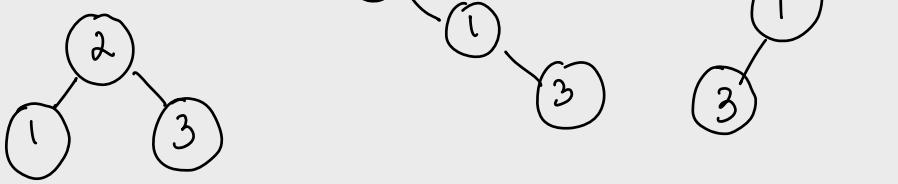
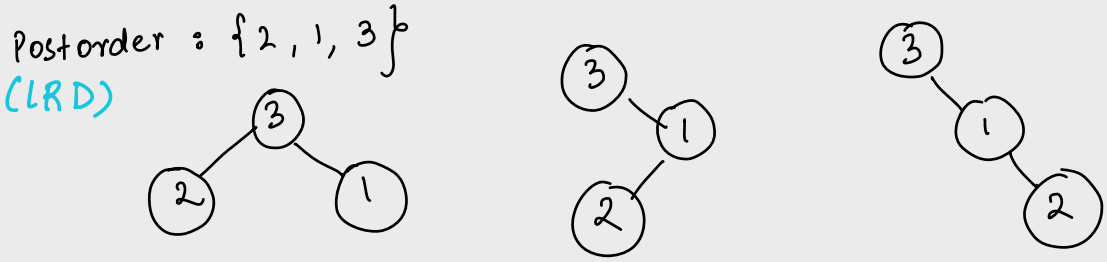
Construct the binary tree

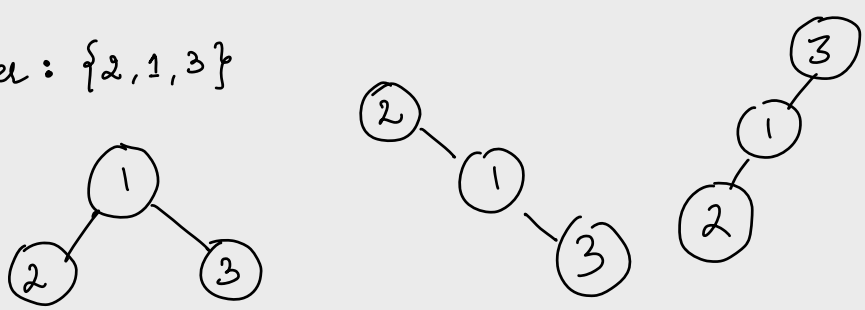
Preorder : {2, 1, 3}
(DLR)



Postorder : {2, 1, 3}
(LRD)

