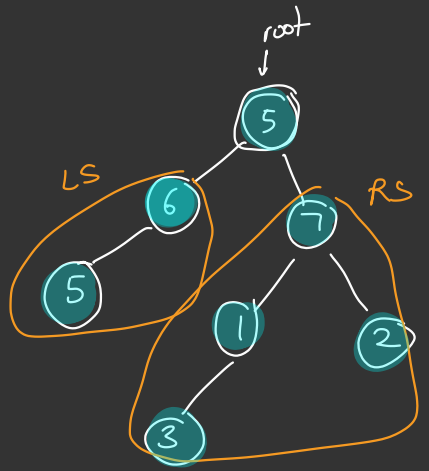
Binary Trees - 3



T = 3

Expensive ⇒ O(N)

O(N)

- → Preorder ✓
- → Inorder ✓
- → Postorder ✓
- → Levelorder ✓

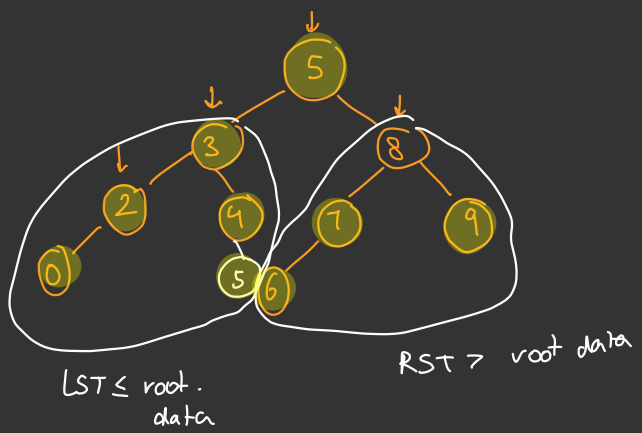
```
bool search (Node root, int T) {
    if (root == NULL)
        return false

    if (root.data == T || search (root.left, T) ||
        search (root.right, T))
        return True;

    return false;
}
```

Binary Search Tree

→ optimised for searching

Problem → store Duplicates

5, 3, 8, 7, 2, 6, 0, 9, 4, 5



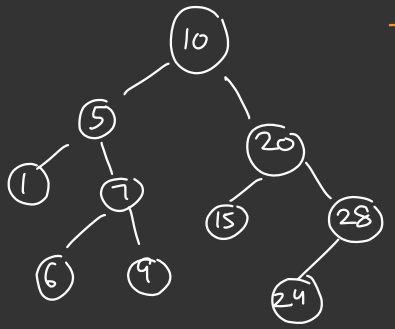
Property

data ≤ root data < data

LST — RST

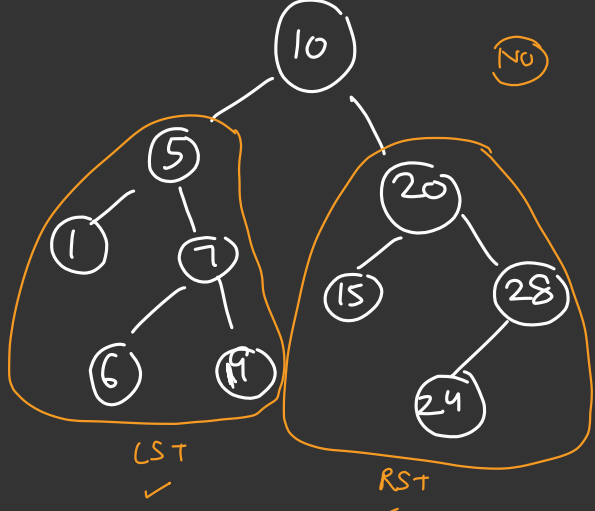
→ Always go left

→ Search in the left.



