Today's Content :-

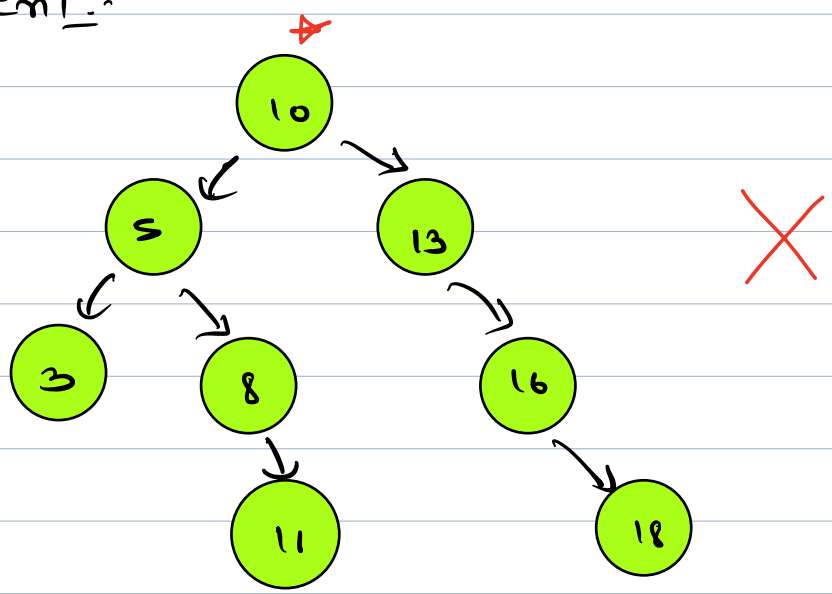
→ BST Basics + Searching
→ Insert / Search / Delete
→ isBST()
→ Construct BST from sorted Arr.

