Trees 3 : Binary Search Trees (BST)

Today's Agenda

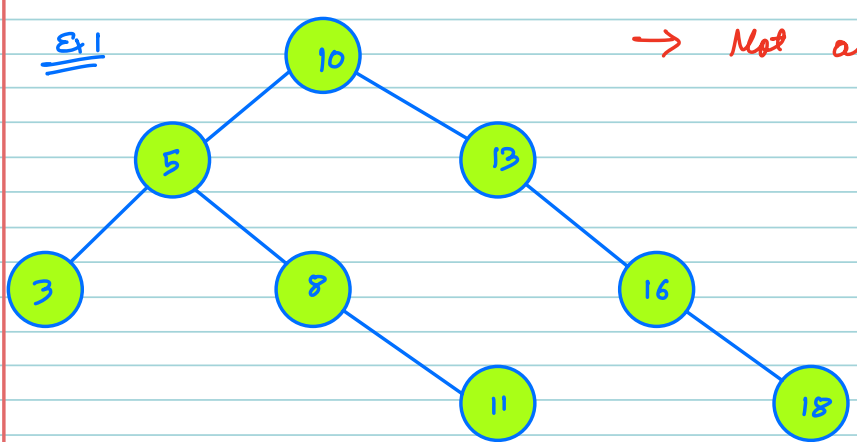
- BST Basics + Searching
- Insert / Search / Delete
- Construct BST from Sorted Array
- Is BST()

Binary Search Trees (BST)

↳ A Binary Tree is called BST if For all nodes

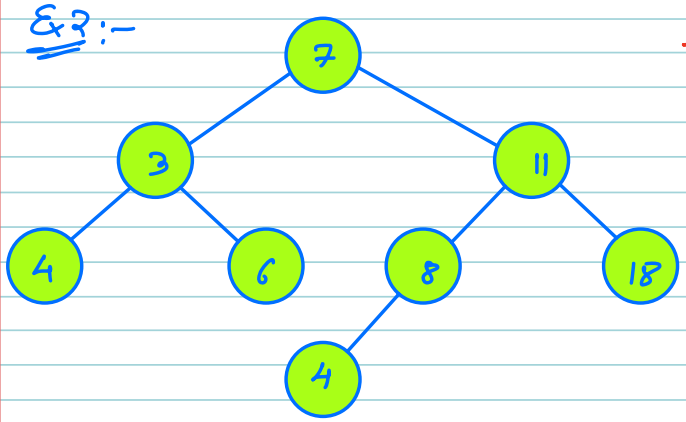
All element in LST	< Node <	All element in RST