Tree-1

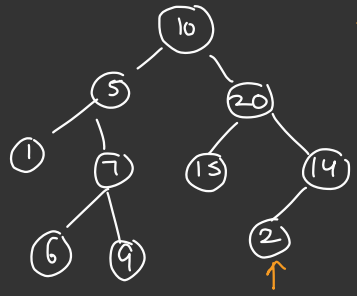
Yes



No

LST

RST

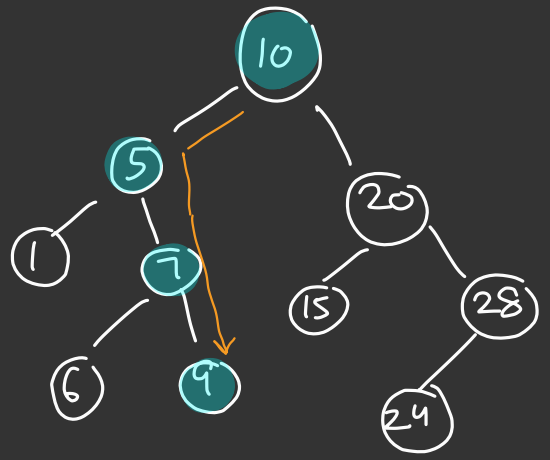


Tree-2

No

Advantage of BST

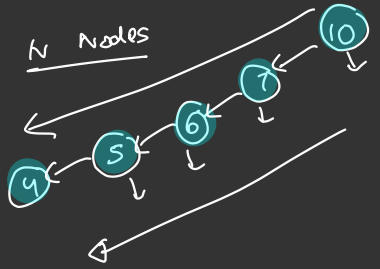
↳ faster searching



Search `t = 9`

Time to Search → $O(H)$ = Root to Leaf

$$\log_2 N \leq H \leq N$$



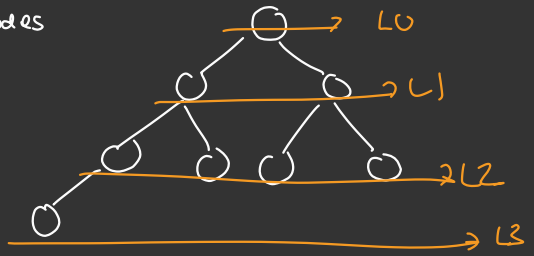
Skewed BST

$O(\log N) \leq O(H) \leq O(N)$

Balanced BST Tree

Skewed Tree

8 Nodes



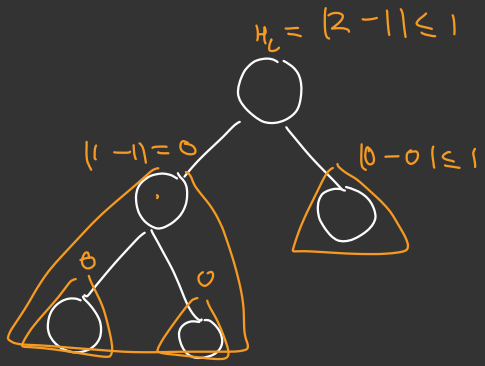
Height balanced Tree $H = O(\log N)$

Balanced BT / BST

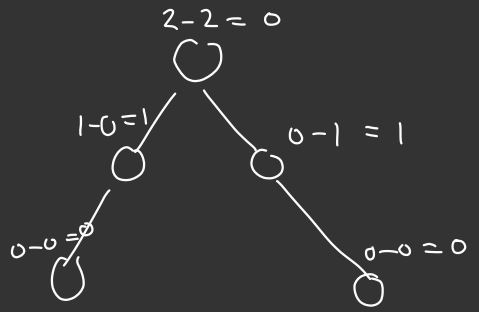
A BT/BST is height balanced if height of LST and height of RST at all node differ by atmax 1.

