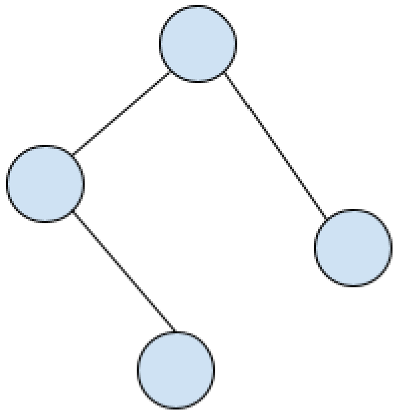
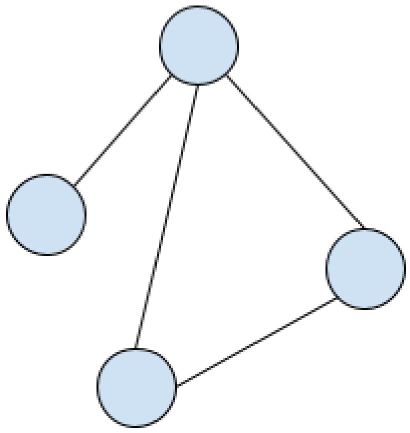
Tree is a structure where every node should be connected and there are no cycles.



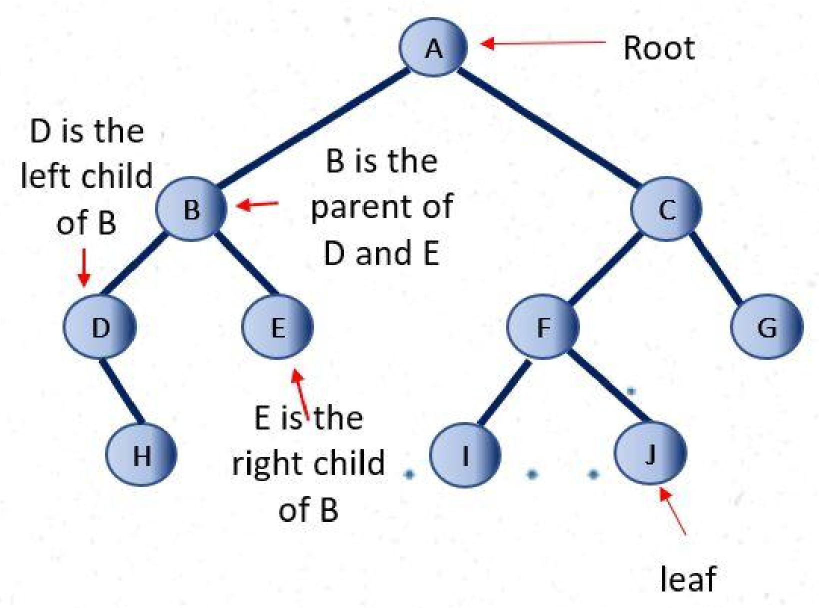
Yes



no

Nodes are represented as circles and they are data items. Their relations are represented by lines.

Binary tree is a tree where its nodes have a maximum of 2 children.



•The node at the top of the tree is called root.

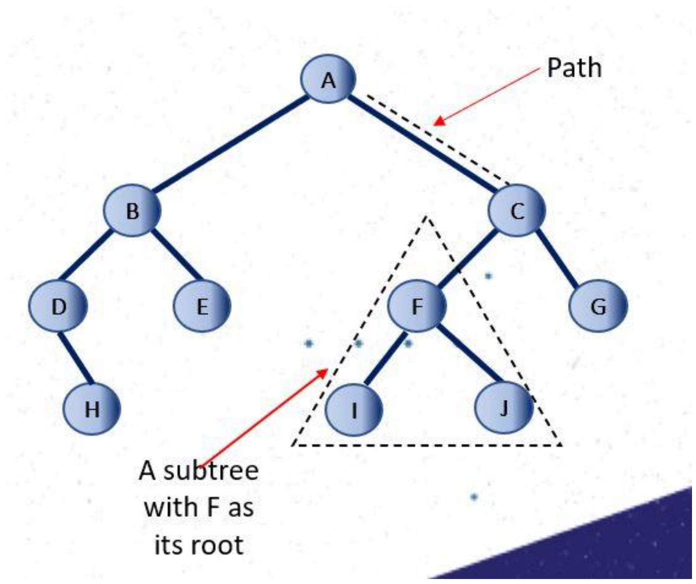
•Any node which has exactly one edge running upwards to other node is called a child

•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 a path? And what is a subtree?