Ex 1



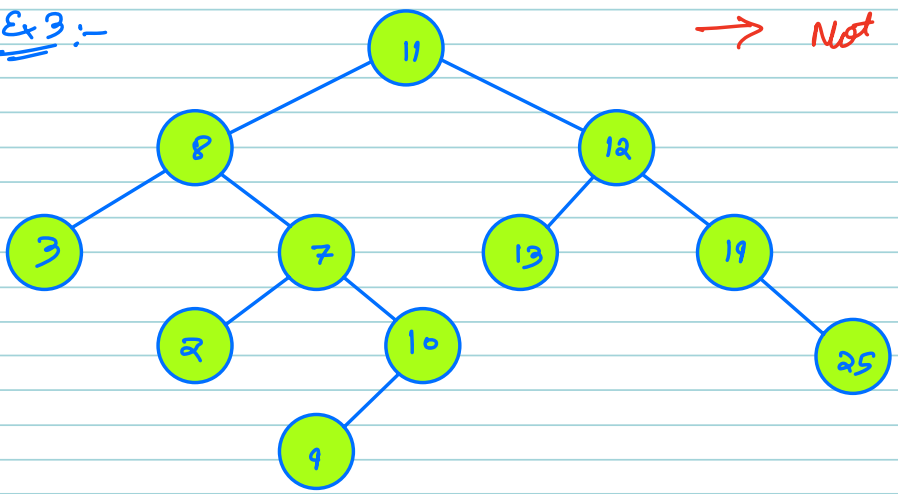
→ Not a BST

Ex2 :-



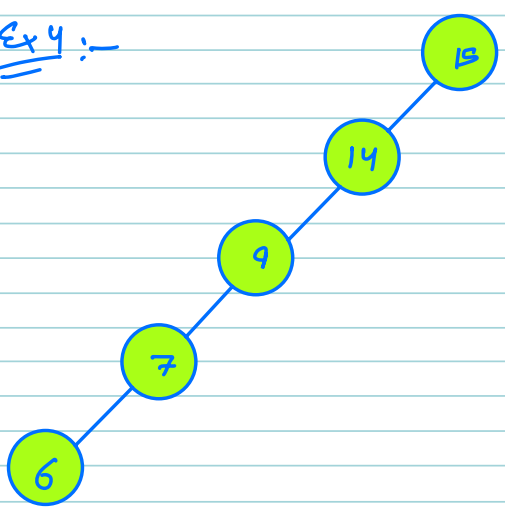
→ Not a BST

Ex3 :-



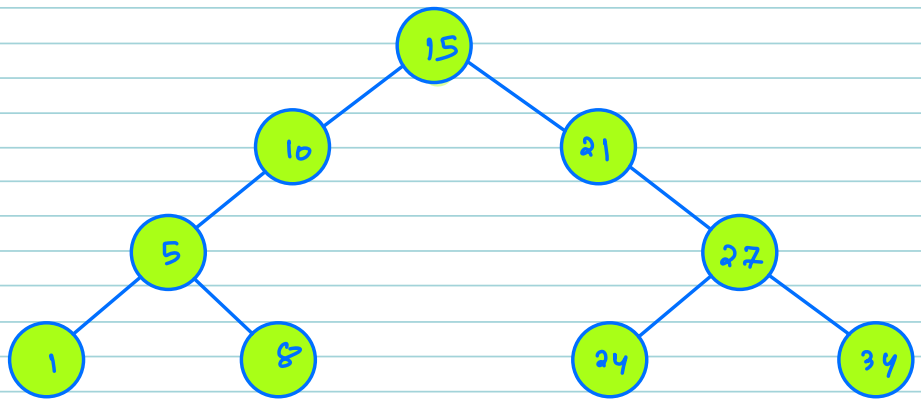
→ Not a BST

Ex4 :-



→ It is a BST

⇒ PROPERTY OF BST



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

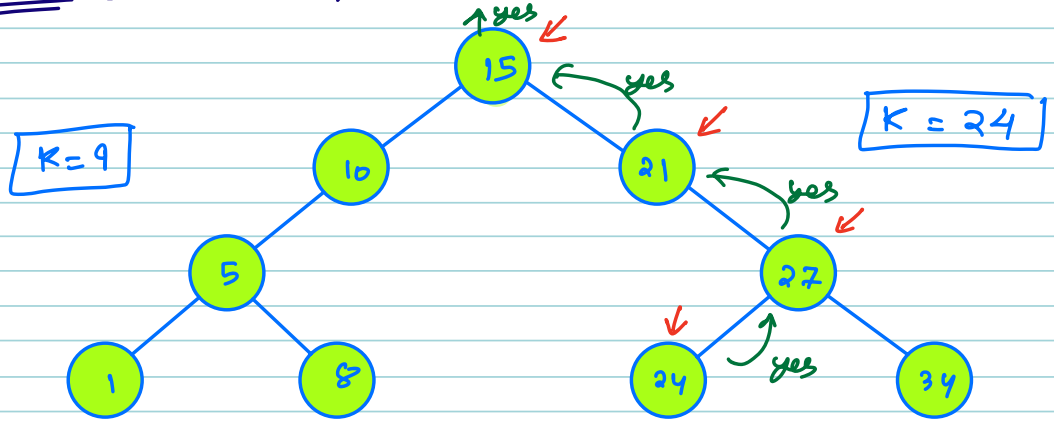
↳ observation :- Inorder of a BST is sorted.

(L D R)

Quiz ! :- what is a Binary Search Tree (BST) ?

↳ A tree where for a node x, everything on the left has data <= x & on the right > x.

Question:- Searching in a BST



Brute force

↳ Do traversal & check for Node == K.

