## Experiment No.8

Aim: Binary search tree ADT

Theory:

Binary Search Tree is a node-based binary tree data structure which has the following properties:

- The left subtree of a node contains only nodes with keys lesser than the node"s key.
- The right subtree of a node contains only nodes with keys greater than the node"s key.
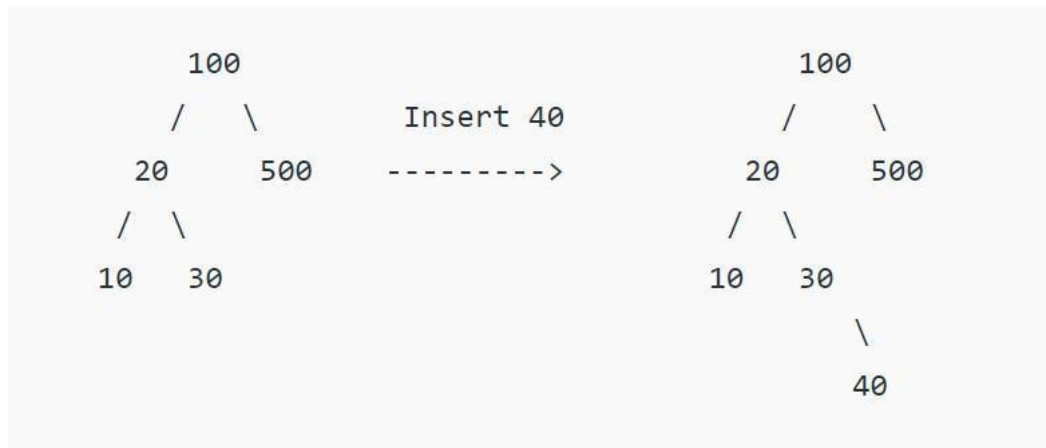- The left and right subtree each must also be a binary search tree.
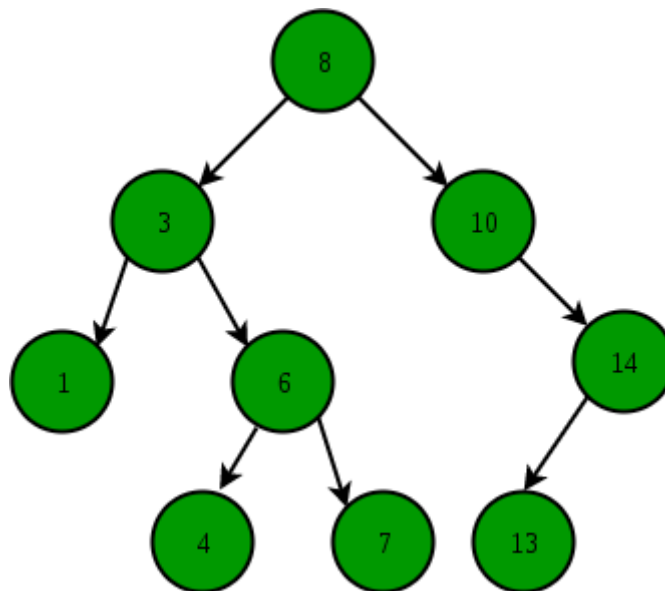
Illustration to search 6 in below tree:

1. Start from the root.

2. Compare the searching element with root, if less than root, then recurse for left, else recurse for right.

3. If the element to search is found anywhere, return true, else return false.
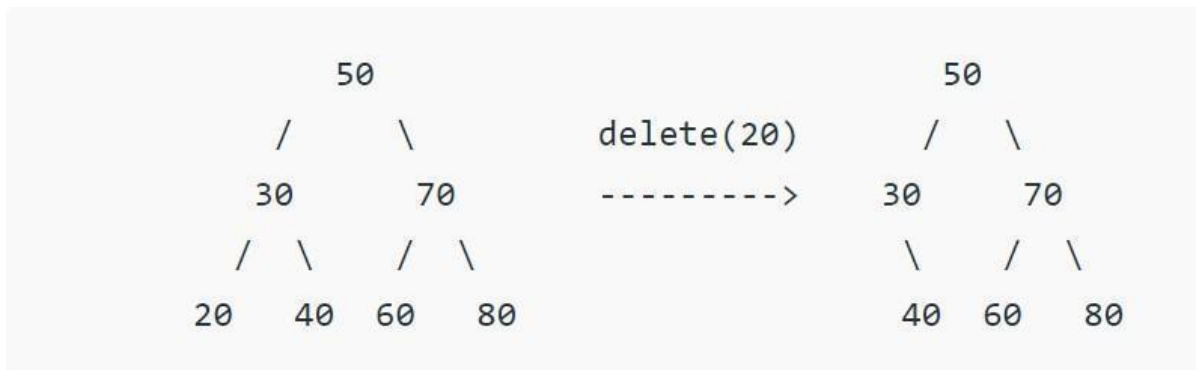
Insertion of a key