$$\text{abs}\left|H_{LST} - H_{RST}\right| \leq 1$$

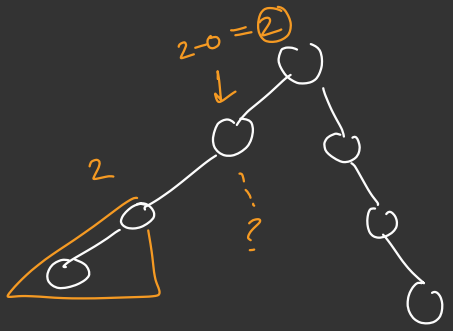
Height of HB Tree → $\log N$



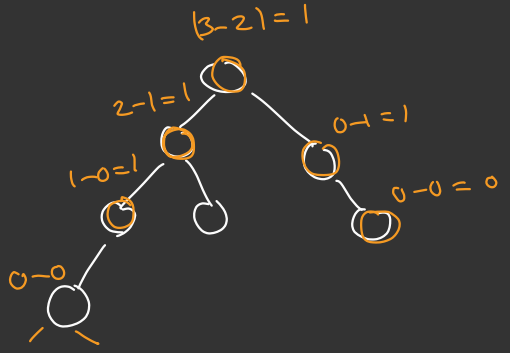
HB = Yes



Yes



T= No



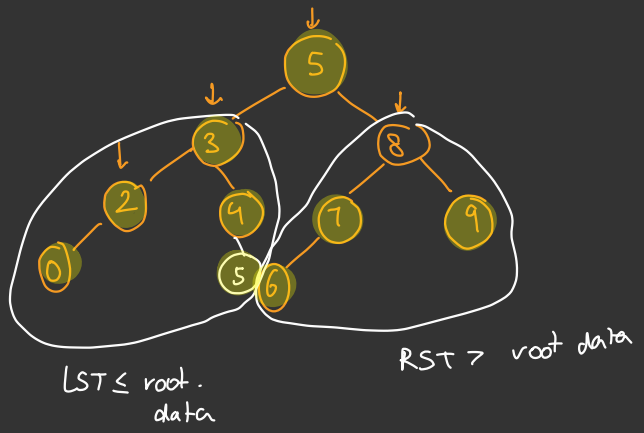
Yes

Ex-1 given N Inputs, write a function to construct BST 4 Mins

Ex-2 In above BST, write a function to search in BST 4 Mins

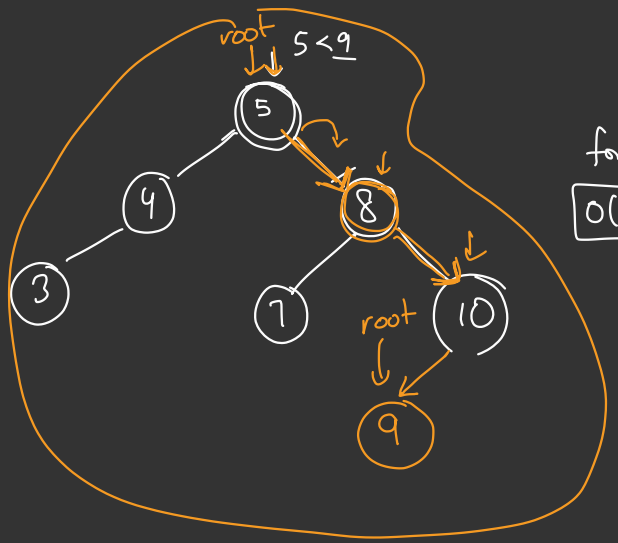
5, 3, 8, 7, 2, 6, 0, 9, 4, 5

9 4 5



Live ToDo

```
Node makeBST() {
    ||||||
}
Node searchBST(d) {
    ||
}
```



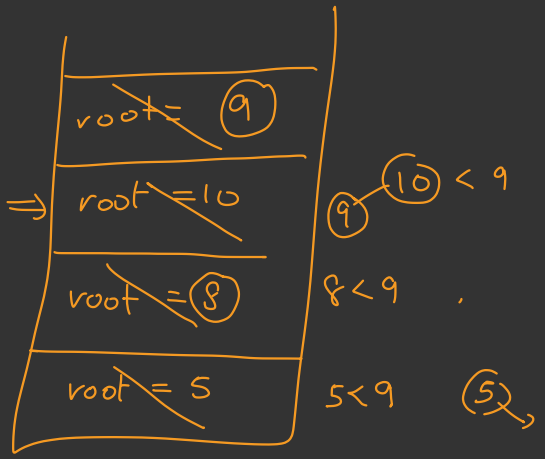
for each node

O(H)