BST : Binary Search Tree

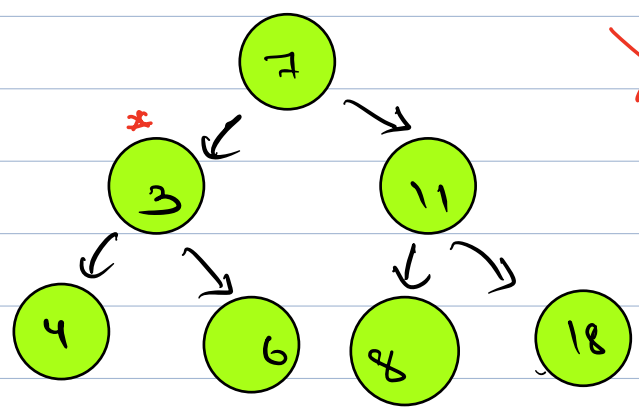
A B.T is called B.S.T if,

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

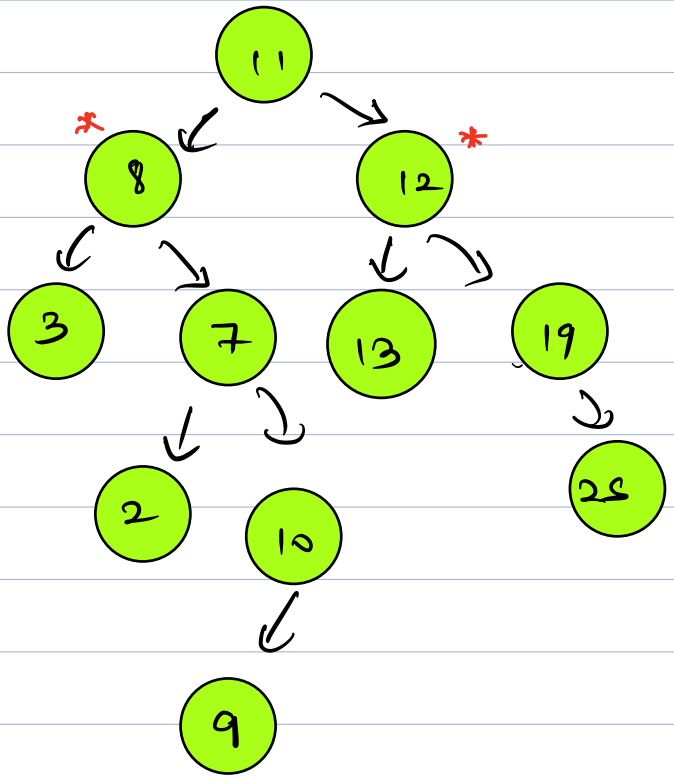
Ex 1 :-



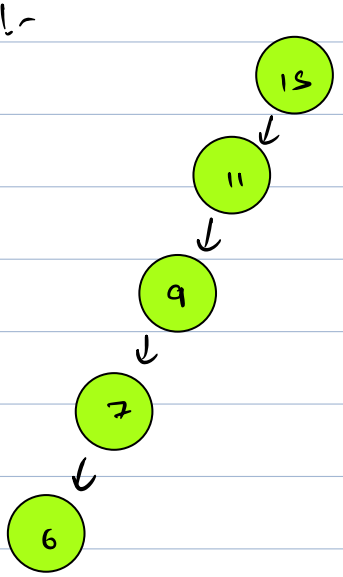
Ex 2 :-



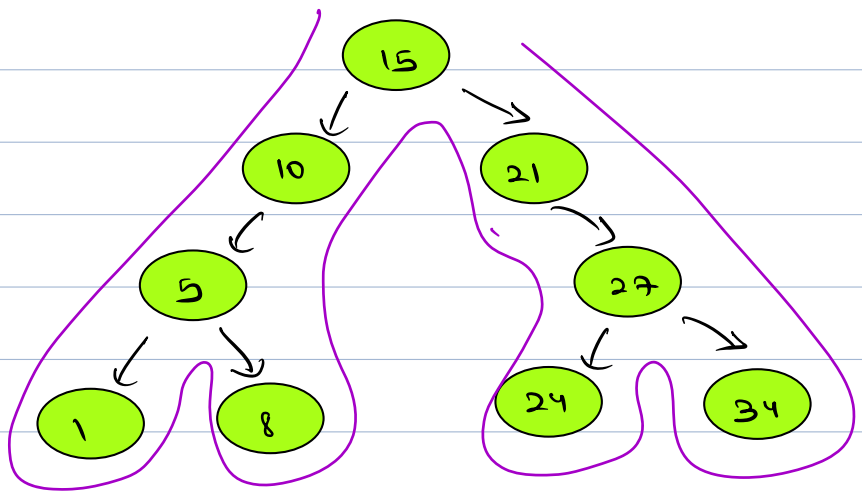
Ex 3:-



Ex 4:-



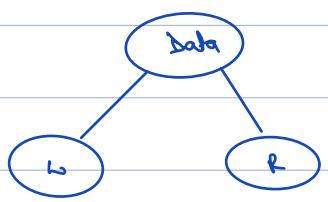
* Interesting Property :-



↳ LDR

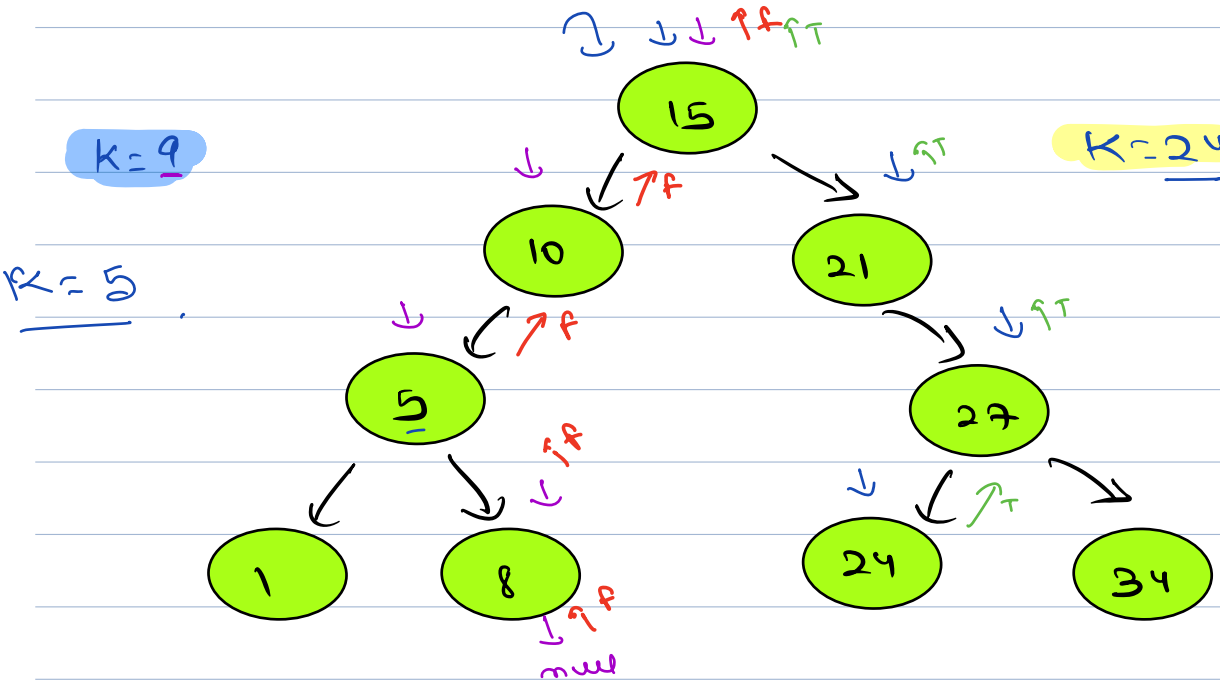
