Today's Content :-

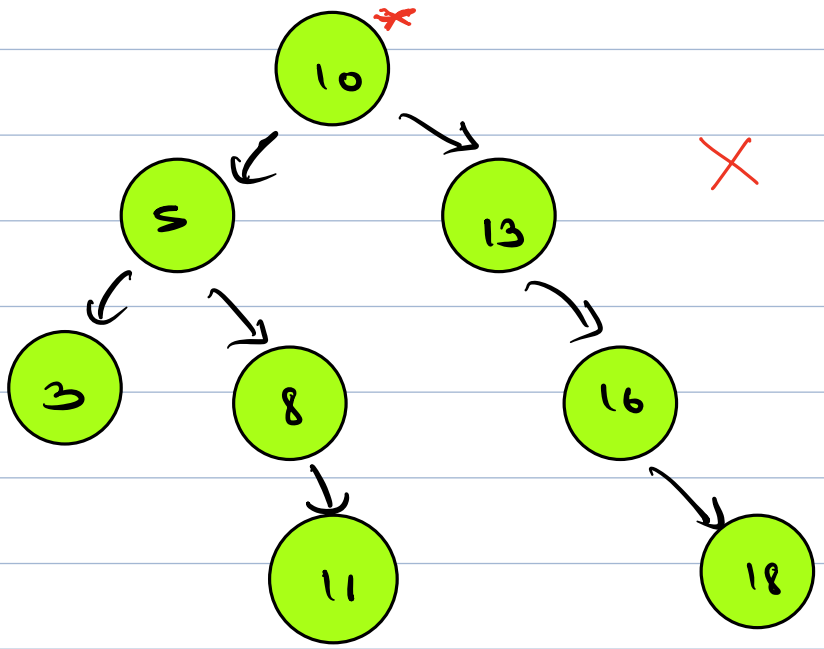
- → BST Basics + Searching
- → Insert / Search / Delete
- → isBST()
- → Construct BST from sorted Arr.