TC →	O(N)
SC →	O(H)

=> OPTIMIZATION

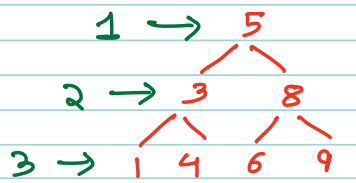
↳ Use the property that tree is a BST.

i.e. → If a Node > element, then move Left.

→ else if Node < element, then move right

→ else Got data.

Quiz 2 :- what is the no. of nodes you need to visit to find the number 1 in the following BST ?



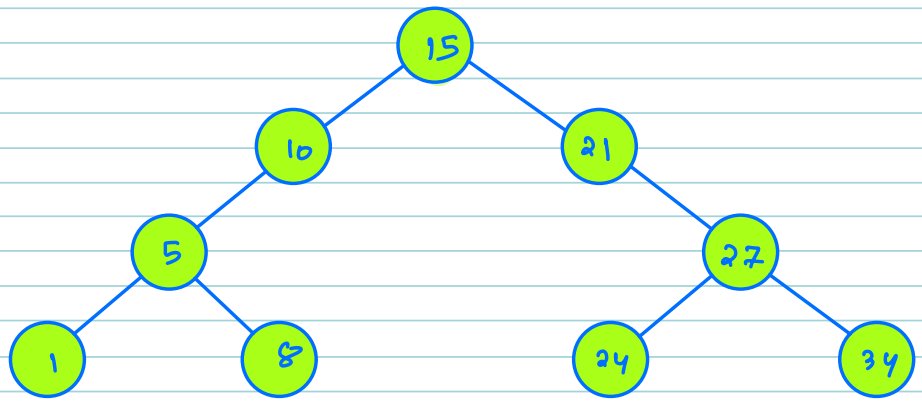
↳ 3 Ans

PSEUDO CODE

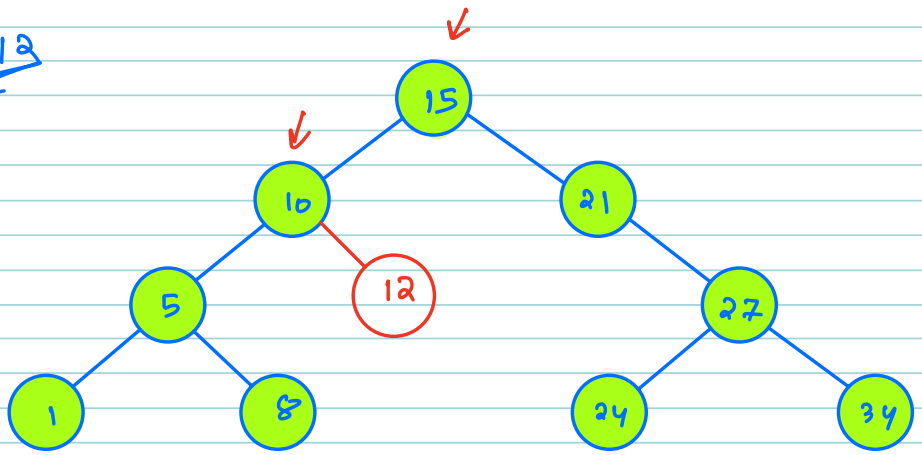
```
boolean Search ( Node root, int k){

    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);
    }
}
```

TC – O(H)
SC – O(H)

