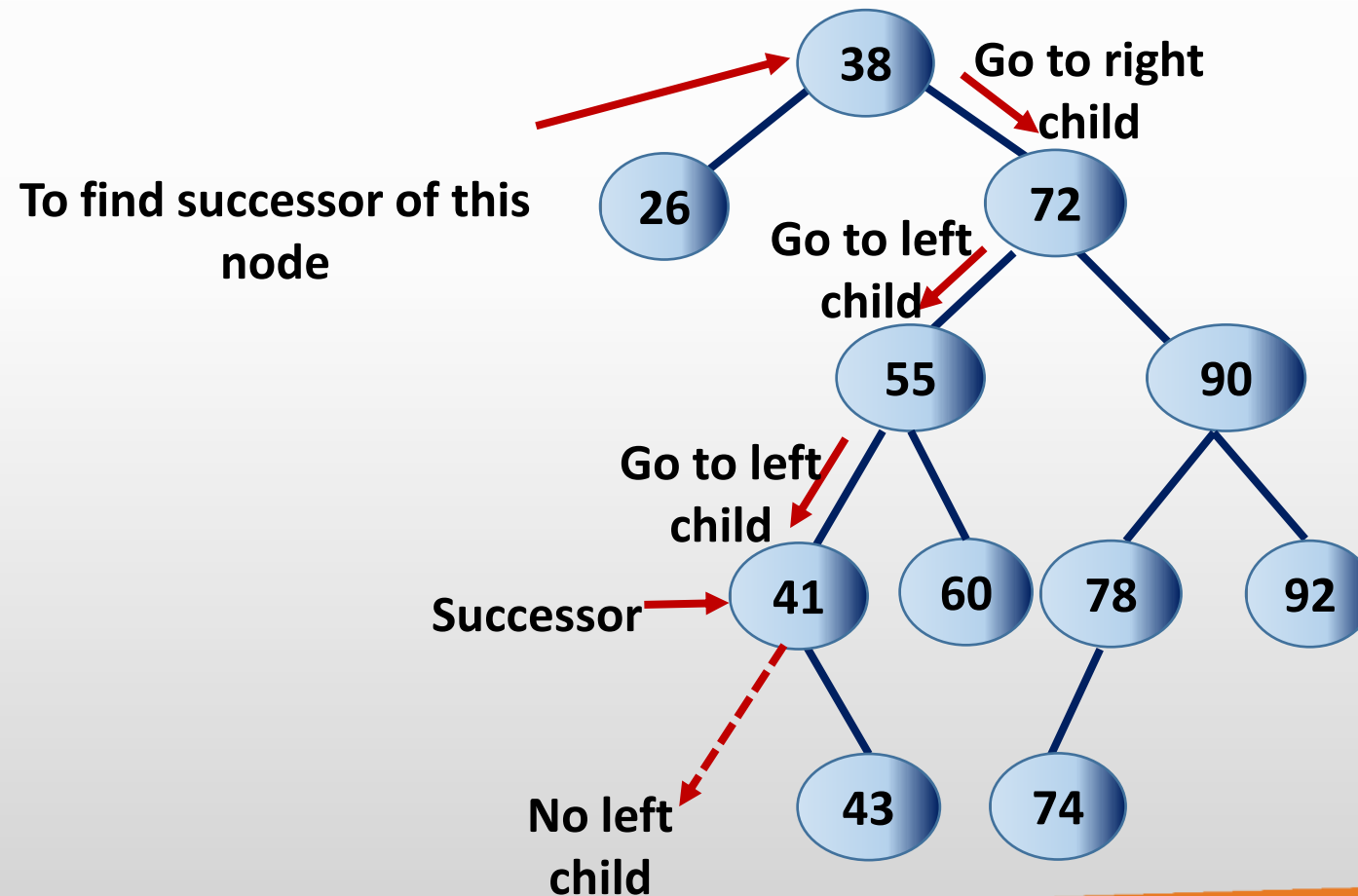
a) Before deletion



b) After deletion

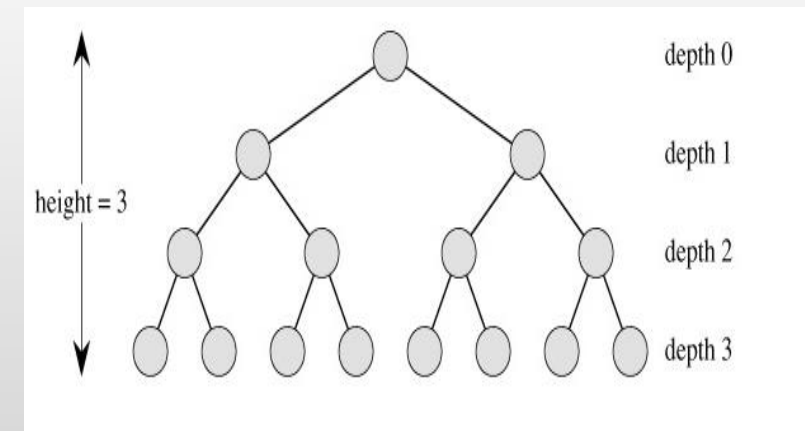
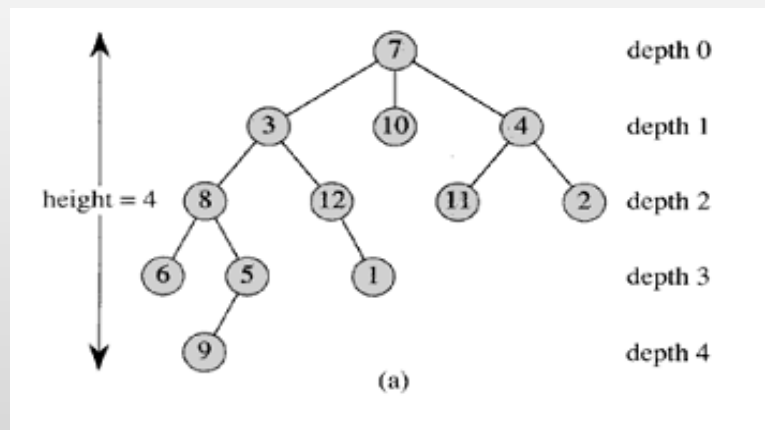
How to find a successor of a node?

- In a Binary Search Tree, successor of a node is a node with next-highest key.



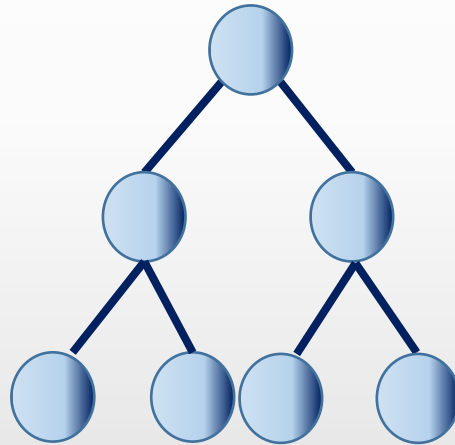
Tree Terminology

- **Degree of a node:** The number of children it has
- **Depth:** The depth of x in a tree is the length of the path from the root to a node x .
- **Height:** The largest depth of any node in a tree.