BST : Binary Search Tree

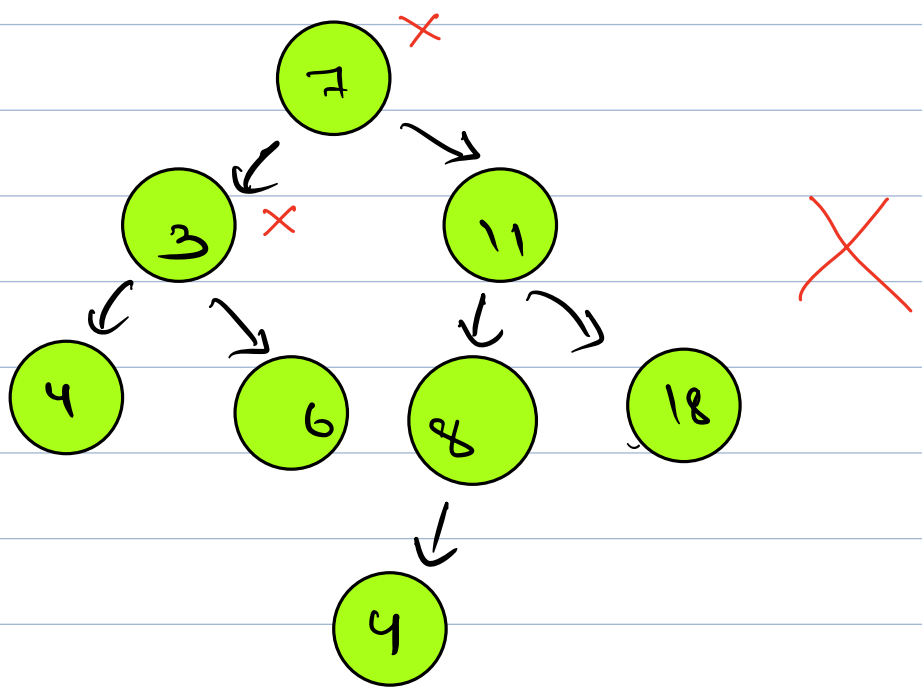
A B.T is called B.S.T if,

for all nodes :- All elements in LST < node < All elements in RST

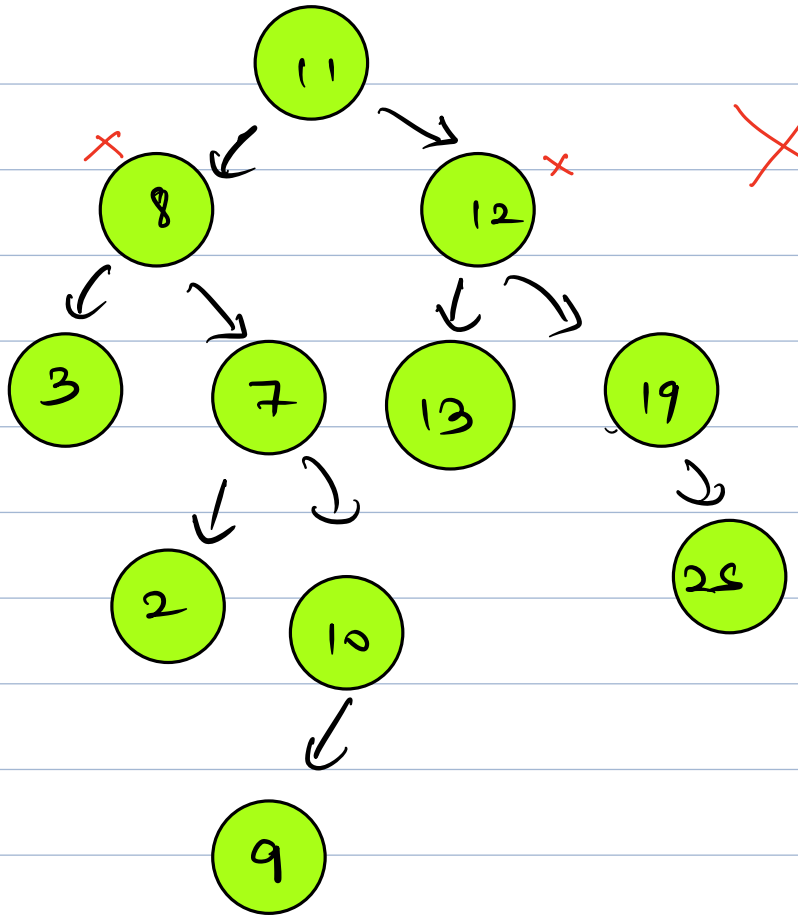
Ex 1 :-



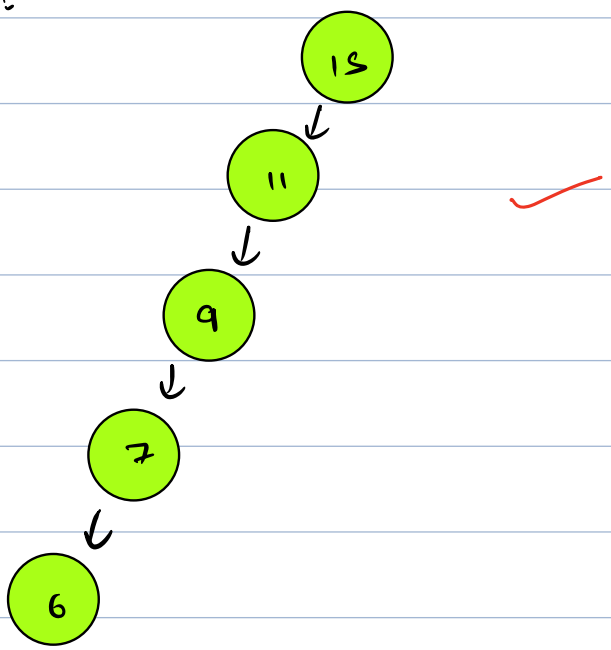
Ex 2 :-



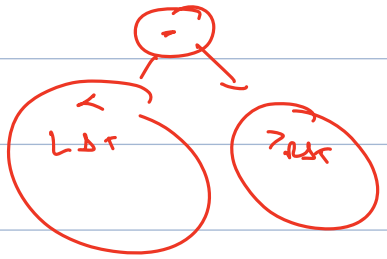