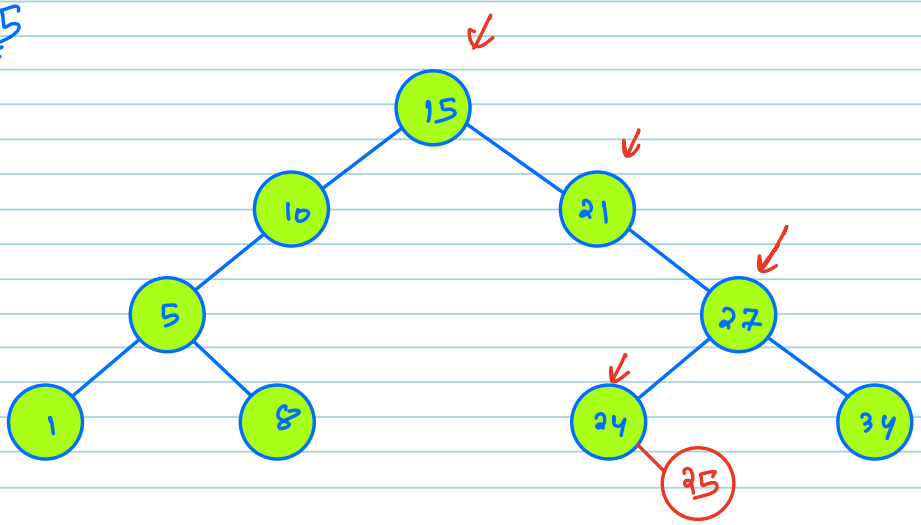
Question :- Insertion in a BST



K = 12



K = 25



IDEA :- ↳ If element > Node, call will be made to right to insert & expect root of find RST in return.

↳ If element < Node, call will be made to left to insert & expect root of find RST in return.

↳ If Node is null, create Node with value to insert & return it.

PSEUDO CODE

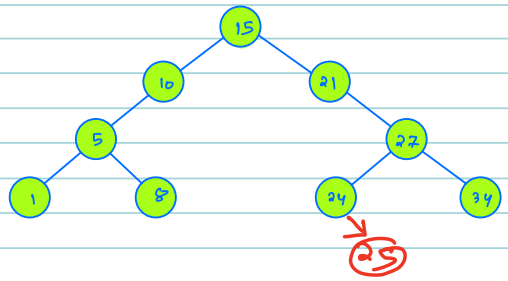
```
Node Insert ( Node root, int k){
    if ( root == Null) {
        return new Node (k);
    }

    if ( root.data > k) {
        root.left = Insert ( root.Left, k);
    } else {
        root.right = Insert (root.right, k);
    }

    return root;
}
```

DRY RUN

Node

Insert (15K, 25)

K = 25




```
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;
}
```

15K

=15K =25

21K

=21K =25

27K

=27K =25

24K

=24K =25

25K

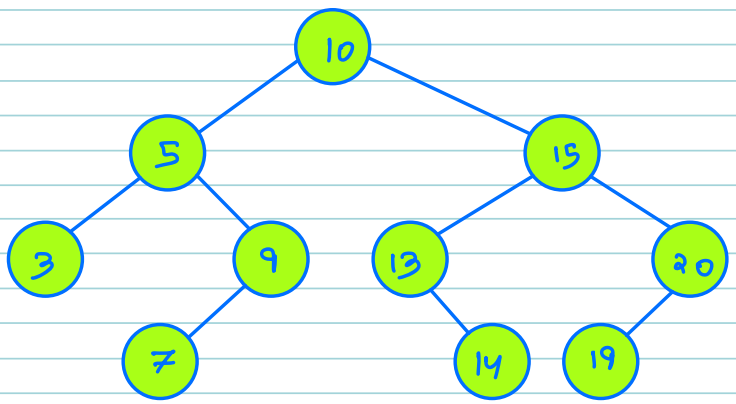
=NULL =25

```
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;
}
```

Quiz 3 :- where does the node with the smallest value resides in a BST ?

↳ We need keep on going Left till we get null. And there we will find the smallest Node.

question :- Min in BST



IDEA :- We Need to go as Left as possible, Not even single Right as it will make the Ans increased.

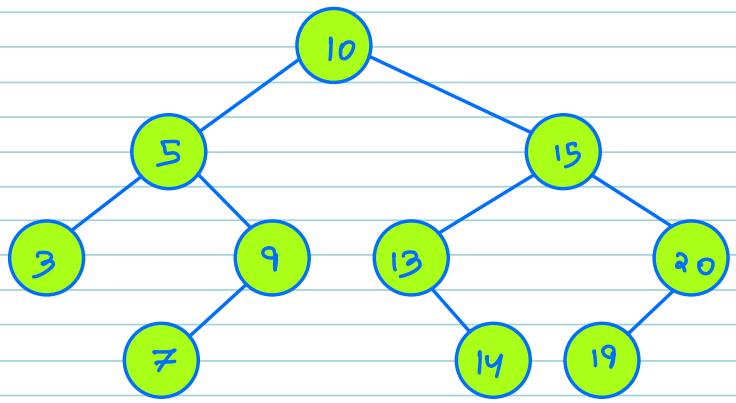
PSEUDO CODE

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

TC → O(H)

SC → O(1)

Question Max in B.S.T.