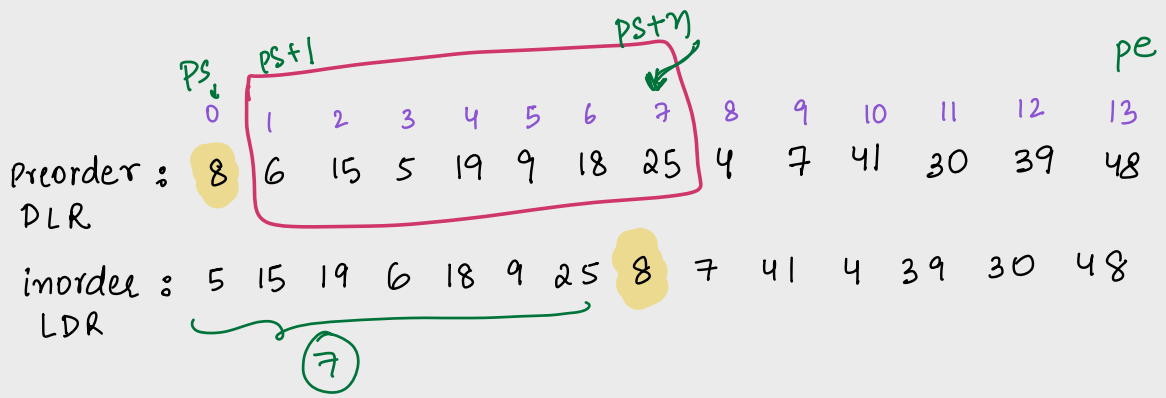


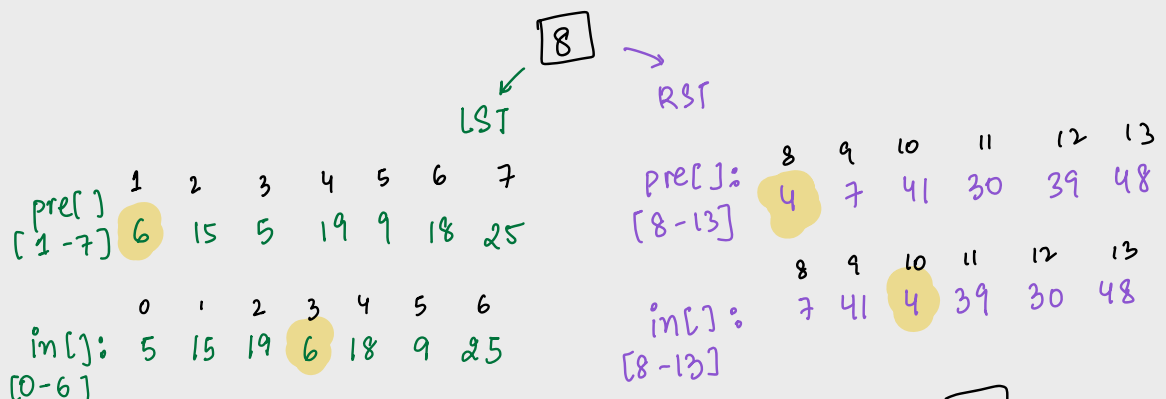
Inorder : {2, 1, 3}
LDR

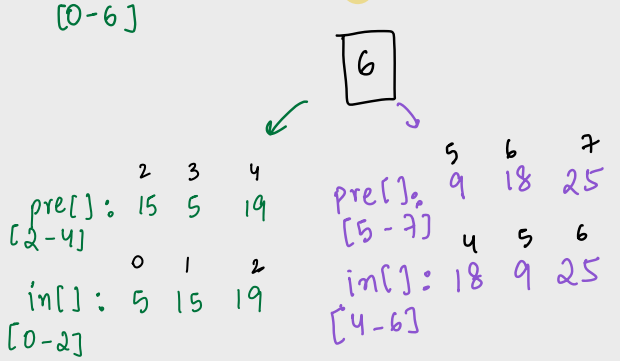


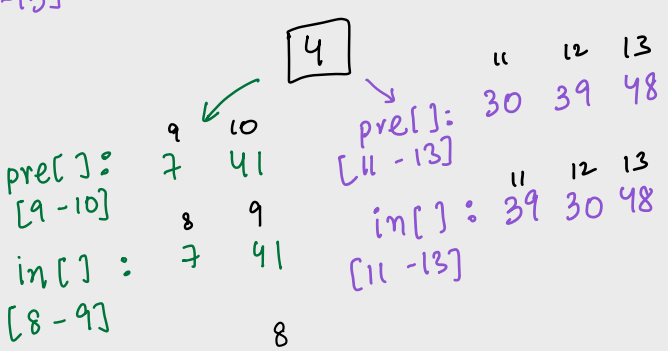
(DLR) (LRD)
Given Preorder & Postorder, can we construct a tree?

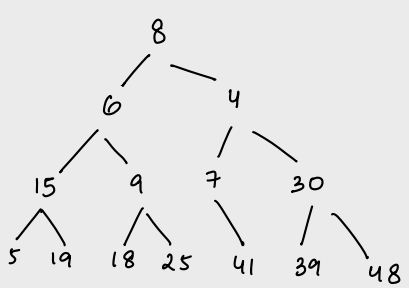
Que: Given inorder & preorder traversal, construct Binary Tree. [data will be unique]











Idea:

1. In preorder, first ele is the root node
2. Find root node in inorder
3. Get the no. of ele in LST from inorder
4. Divide Pre[] & In[]

Use Hash Map

< ele, index >

```
for(i=0; i<N; i++){
    hm[in[i]] = i
}
```

main(){

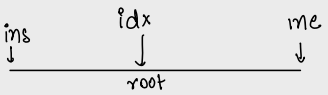
// Assumption : Given pre[] & in[], your code will Construct the BT. & return the root.