A new key is always inserted at the leaf. We start searching a key from the root until we hit a leaf node. Once a leaf node is found, the new node is added as a child of the leaf node.
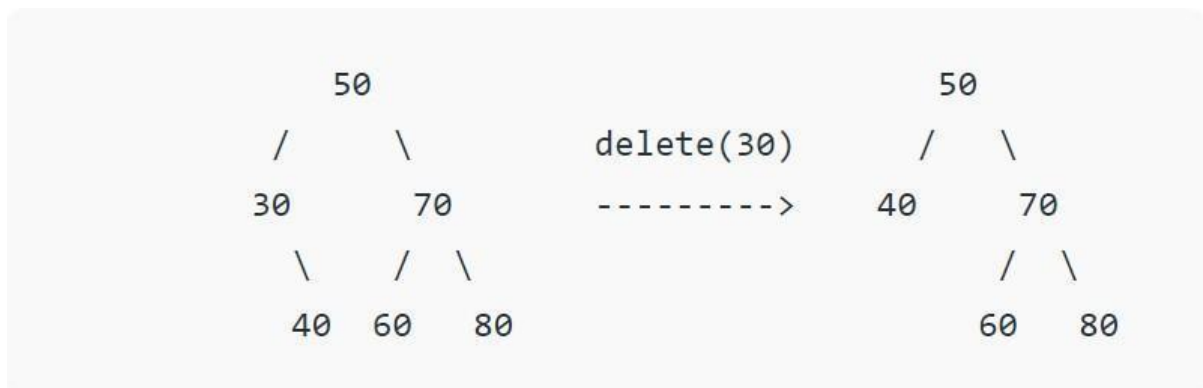
```
        100                             100
        /  \       Insert 40            /  \
      20    500    --------->         20    500
     /  \                            /  \
   10    30                        10    30
                                          \
                                           40
```

Delete a Node
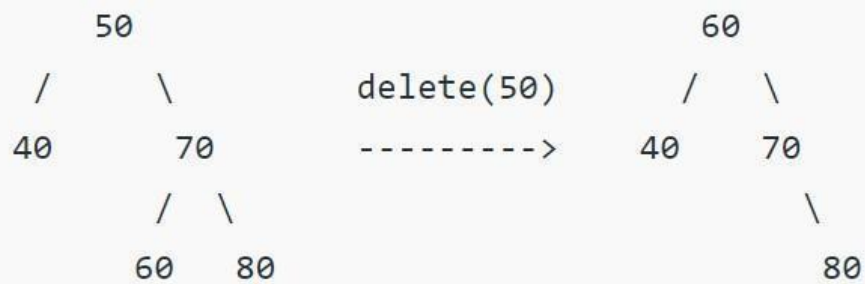
When we delete a node, three possibilities arise.

1) *Node to be deleted is the leaf:* Simply remove from the tree.

```
           50                                  50
         /     \        delete(20)           /    \
       30       70      --------->         30       70
      /  \     /  \                          \     /  \
    20   40  60    80                        40  60    80
```

2) *Node to be deleted has only one child:* Copy the child to the node and delete the child

```
           50                                  50
         /     \        delete(30)           /    \
       30       70      --------->         40       70
         \    /  \                                 /  \
        40  60    80                             60    80
```

3) *Node to be deleted has two children:* Find inorder successor of the node. Copy contents of the inorder successor to the node and delete the inorder successor. Note that inorder predecessor can also be used.

```
        50                                    60
      /    \         delete(50)             /    \
    40       70      - - - - - - - - ->    40     70
           /  \                                     \
          60    80                                    80
```

The important thing to note is, inorder successor is needed only when the right child is not empty. In this particular case, inorder successor can be obtained by finding the minimum value in the right child of the node.

Conclusion: (Students write conclusion in your own words. U have to describe what u you understood from the experiment and the concept of the experiment. **Conclusion carry 4 marks out of 10**)