IDEA :- We Need to go as right as possible, Not even single Left as it will make the Ans decrease.

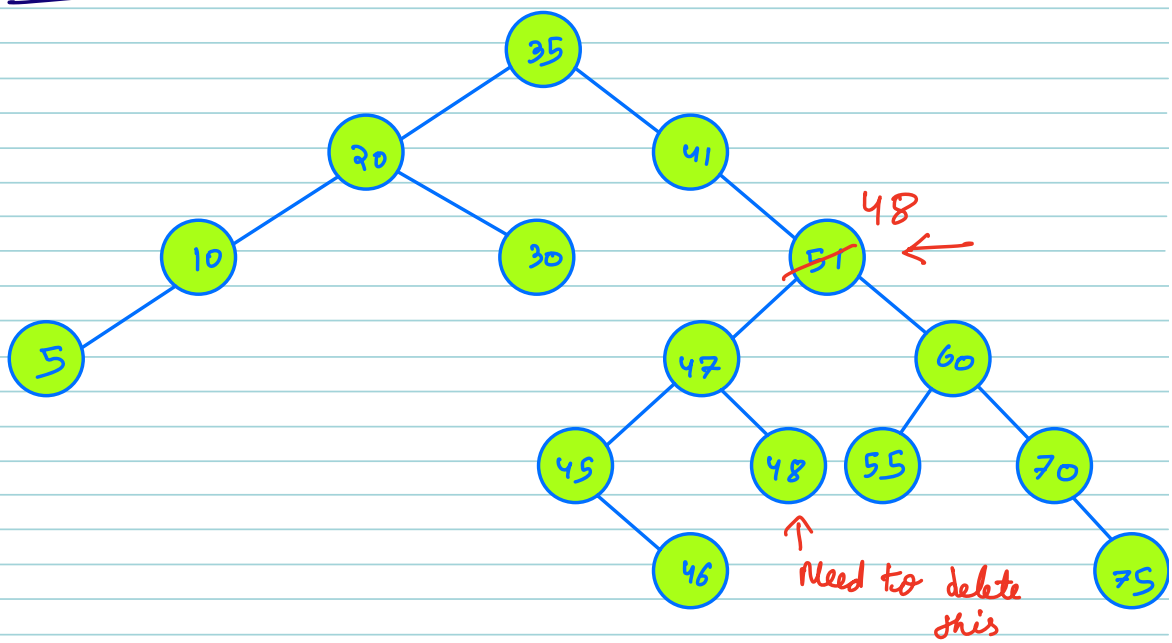
PSEUDO CODE

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

TC → O(H)
SC → O(1)

10:22 ← Resume time

Question :- Delete a node in BST



K = 5, K = 30 } 0 Children

K = 10, K = 41 } 1 child

K = 51 → 2 Children

Case I → 0 Children

IDEA :- → If Node > K, move Left & give responsibility of Deletion to Left & In return expect root of final LST with deleted Node.

→ If Node < K, move right & give responsibility of Deletion to right & In return expect root of final RST with deleted Node.

→ If Node.data == K, return Null;

Case II → 1 child

IDEA:- → If Node > K, move Left & give responsibility of Deletion to Left & In return expect root of find LST with deleted Node.

→ If Node < K, move right & give responsibility of Deletion to right & In return expect root of find RST with deleted Node.

→ If Node.data == K, then return the Non Null child.

Case III → 2 children

IDEA:- → If Node > K, move Left & give responsibility of Deletion to Left & In return expect root of find LST with deleted Node.

→ If Node < K, move right & give responsibility of Deletion to right & In return expect root of find RST with deleted Node.

Q If I want to replace 51 with any element in subtrees then which will be the best element ?

↳ Max in LST, because It will be greater than all the element on Left & smaller than all the element on right.

Alternative :- Min in RST

★ Post Selecting Node from LST, delete A Selected Node from LST.