CBT(pre[], 0, N-1, in[], 0, N-1)

MICROSOFT

```
Node CBT(pre[], ps, pe, in[], ins, ine){

    // Base case
    if(ps > pe) return NULL

    // main logic
    Node root = new Node(pre[ps])
    // search root in inorder
    int idx;
    for(i=ins; i<=ine; i++){
        if(in[i] == root.data){
            idx = i
            break
        }
    }

    n = idx - ins

    // Function call
    root.left = CBT(pre[], ps+1, ps+n, in[], ins, idx-1)

    root.right = CBT(pre[], ps+n+1, pe, in[], idx+1, ine)

    return root
}
```

idx = hm[root.data]



in[]: [ins idx-1]

// n = idx - ins

= idx - ins

pre[]: [ps+1 ps+n]

RST

in[]: [idx+1 ine]

pre[]: [ps+n+1 pe]

Hashmap

$T(N) = 2T(N/2) + 1$

$T(N) = T(N-1) + 1$

$O(N)$

Recurrence

$T(N) = 2T(N/2) + N$ [Best] $\Rightarrow O(N \log N)$

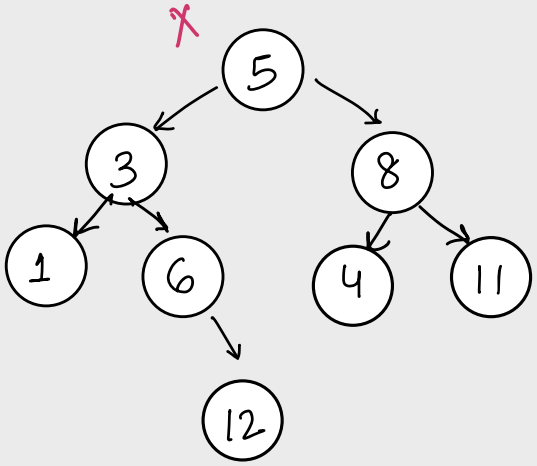
$T(N) = T(N-1) + N$ [Worst] $\Rightarrow O(N^2)$

Binary Search Tree [BST]

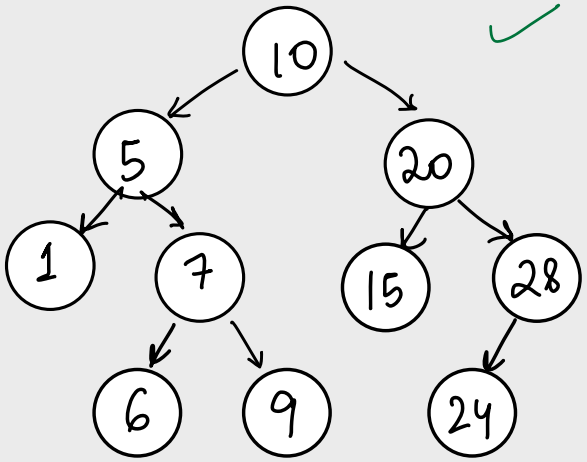
↳ helps in searching

Definition: For all nodes

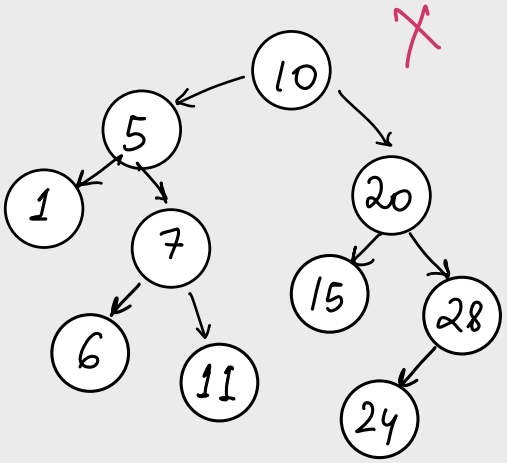
ALL LST < root < ALL RST.