•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 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

- It 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 on a 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

Find

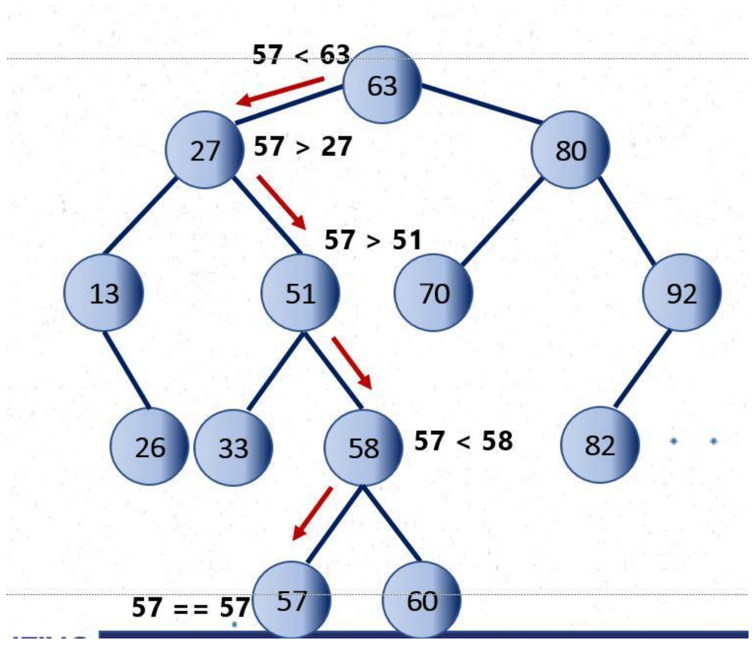
•Find always start at the root.

•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.

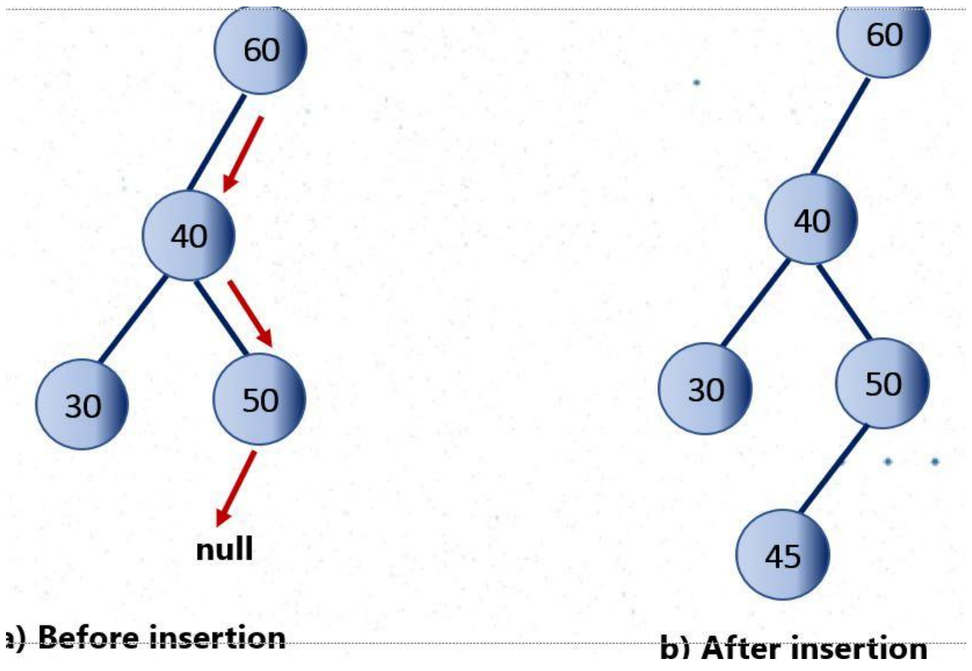


Insert

•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.