Inorder :- 1 5 8 10 15 21 24 27 34

sorted



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

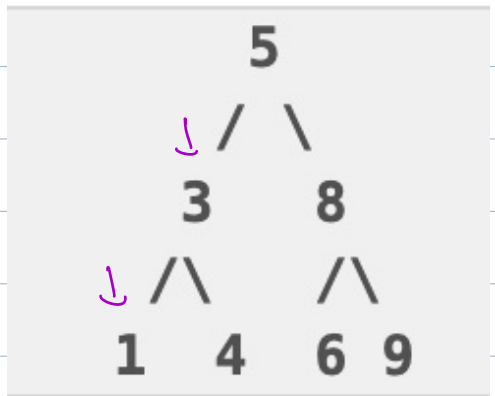
Ques Searching in a B.S.T.



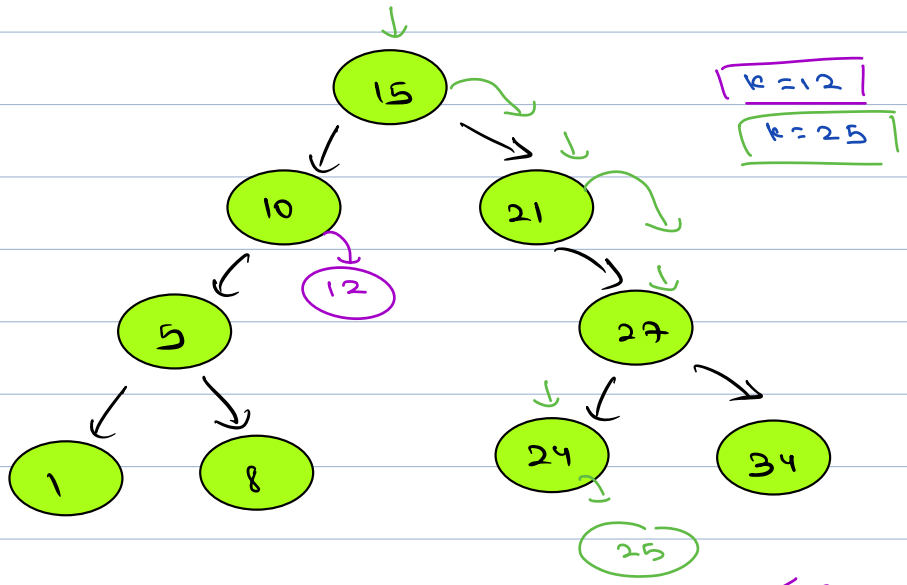
T.C → O(H)
S.C → O(H)

```
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.right, k);
    }
    else { return search (root.left, k);
    }
}
```


5
3 8
1 4 6 9

Ques Insertion in a B.S.T.