Full Binary Tree

- A Full binary tree of height ***h*** that contains exactly $2^{h+1}-1$ nodes



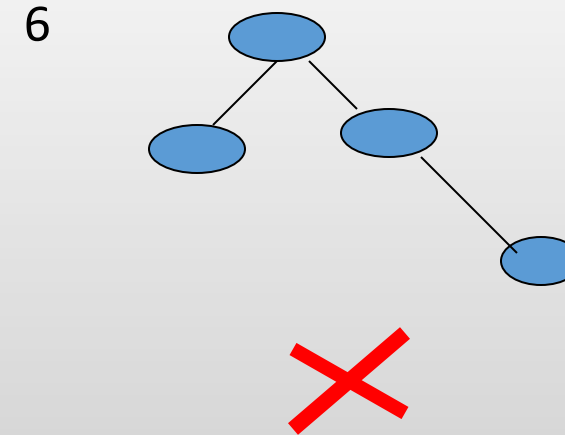
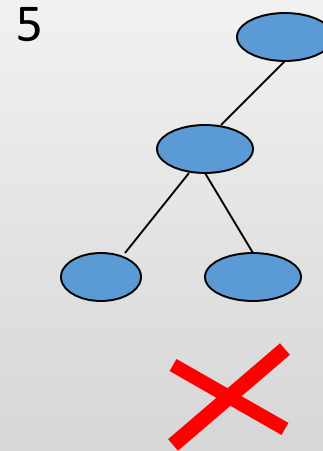
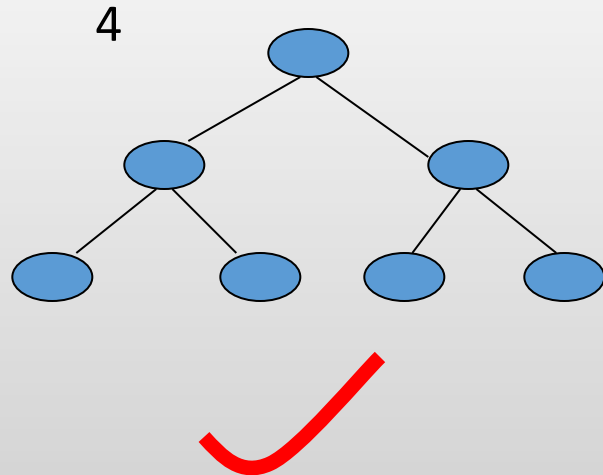
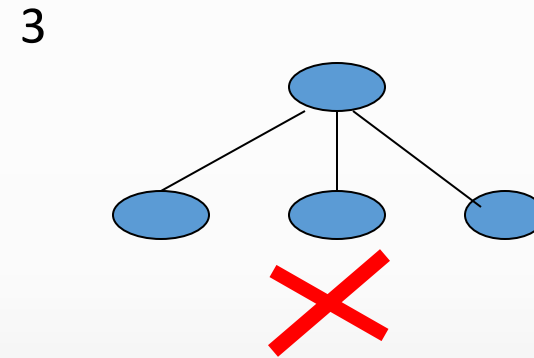
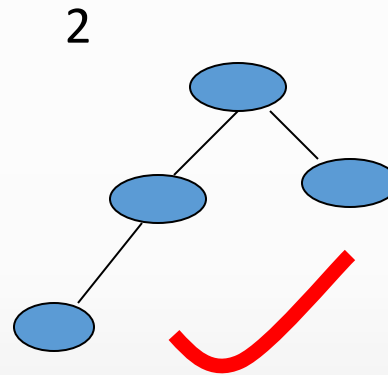
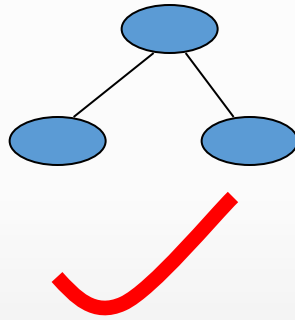
Height, ***h*** = 2, nodes = $2^{2+1}-1=7$

Complete Binary Tree

- It is a Binary tree where each node is either a leaf or has degree ≤ 2 .
- Completely filled, except possibly for the bottom level
- Each level is filled from **left to right**
- All nodes at the lowest level are as far to the left as possible
- Full binary tree is also a complete binary tree

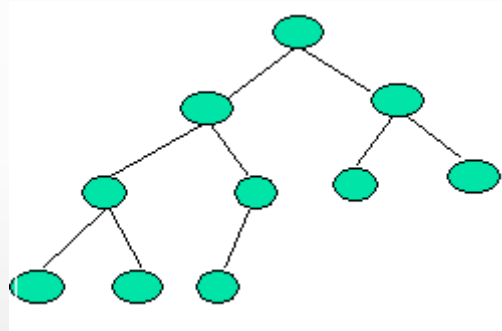
Question

- Find Complete Binary Trees



Height of a complete binary tree

- Height of a complete binary tree that contains n elements is $\lfloor \log_2(n) \rfloor$
- Example



Above is a Complete Binary Tree with height = 3

No of nodes: $n = 10$

Height = $\lfloor \log_2(n) \rfloor = \lfloor \log_2(10) \rfloor = 3$