Traversing

There are 2 ways of traversing
1)breadth first traversing

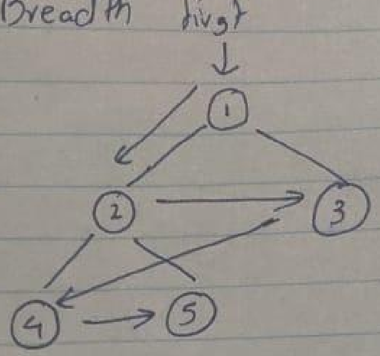
2)depth first traversing

- 1.Inorder Traversal (Left-Root-Right)
- 2.Preorder Traversal (Root-Left-Right)
- 3.Postorder Traversal (Left-Right-Root)

No.: Date: / /

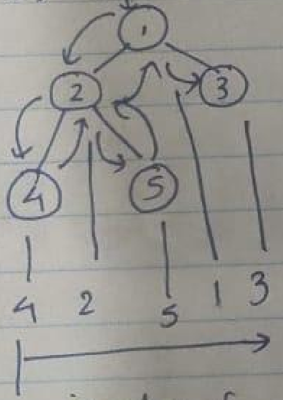
~~① Length~~

① Breadth first



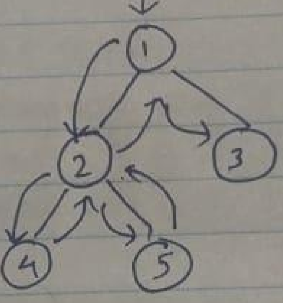
② Depth first

(i) inorder



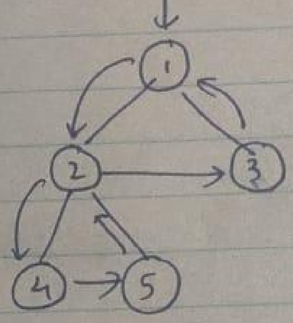
in order of position.

(ii) Preorder (Parent order)



1, 2, 4, 5, 3
parent is written first

(iii) post order



4, 5, 2, 3, 1

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 {
    int iData; // data used as key value
    double fData; // other data
    node leftChild; // this node’s left child
    node rightChild; // this node’s right child

    public void displayNode() {
            System.out.print(“{“);
            System.out.print(iData);
            System.out.print(”, “);
            System.out.print(fData);
            System.out.print( “ } ”);

    }
}
```

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){
        }
    }
```

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

    public Tree() {
        root = null;
    }

    public void insert(int id, double dd) {
        Node newNode = new Node(); // make new node
        newNode.iData = id; // insert data
        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) // (exits internally)
            {
                parent = current;
                if (id < current.iData) // go left?
                {
                    current = current.leftChild;
                    if (current == null) // if end of the line,
                    { // insert on left
                        parent.leftChild = newNode;
                        return;
                    }
                } // end if go left
                else // or go right?
                {
```

```
                current = current.rightChild;
                if (current == null) // if end of the line
                { // insert on right
                    parent.rightChild = newNode;
                    return;
                }
            } // end else go right
        } // end while
    } // end else not root
} // end insert()
  // -------------------------------------------------------------

public Node find(int key) // find node with given key
{ // (assumes non-empty tree)
    Node current = root; // start at root
    while (current.iData != key) // while no match,
    {
        if (key < current.iData) // go left?
            current = current.leftChild;
        else
            current = current.rightChild; // or go right?
        if (current == null) // if no child,
            return null; // didn’t find it
    }
    return current; // found it
}

private void inOrder(node localRoot) {
    if (localRoot != null) {
        inOrder(localRoot.leftChild);
        System.out.print(localRoot.iData + "");
        inOrder(localRoot.rightChild);
    }
}

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

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

Insert

- Create a new node
- Check whether its the first element(if so insert the new node there)
- If not get a node type variable called parent to keep the pointer to the parent
- Initialize the current node with a pointer to node.
- Check if the new nodes id is less than or greater than the current node
- Go left of right accordingly and update the current node pointer
- Check to see if its equal to null(no value was there before)
- If so then add the parent.left or parent.right values with new node.

```
public void insert(int id, double dd) {
        Node newNode = new Node(); // make new node
        newNode.iData = id; // insert data
        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) // (exits internally)
            {
                parent = current;
                if (id < current.iData) // go left?
                {
                    current = current.leftChild;
                    if (current == null) // if end of the line,
                    { // insert on left
                        parent.leftChild = newNode;
                        return;
                    }
                } // end if go left
                else // or go right?
                {
                    current = current.rightChild;
                    if (current == null) // if end of the line
                    { // insert on right
                        parent.rightChild = newNode;
                        return;
                    }
                } // end else go right
            } // end while
        } // end else not root
    } // end insert()
      // -------------------------------------------------------------
```