PSEUDO CODE

```
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;
```

Case I

Case II

```
        else {
            int v = max(root.left);
            root.data = v;
            root.left = delete(root.left, v);
        }
    }
    return root;
```

TC → O(H)
SC → O(H)

BALANCED BST

↳ Its there so that the height of the tree can be constrained to (log N) because when BST is balanced the height is of (log N) order.

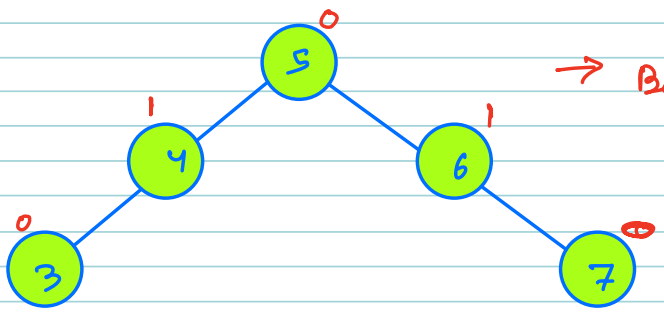
Quiz:- What is the purpose of balancing a Binary Search tree?

↳ To maintain efficient search, insert & delete operation.

Question :- Construct a Balanced BST using sorted array.

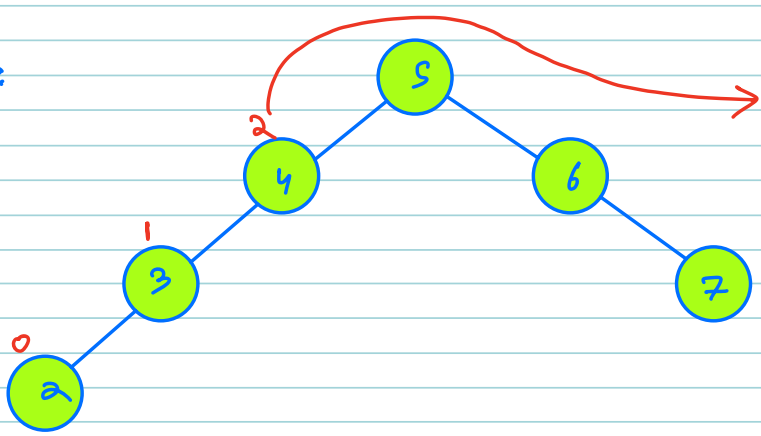
Balanced BST → | LST height - RST height | <= 1

Ex1 :-



→ Balanced BST

Ex2 :

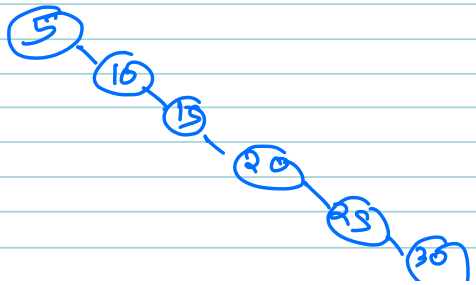


→ Imbalanced BST.

Ex :- 5, 10, 15, 20, 25, 30

↳ Create Balanced BST out of given array.

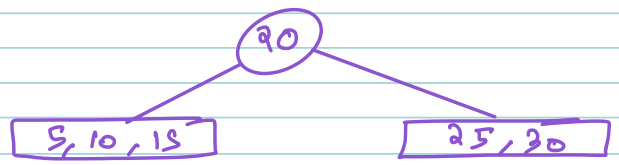
IDEA :- Call Insert Method on BST for every Node.

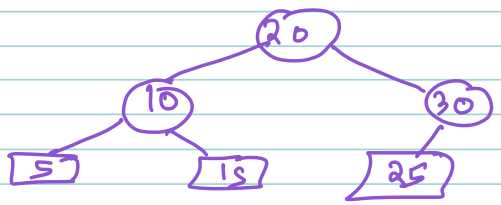


Problem :- A Tree formed is not a Balanced BST

IDEA? :- 5, 10, 15, 20, 25, 30

Q which Node will be best to be treated as root of Balanced BST ?





Q How we should write the code ?