Ex 3:-

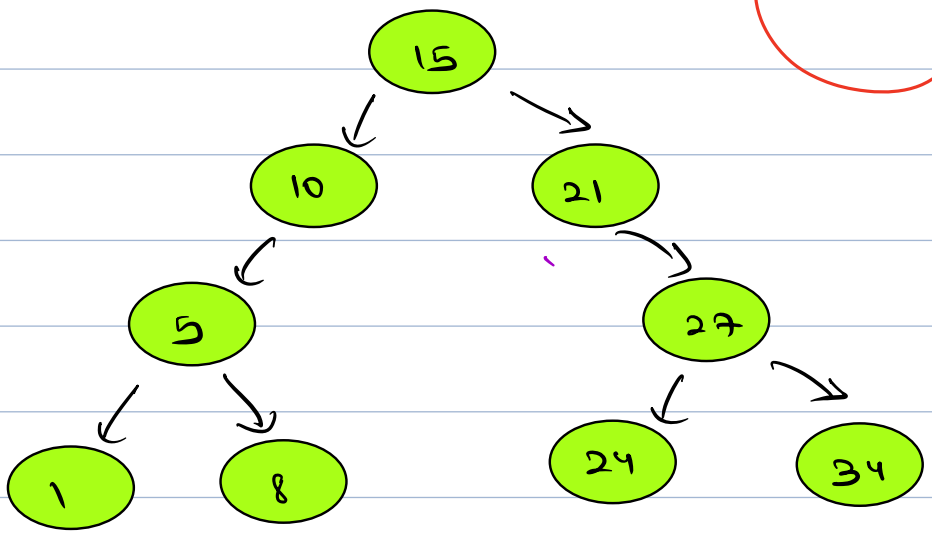


Ex 4:-



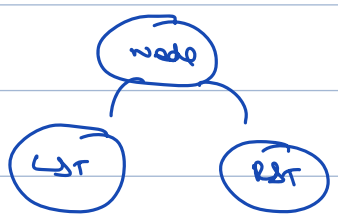
* Interesting Property :-





↳ LDR

Inorder :- 1 5 8 10 15 21 24 27 34



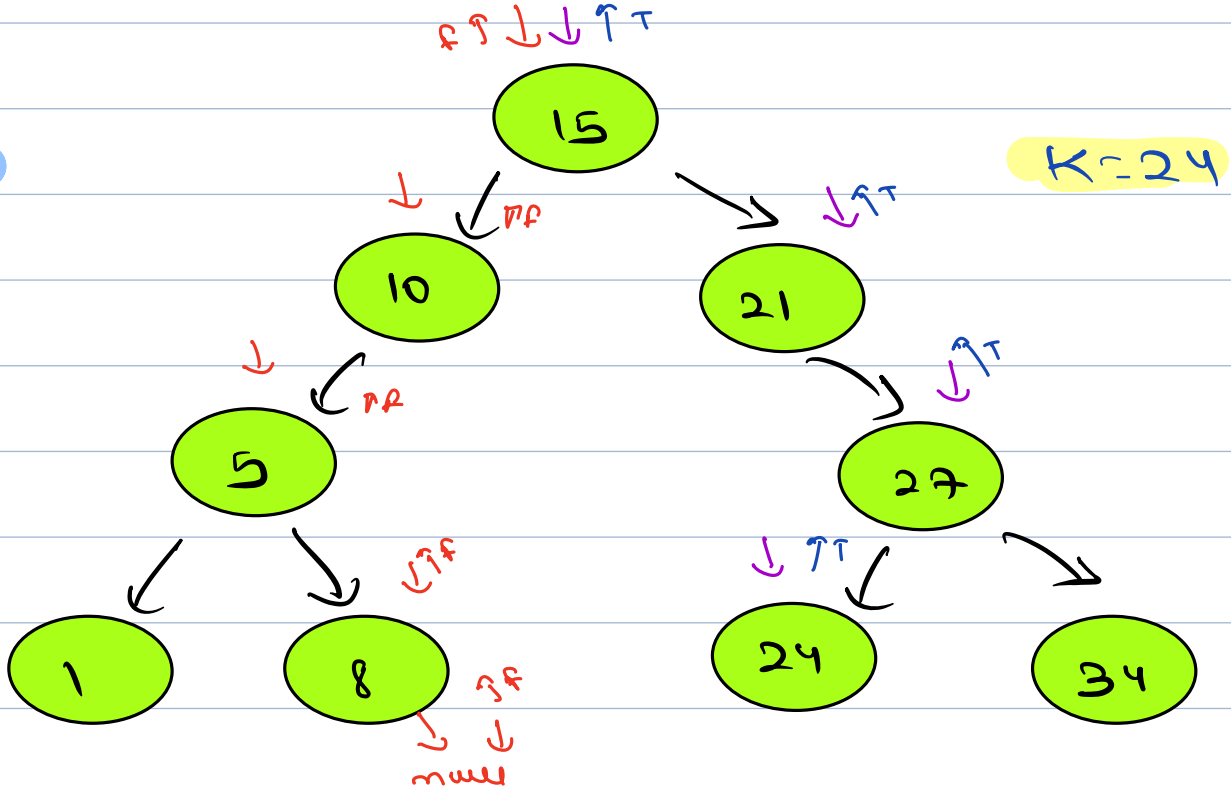
Obs:- Inorder of a BST will always be sorted.