T.C → O(H)
S.C → O(H)

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

root = null,

root = insert (null, 24)

root = insert (15, 24)
root, k

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

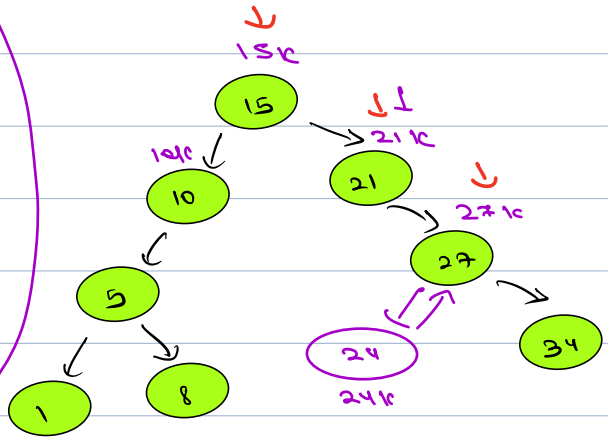
21K

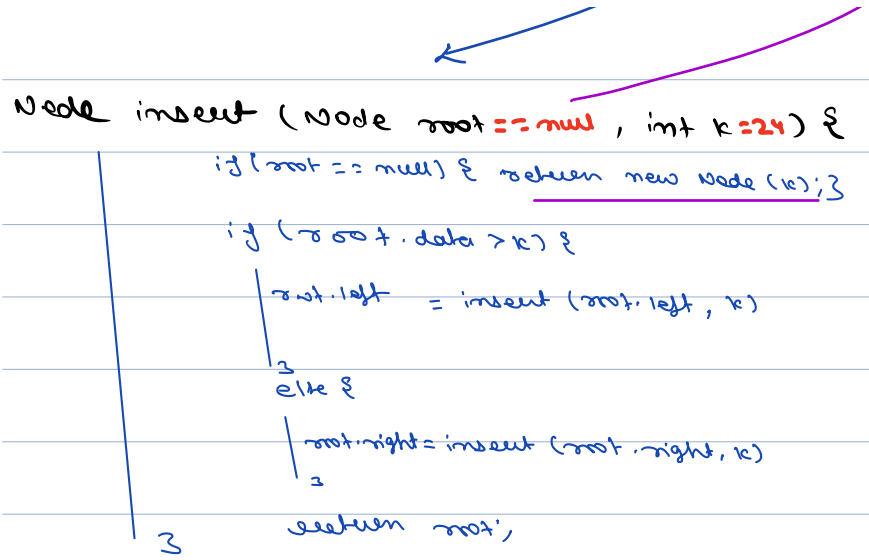
```
Node insert (Node root == 21K, int k=24) {
    if (root == null) { return new Node(k); }
    if (root.data > k) {
        root.left = insert(root.left, k)
    } else {
        root.right = insert(root.right, k)
    }
    return root;
}
```

27K

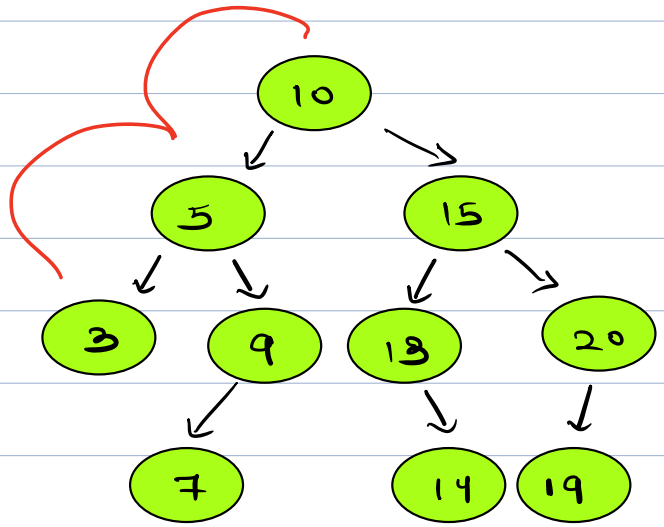
```
Node insert (Node root == 27K, int k=24) {
    if (root == null) { return new Node(k); }
    if (root.data > k) {
        root.left = insert(root.left, k)
    } else {
        root.right = insert(root.right, k)
    }
    return root;
}
```