Insert - (9) ← key

→ TODO: ith insert

```
Node Insert(Node root, int key){
    if(root == null){
        root = new Node(key)
        return root
    }
    if(root.data < key){
        root.right = insert(root.right, key)
    }
    else {
        root.left = insert(root.left, key)
    }
    return root
}
```



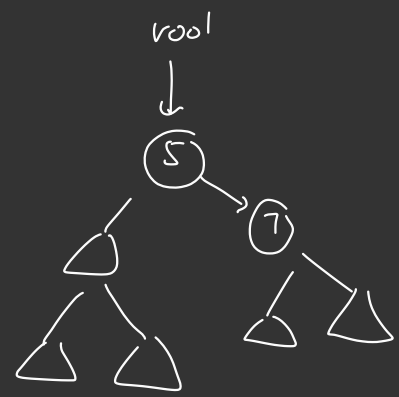
```
Node makeTree( ) {
    n = input()
    Node root = null
    for( i=0 ——— n-1) {
        data = input()
        root = insert( root, data);
    }
    return root
}
```

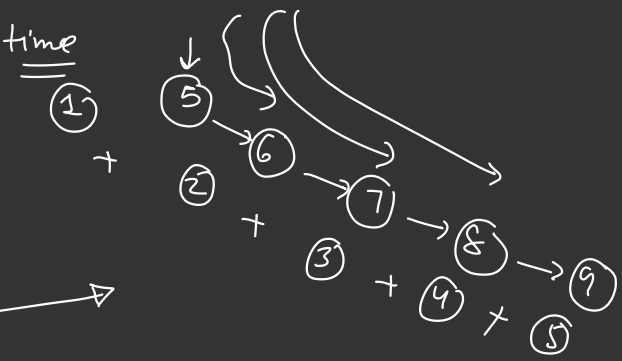
O(N H)

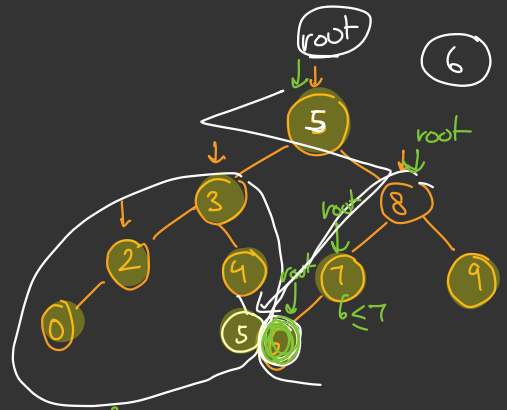
↓

worst case

$O(N^2)$







6≤8

6≤5

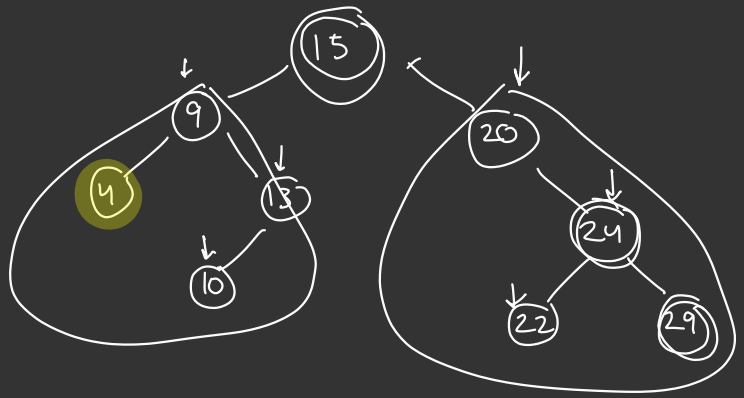
Itr search

```
bool search (Node root, int key){

    while (root != null){                    O(H)
        if (Key == root.data)
            return True
        else if (key ≤ root.data)
            root = root.left
        else
            root = root.right
    }
    return false
```

extra stack space.

```
bool searchRec (Node root, int Key){
    if (root == null)
        return false
→   if (root.data == key){ return True },
    else if (Key ≤ root.data)
        return searchRec(root.left, key)
    else
        return searchRec(root.right, key)}
}
```