What is a Binary Search Tree (BST)?

31 users have participated

A	A tree with only two nodes	3%
B	A tree where the left child of a node has a value <= the node, and the right child has a value > the node	58%
C	A tree where for a node x, everything on the left has data <= x and on the right > x.	39%
D	A tree that has height log N.	0%

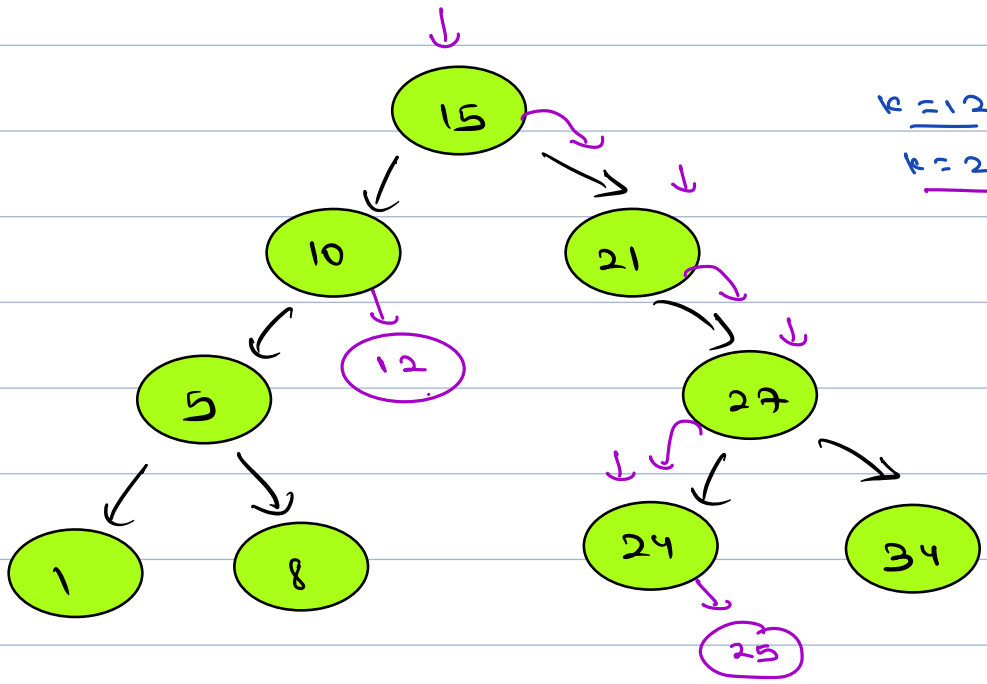
Ques Searching in a B.S.T.



T.C → O(H)
S.C → O(H)

```
bool search (Node root, int k) {
    if (root == null) { return false }
    if (root.data == k) {
        return True;
    } else if (root.data > k) {
        return search (root.left, k)
    } else {
        return search (root.right, k);
    }
}
```

Ques Insertion in a B.S.T.