24K




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

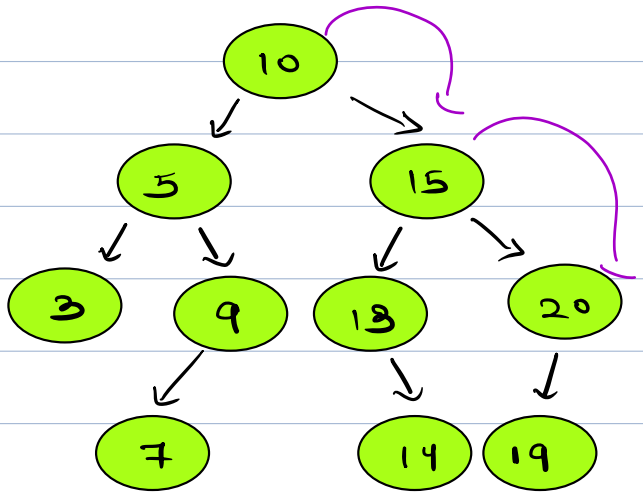
Ques) min in BST.



T.C → O(H)
S.C → O(1).

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

Ques) Max in BST.



T.C → O(h)
S.C → O(1).

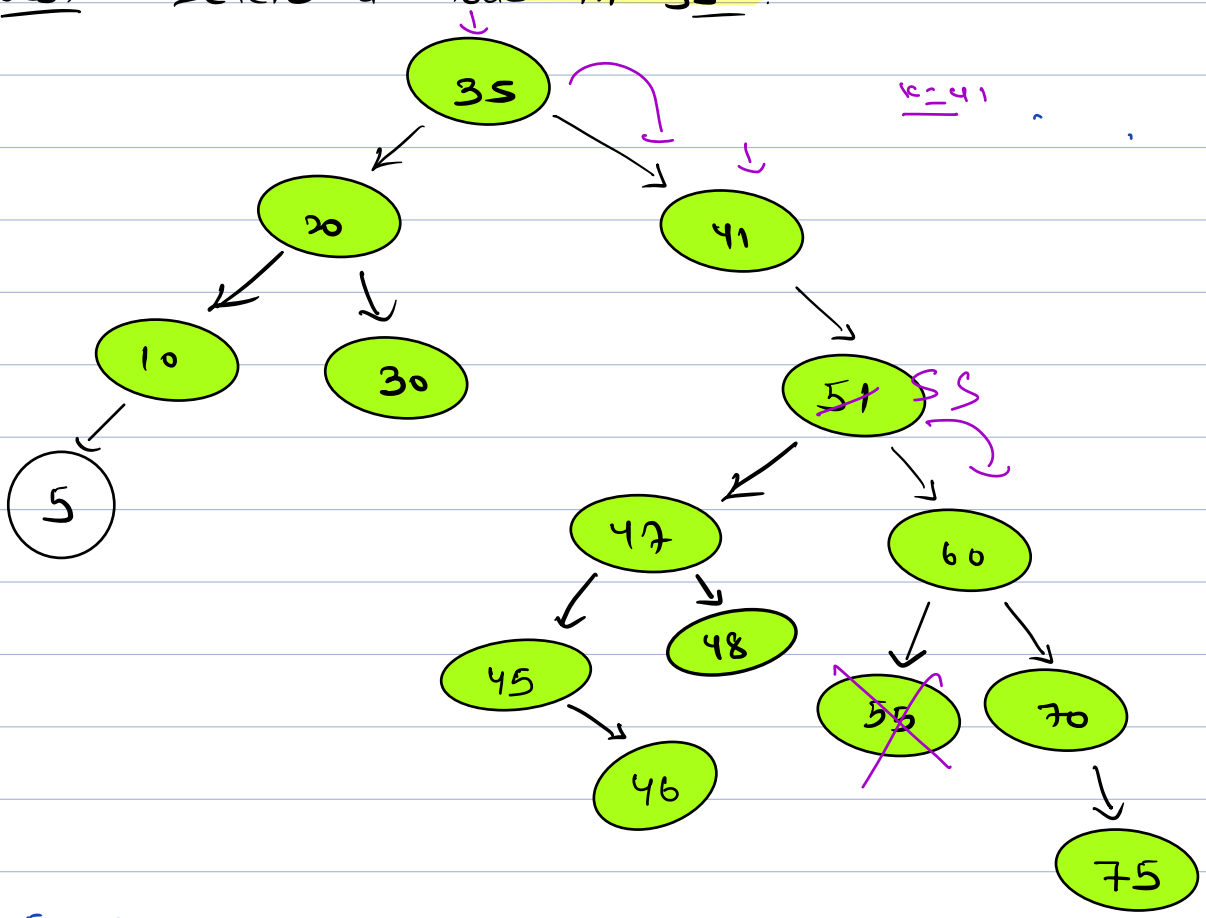
```
temp = root// not null
while(temp.right != null)
{
    temp = temp.right
}
return temp.data;
```