Find

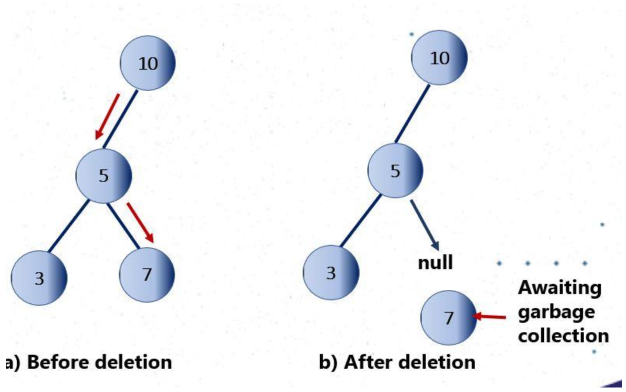
Take a current node type pointer
Initialize it with root
Iterate through the tree tree wither until the current points to a null or until the key is found in the tree

```
public Node find(int key) // find node with given key
    { // (assumes non-empty tree)
        Node current = root; // start at root
        while (current.iData != key) // while no match,
        {
            if (key < current.iData) // go left?
                current = current.leftChild;
            else
                current = current.rightChild; // or go right?
            if (current == null) // if no child,
                return null; // didn’t find it
        }
        return current; // found it
    }
```

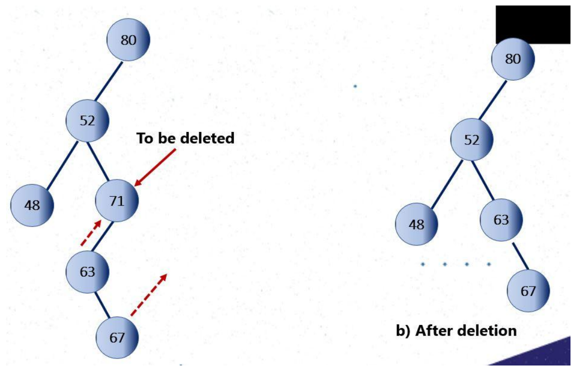
Delete

There are 3 instances of delete

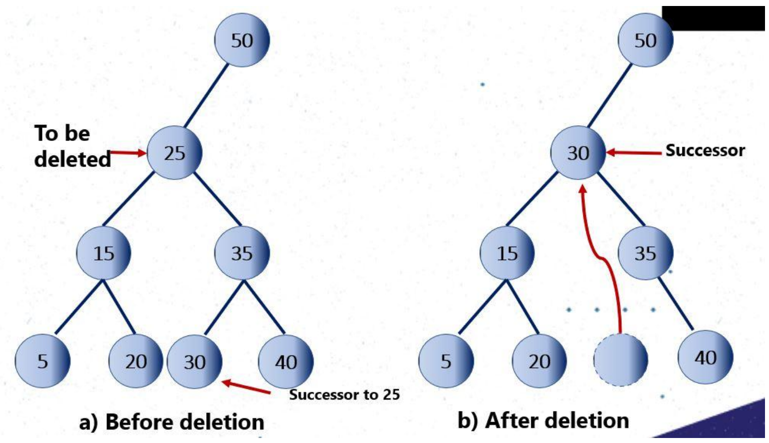
1. Deleting a leaf



2. Deleting a parent with one child


[redacted]

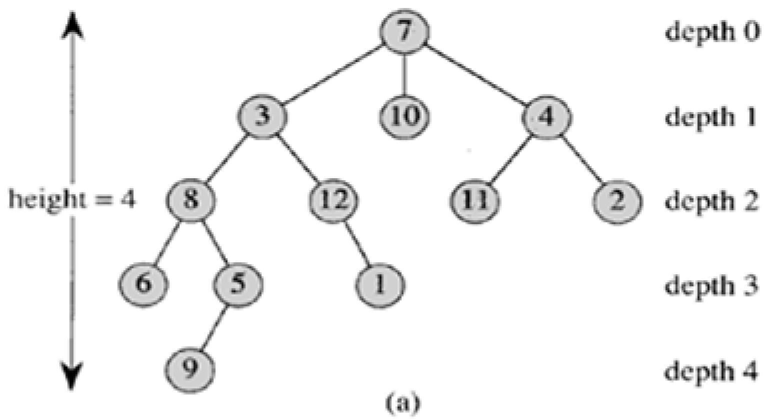
3. Deleting a parent with 2 children


[redacted]

The successor is the next highest value to the one to be deleted.It is in the right branch
In a more algorithmic sense this is the value found by first going right from the value to be deleted and then going left until the value becomes null. The value that is before it went null is the successor value.