Special Thing ⇒ BST



Inorder ⇒

Left | Root | Right

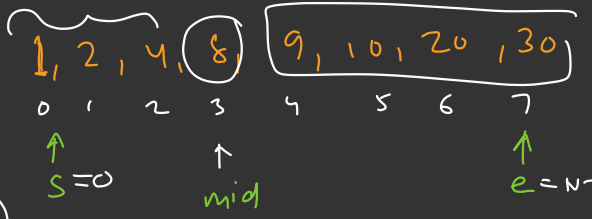
Small | | Big.

at every node

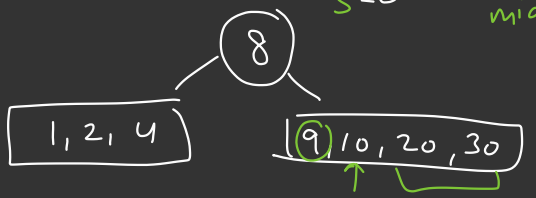
4, 9, 10, 13, [15], 20, 22, 24, 29

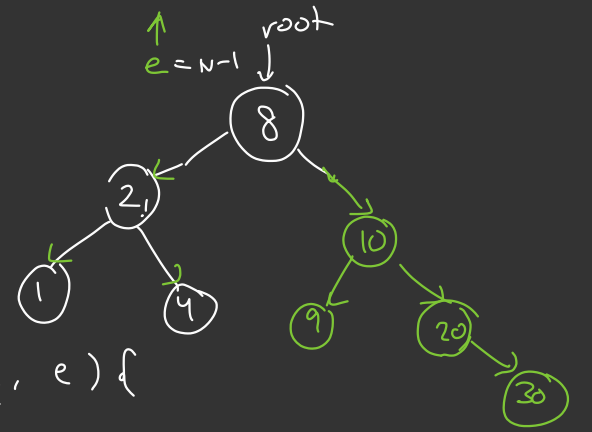
Sorted [Always]

Q Given a Sorted array, create a height balanced BST from it



mid = (s+e)/2

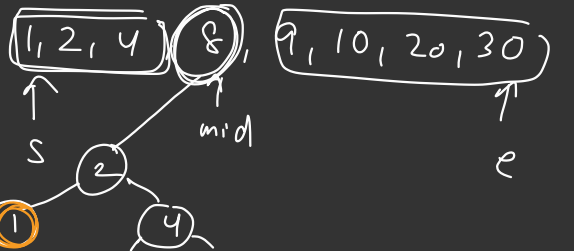





```
Node createBST ( arr[], s, e ) {
// Base case
if (s>e) {
    return null;
}
```

Time → O(N)

Space → O(H)



// Rec Case

mid = (s+e)/2

Node root = new Node(arr[mid])

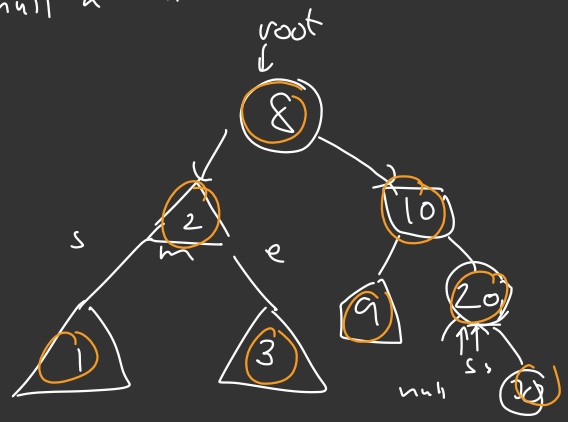
root.left = createBST(arr, s, mid-1)