Break 10:05pm - 10:12pm

↳ Insertion in a BST

Max
min

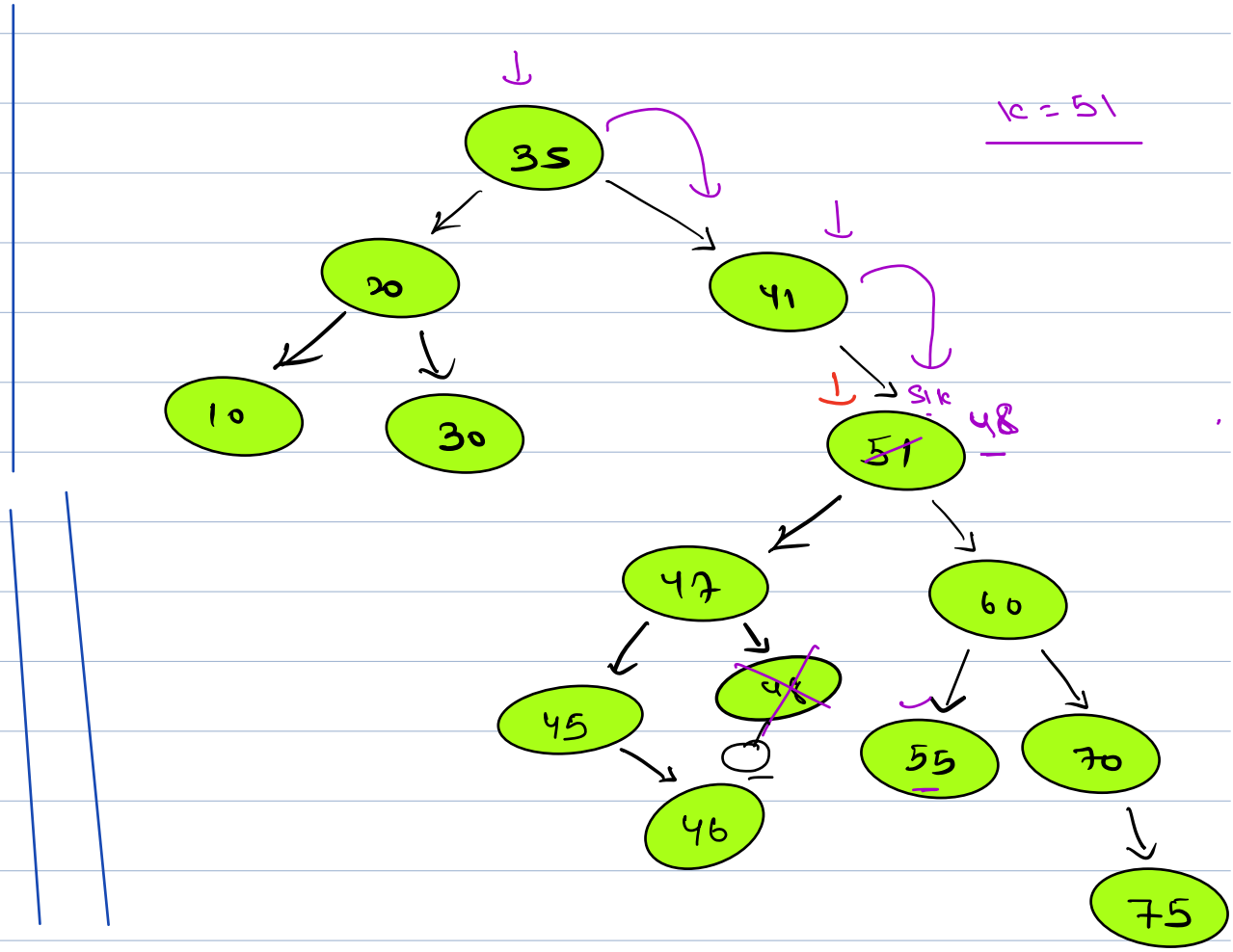
Ques) Delete a node in BST.