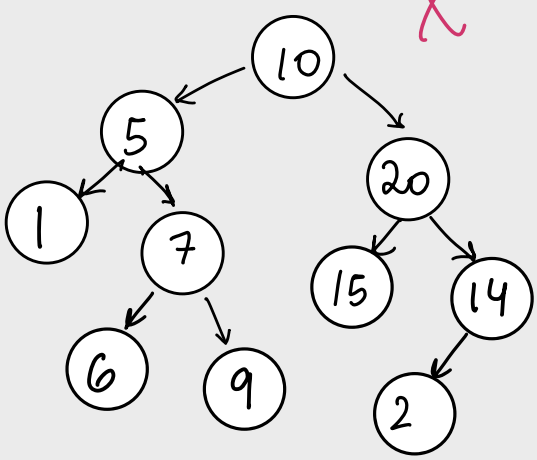
✗



✓



✗



✗

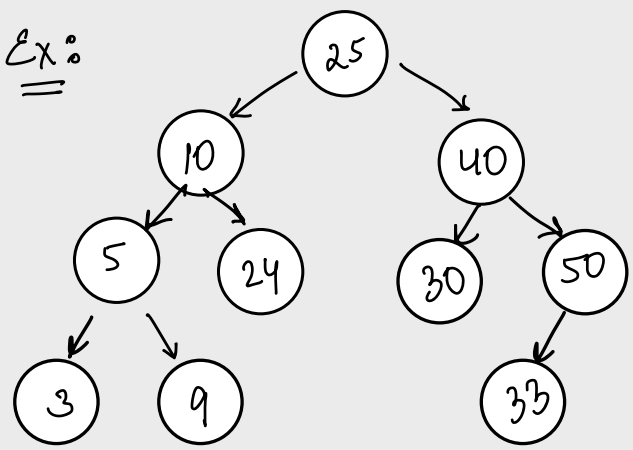
⊃

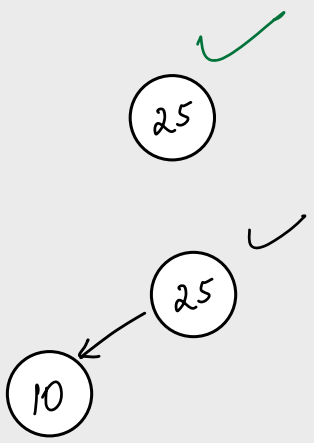
For all Nodes,

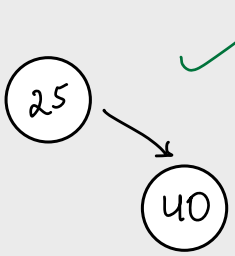
All LST < Node.data < All RST

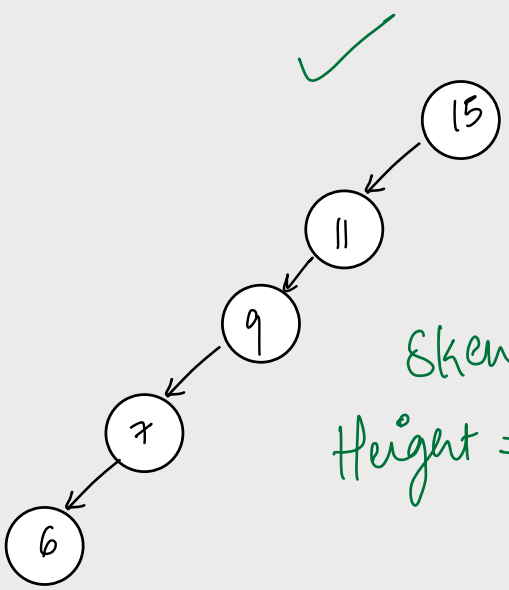
OR

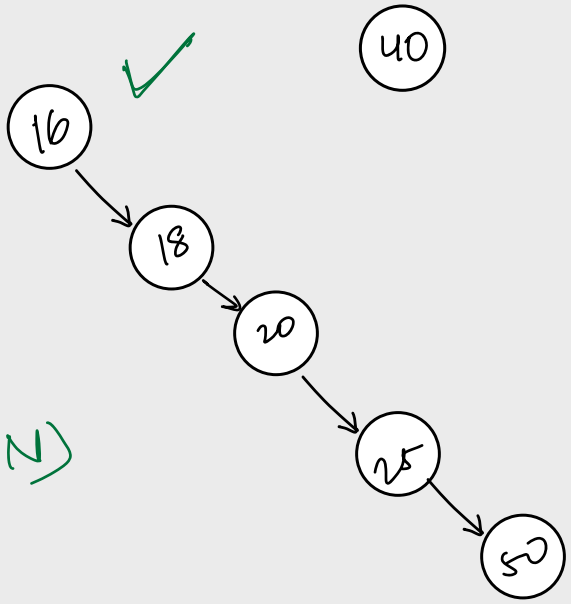
max(LST) < Node.data < min(RST)