T.C → O(H), S.C → O(H)

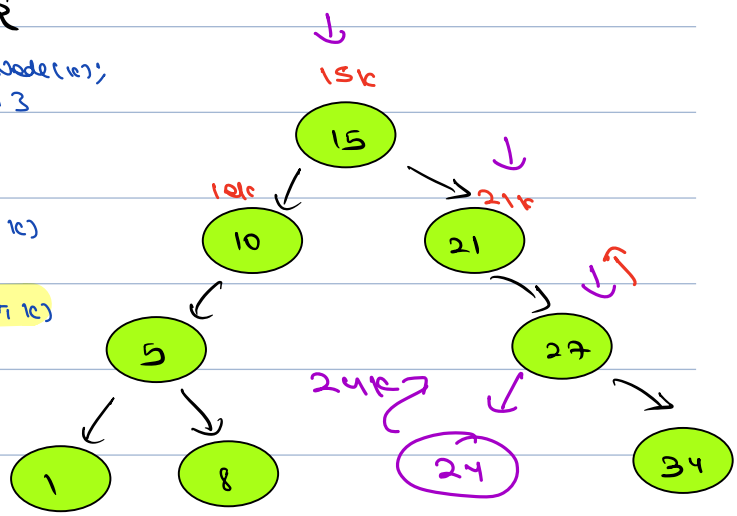
```
Node insert (Node root, int k) {
    if (root == null) { Node nn = new Node(k);
                        return nn; }

    if (root.data > k) {
        root.left = insert (root.left, k)
    } else {
        root.right = insert (root.right, k)
    }

    return root;
}
```

insert (root, 24)

```
Node insert (Node root=15k, int k=24) {
    if (root == null) { Node nn = new Node(k);
                        return nn; }
    if (root.data > k) {
        root.left = insert (root.left, k)
    } else {
        root.right = insert (root.right, k)
    }
    return root;
}
```




```
Node insert (Node root=21k, int k=24) {
    if (root == null) { Node nn = new Node(k);
                        return nn; }
    if (root.data > k) {
        root.left = insert (root.left, k)
    } else {
        root.right = insert (root.right, k)
    }
    return root;
}
```

```
Node insert (Node root=27k, int k=24) {
    if (root == null) { Node nn = new Node(k);
                        return nn; }
    if (root.data > k) {
        root.left = insert (root.left, k)
    } else {
        root.right = insert (root.right, k)
    }
    return root;
}
```

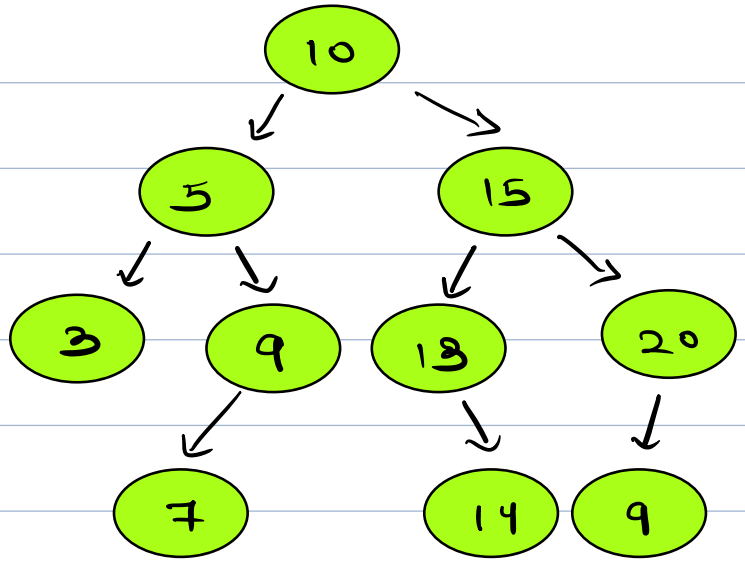
```
Node insert (Node root=null int k=24) {
    if (root == null) { Node nn = new Node(k);
                        return nn }

    if (root.data > k) {
        root.left = insert (root.left, k)
    } else {
        root.right = insert (root.right, k)
    }

    return root;
}
```

→ Try creating a helper function in which you'll pass parent along with node.