Tree terminology

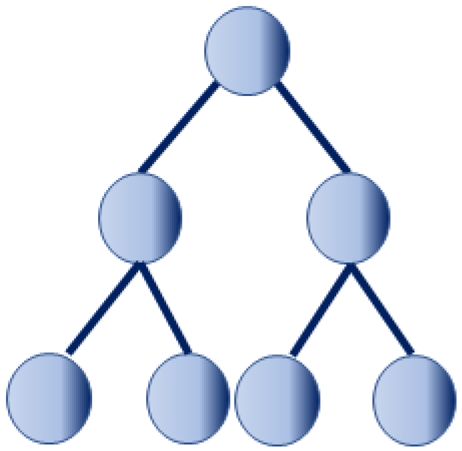
Degree of nodes: the number of nodes in a tree
Depth: the number of layers of horizontal nodes from root to the given node starting root at 0
Height : the highest depth of the tree



(a)

Full binary tree

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



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

Complete binary tree

A binary tree where the nodes that are existing are in order

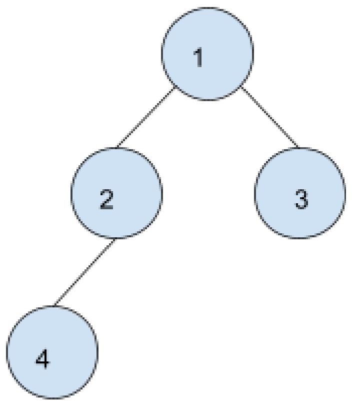
• It is a Binary tree where each node is either a leaf or has degree <= **2**.

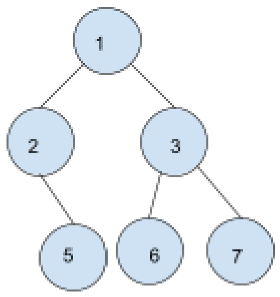
•Completely filled, except possibly for the bottom level

•Each level is filled from **left to right**

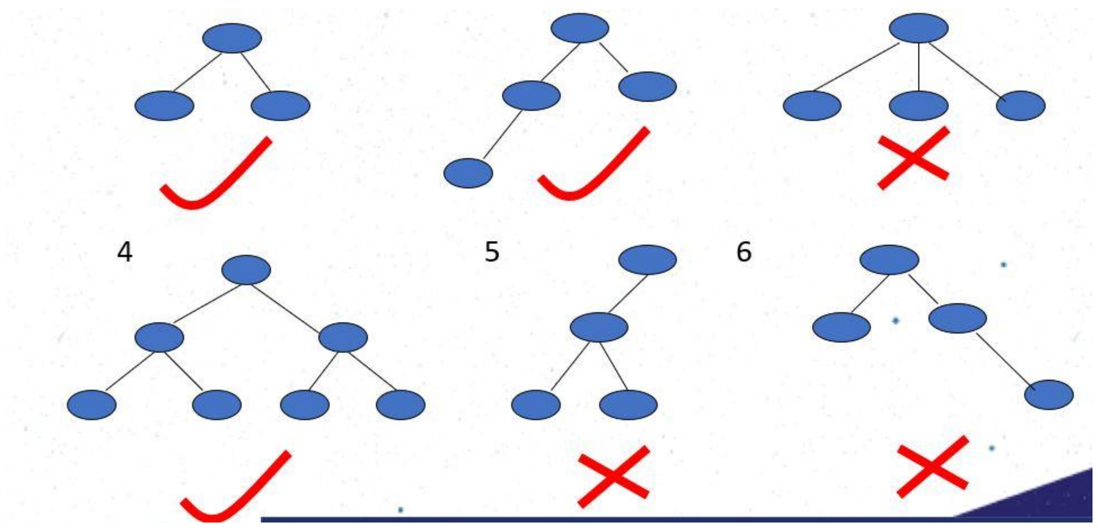
•All nodes at the lowest level are as far to the left as possible

•Full binary tree is also a complete binary tree





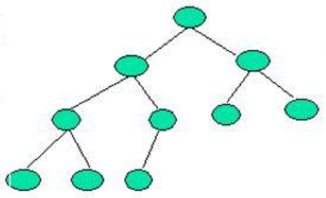
The number are just position values and not the actual keys.In the first one , since all the nodes are in order it is a complete tree.but the second one is not



Height of a complete binary tree

[redacted]

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



Above is a Complete Binary Tree with height = 3

No of nodes: n = 10

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