root.right = createBST(arr, mid+1, e),

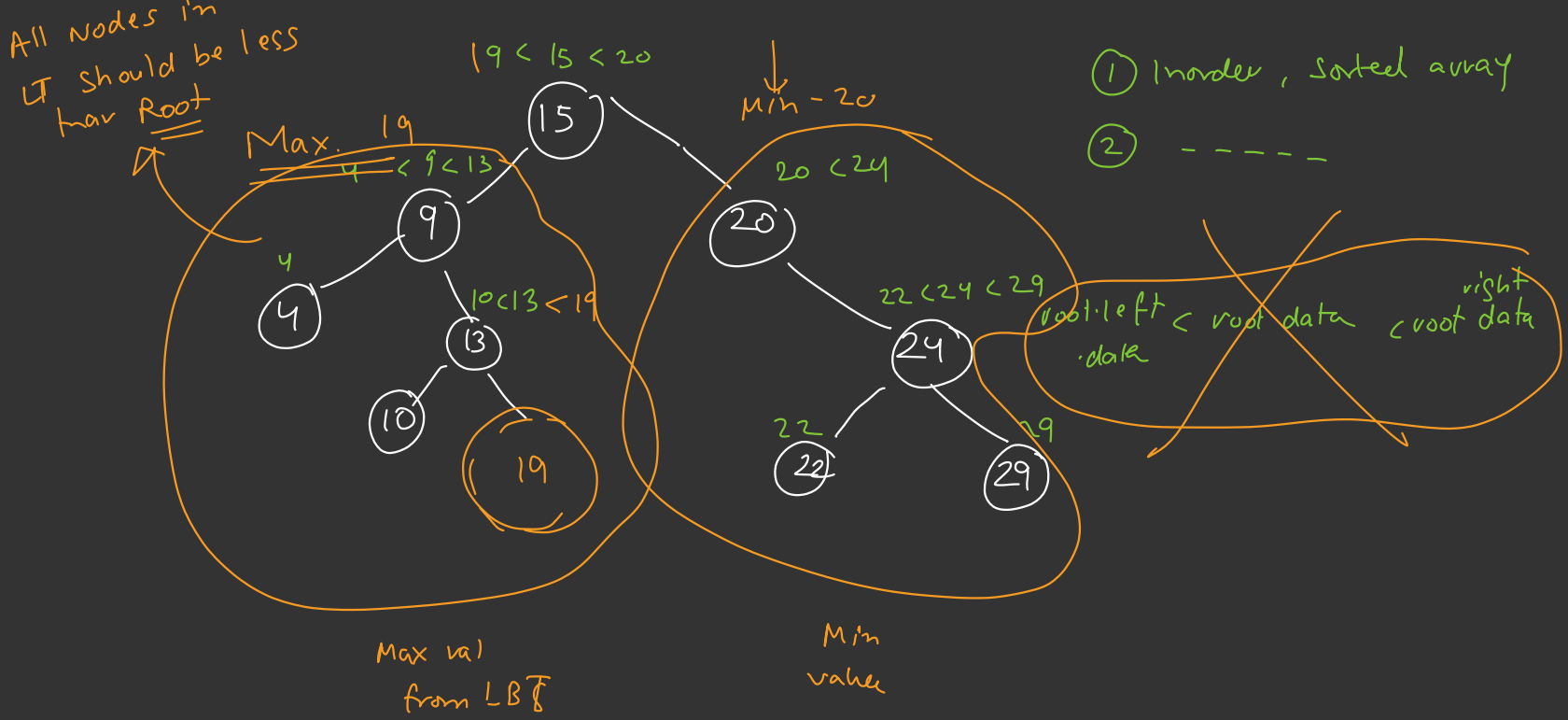
return root;