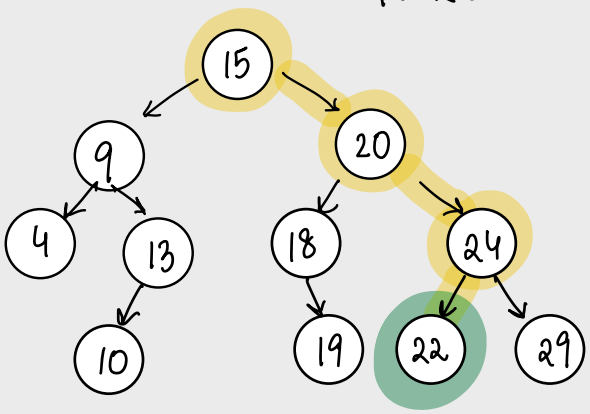




Search in a BST.

K = 16 ✗

K = 22 ✓



Search 22

root.data = 15

15 < 22 // go to right

root = root.right

root.data = 20

20 < 22

// go to right

root.data = 24

24 > 22

// go left

root.data = 22

// Iterative

```
bool search (Node root, int K){
    while (root != NULL){
        if( K == root.data){
            return true
        }
        if( K < root.data){
            //go left
            root = root.left
        }
        else root = root.right
    }
    return false
}
```

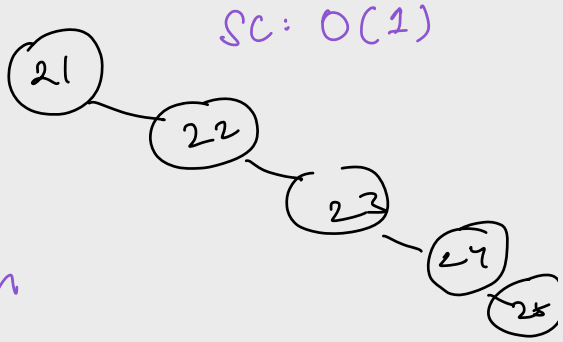
TC: O(log N)

SC: O(1)

Skewed Tree

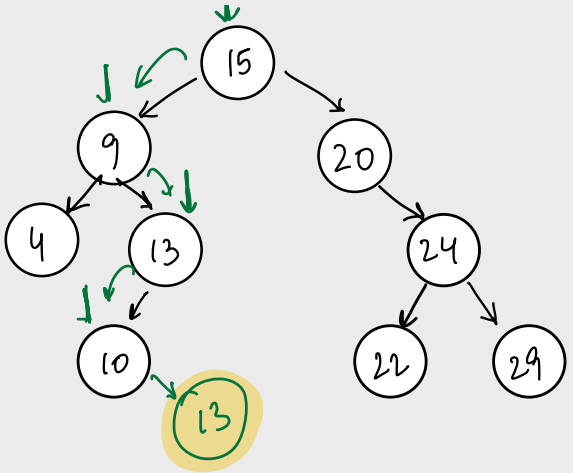
TC: O(N)

SC: O(1)



Note: Write using Recursion

Insertion



K = 13

insert 23 ⇒

values are distinct

prev = NULL
root.data = 15 15<23
prev = 15 go to right
r.data = 20 20<23
prev = 20 go to r
r.data = 24 24>23
pre = 24 go to l
r.data = 22 22<23
prev = 22 go to r
r.data = NULL

```
if(root == NULL) {
    root = new Node(K)
    return root
}
prev = NULL
while(root != NULL){
    prev = root
    if( K ≤ root.data) {
        //go left
        root = root.left
    }
    else root = root.right
}
if ( K < prev.data ) {
    prev.left = new Node(K)
}
else prev.right = new Node(K)
return root
```

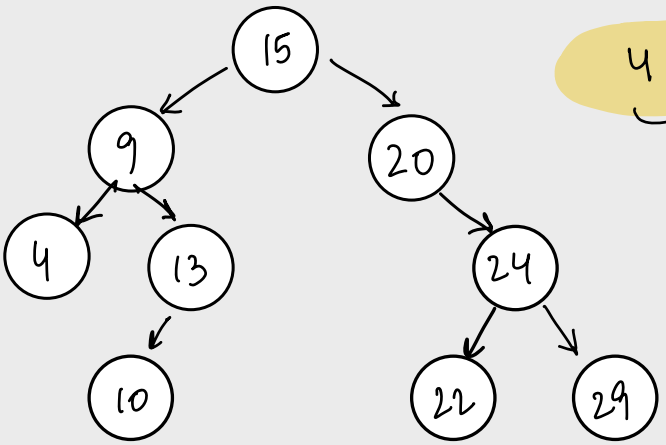
TC : O(Height)
SC : O(1)

For duplicates

↓

LST ≤ root < RST

Special BST property



inorder :

LDR

4 9 10 13 15 20 22 24 29

⇓

sorted

if BT is a BST.