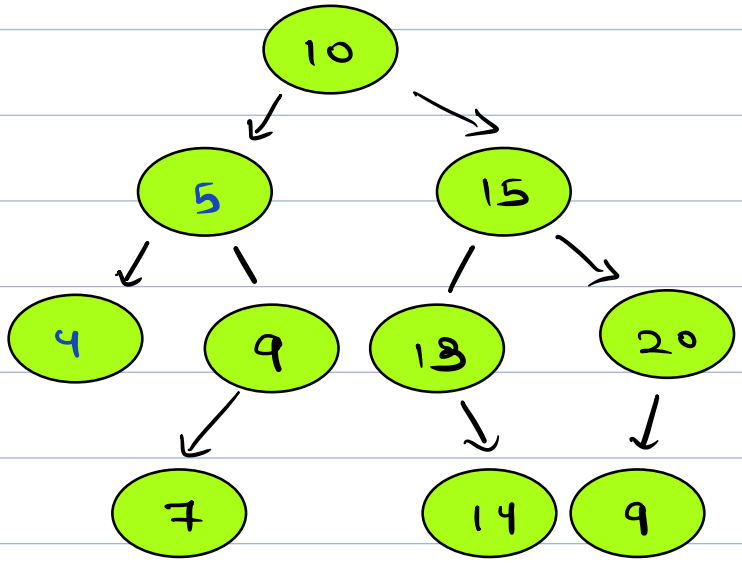
Ques) Given BT check BST or not ?.



idea 1:- check if inorder is sorted or not.

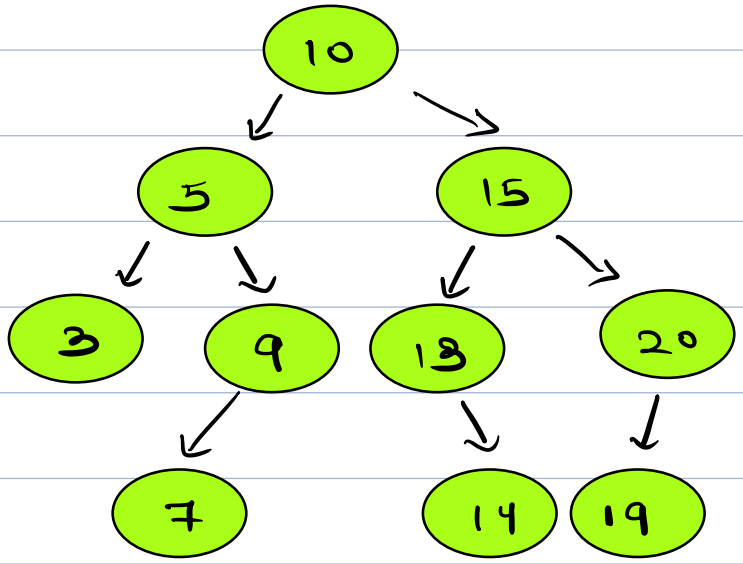
T.C -> O(N), S.C -> O(N).

idea 2 :-



```
bool isBST (node root, int s, int e){
```

Ques) min in BST.

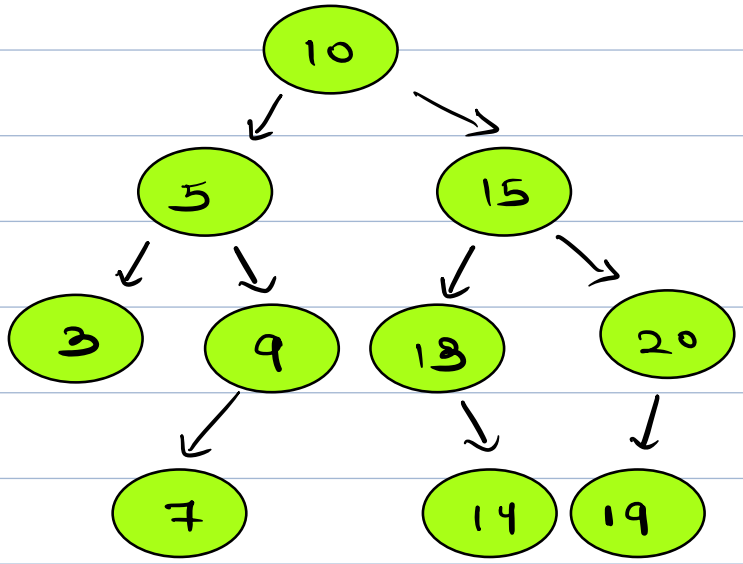


T.C → O(H)

S.C → O(1).

```
temp = root;
while (temp.left != null) {
    temp = temp.left;
}
return temp.data;
```

Ques) Max in BST.