Preorder

}



⑫ Check if a BT is a BST



① Inorder, sorted array

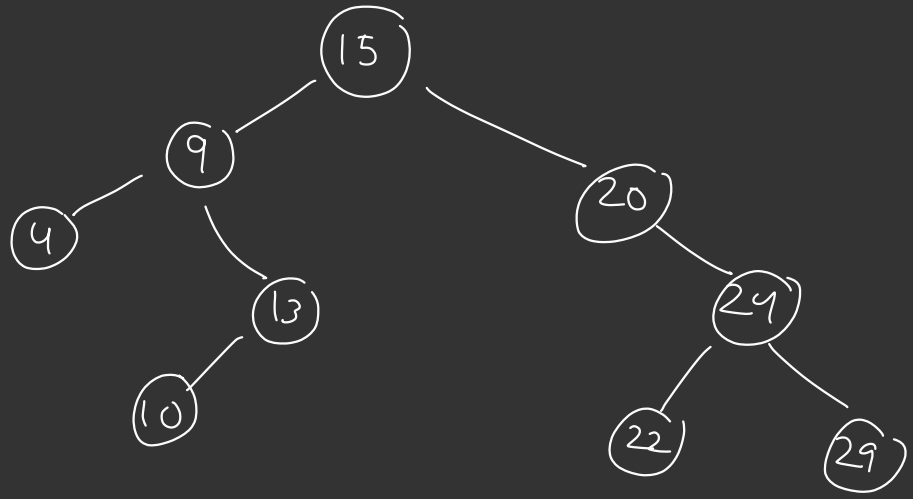
② -----

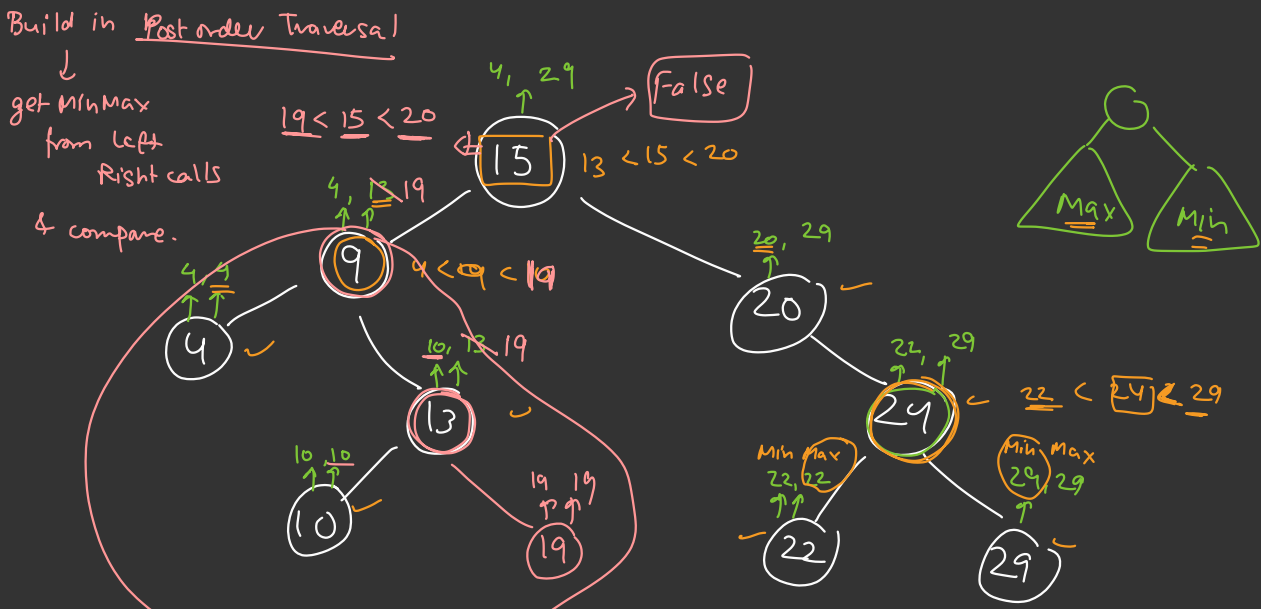
BST

Every Node Satifies prop

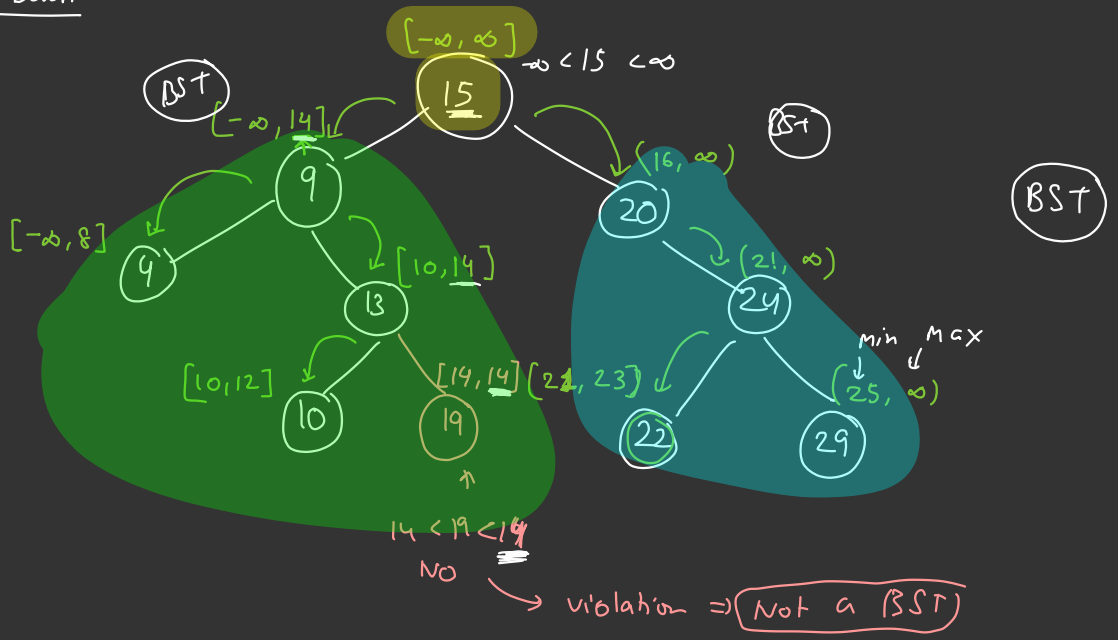
All Nodes in LST ≤ node data

All nodes in RST > node data

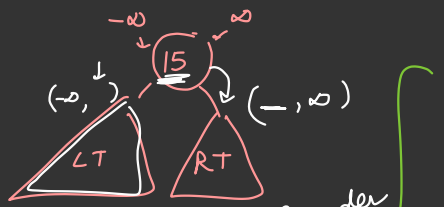



Build in Post order Traversal
get MinMax from left Right calls & compare.
19 < 15 < 20
4, 29
False
15
13 < 15 < 20
4, 19
9
4 < 9 < 19
4
10, 19
13
10, 10
10
19, 19
19
20, 29
20
22, 29
24
22 < 24 < 29
Min Max
22, 22
22
Min Max
29, 29
29
Max
Min

Top-Down


[-∞, ∞]
-∞ < 15 < ∞
15
BST
[-∞, 14]
9
BST
(16, ∞)
20
BST
[-∞, 8]
4
[10, 14]
13
(21, ∞)
24
min
max
[10, 12]
10
[14, 14]
19
[21, 23]
22
(25, ∞)
29
14 < 19 < 14
NO
violation ⇒ Not a BST

```
bool isBST(Node root, int minV, int maxV){
    if(root==null){
        return True;
    }
    if((root.data > minV && root.data < maxV) &&
        isBST(root.left, minV, root.data-1)
        && isBST(root.right, root.data+1, maxV)){
        return True;
    }
    return false
}
```



Preorder

O(N) ← time

O(H) space