k=5
k=30
} 0 children

k=10
k=41
} 1 children

k=51 } 2 children



ic = 51

succ

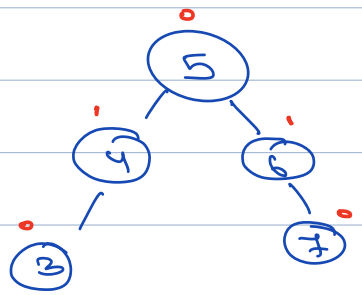
succ.left = delete (47, 48)

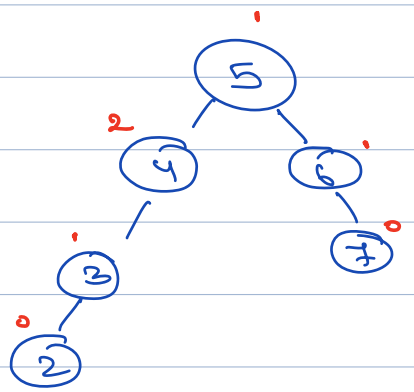
T.C → O(H)
S.C → O(H)

```
Node delete (Node root, int k) {
    if (root.data > k) {
        root.left = delete (root.left, k);
    }
    else if (root.data < k) {
        root.right = delete (root.right, k);
    }
    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;
        }
        else {
            int v = max (root.left);
            root.data = v;
            root.left = delete (root.left, v)
        }
    }
    return root;
```

Ques Construct a Balanced - BST using sorted array.