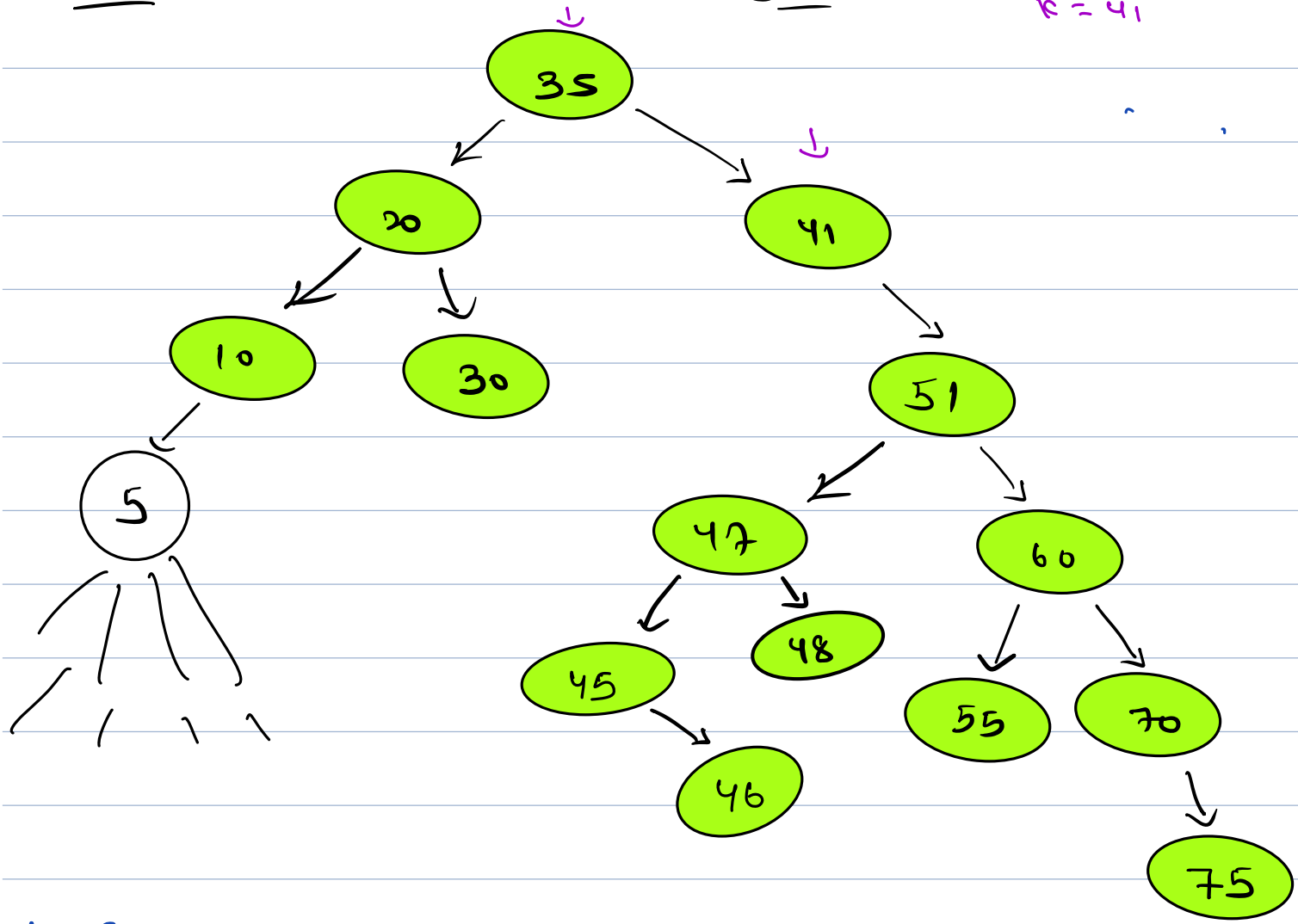

```
temp = root;
while (temp.right != null) {
    temp = temp.right;
}
return temp.data;
```

T.C → O(H)

S.C → O(1)

Break 8:02 Am - 8:10 Am

Ques) Delete a node in BST.



k = 41

k = 5 } no child
k = 30

k = 10 } 1 child
k = 41

k = 51

```
Node delete (Node root, int x) {
    if (root.data > x) {
        root.left = delete (root.left, x);
    }
    else if (root.data < x) {
        root.right = delete (root.right, x);
    }
    else {
        if (root.left == null && root.right == null) {
            return null;
        }
        else if (root.left == null) {
            return root.right;
        }
        else if (root.right == null) {
            return root.left;
        }
```