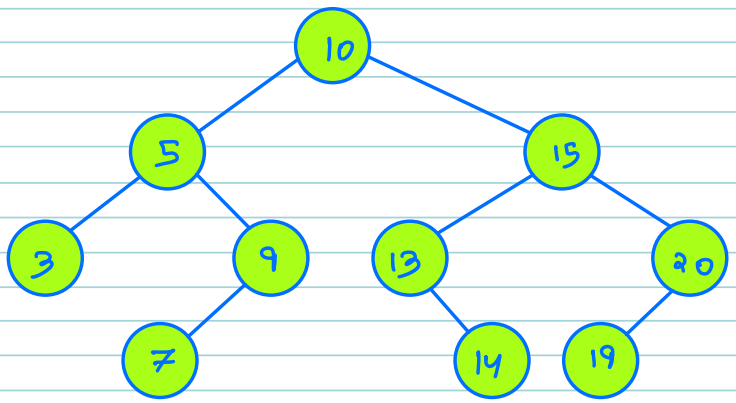
↳ Recursion

PSEUDO CODE

```
Node Create (int[] arr, int s, int e){
    if (s>e) { return null;
    m = s + (e-s)/2;

    Node root = new Node (arr[m]);
    root.left = Create (arr, s, m-1);
    rot.right : Create (arr, m+1, e);
    return root;
}
```

TC → O(N)
SC → O(log N)

Question :- Given BT, check whether it is a BST or not ?

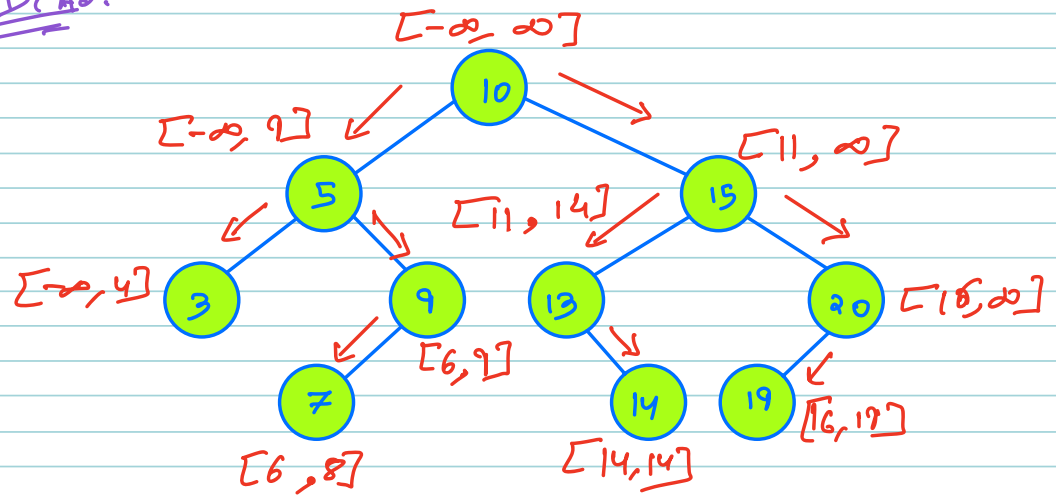


IDEA 1 :- Create Arraylist of Inorder for the given tree. And check if it is Sorted or Not.

TC — O(N)
SC — O(N)

↳ Code :- HW [must to try]

IDEA2.

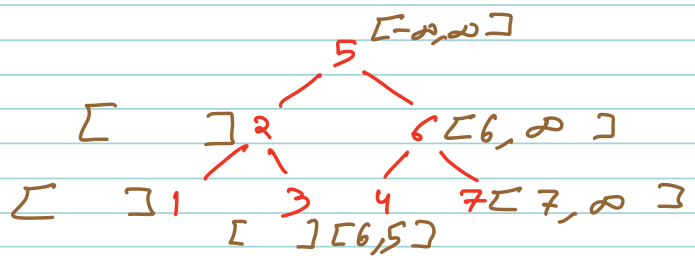


Steps :- ① when going left update right (i.e. max)

② when going right update left (i.e. min).

③ For tree to be a BST, every Node should be is a given range.

Quiz :- Check whether the given BT is BST or not?



↳ Ans No

PSEUDO CODE

```
boolean 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;
}
```

TC → O(N)
SC → O(H)