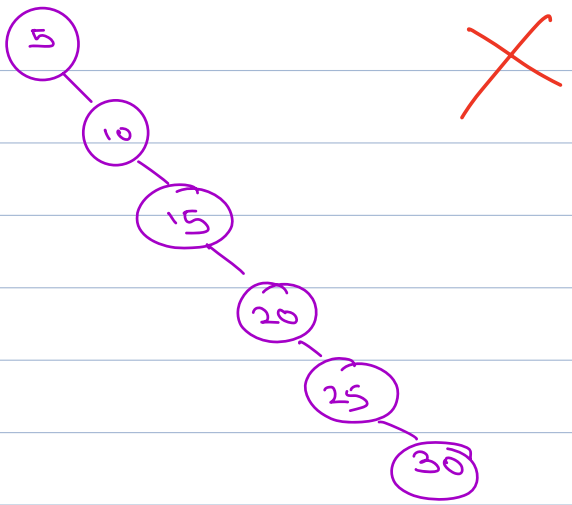
BBST ⟶ $|LST - RST| \leq 1$



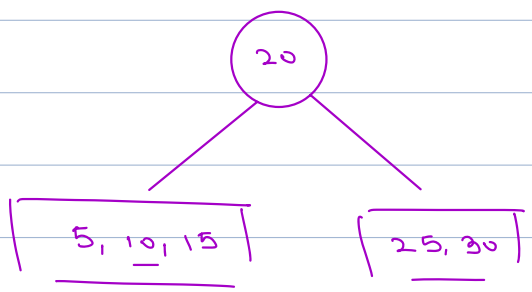


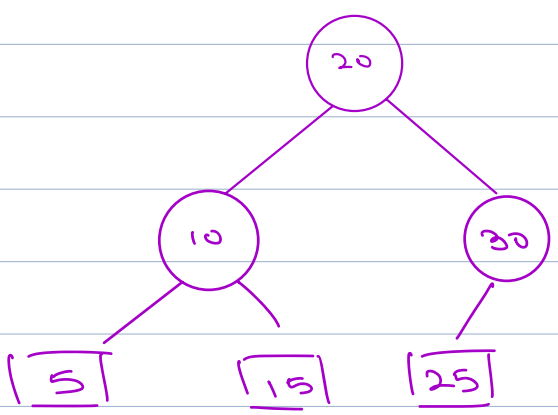
e.g ⟶ 5, 10 15, 20, 25, 30.

idea 1:- Call insert of BST for every node.



e.g 2 5, 10, 15, 20, 25, 30.





root = create (arr, 0, n-1);

```
Node create (int arr[], int s, int e)
    if (s > e) { return null }
    m = s + (e - s) / 2
    Node node = new Node (arr[m])
    node.left = create (arr, s, m-1)
    node.right = create (arr, m+1, e)
    return node;
```

T.C → O(n)

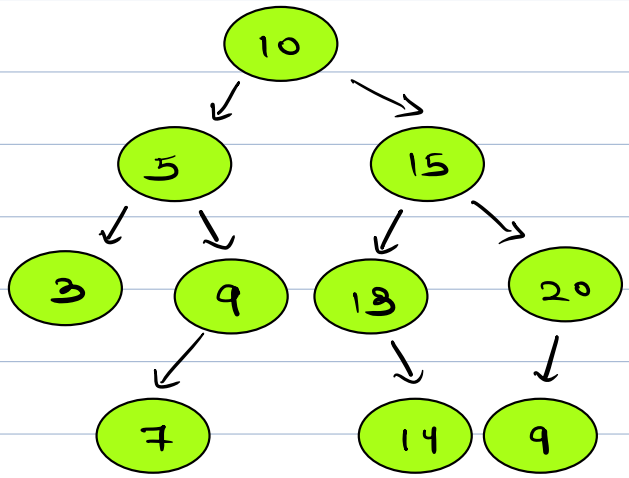
S.C → O(log n)

create (0, 10)

create (0, 4)

create (0, 1)

Ques) Given BT check BST or not ?



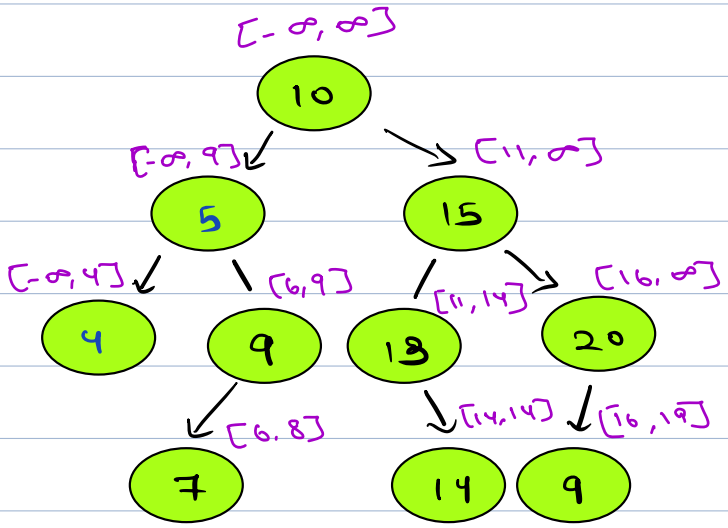
idea 1:- check inorder is sorted or not.

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

code it once.

Todo.

idea 2:-



isBST(root, -∞, ∞);

T.C → O(n)

S.C → O(H)

```
bool isBST (node root, int s, int e){
    if (root == null) {return True}

    if ( s <= root.data && root.data <= e) {
        l = isBST (root.left, s, root.data-1);
        if (l == false) {
            return false;
        }
        r = isBST (root.right, root.data+1, e);
        if (r == false) {
            return false;
        }
        return True;
    }
    return false;
}
```