Q. Given BT, check if it's a BST

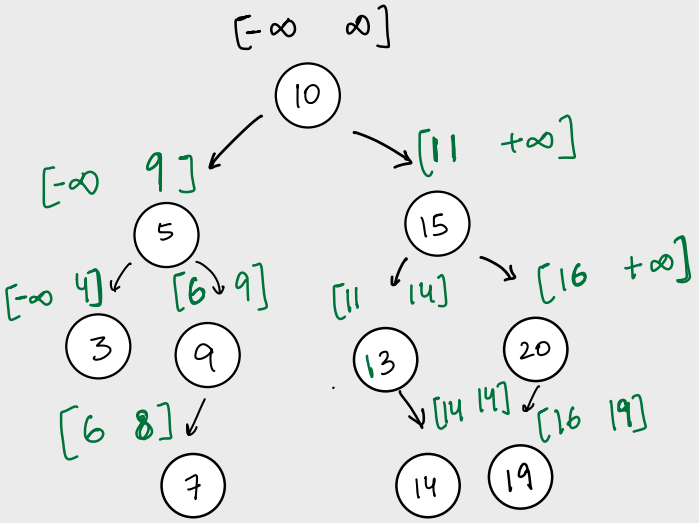
Approach 1 : Find its inorder.
If sorted, return true
else return false.

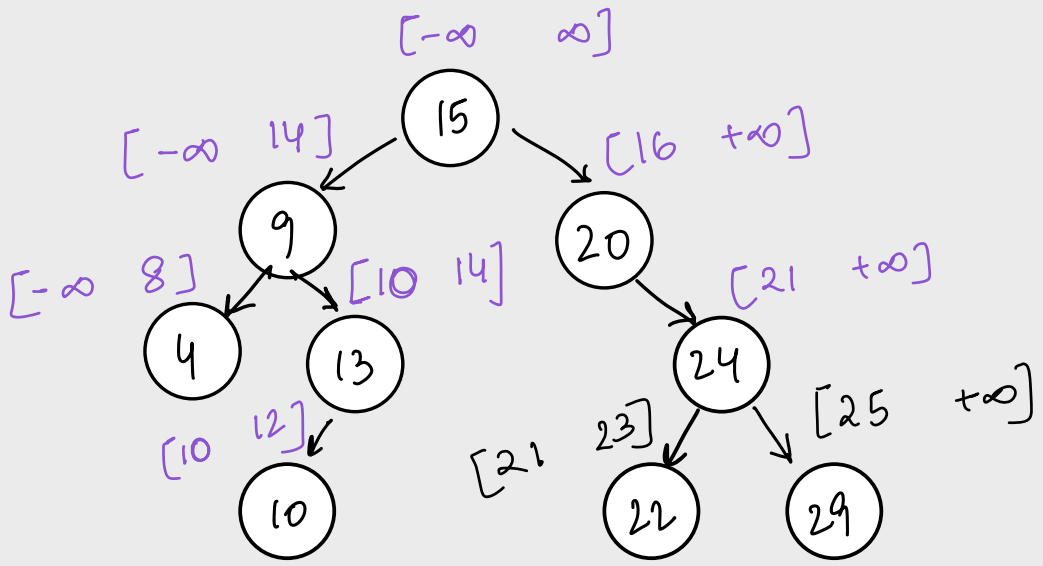
TC : O(N)

SC : O(N)

Approach 2

Max(LST) < Node.data < Min(RST)






```
//main => isBST(root, -∞, +∞)

bool isBST (root, maxL, minR) {
    if (root == NULL) return true
    if (maxL < root.data && root.data < minR) {
        return (isBST(root.left, maxL, root.data -1) &&
                isBST(root.right, root.data +1, minR))
    }
    else return false
}
```

TC: O(N)
SC: O(H)