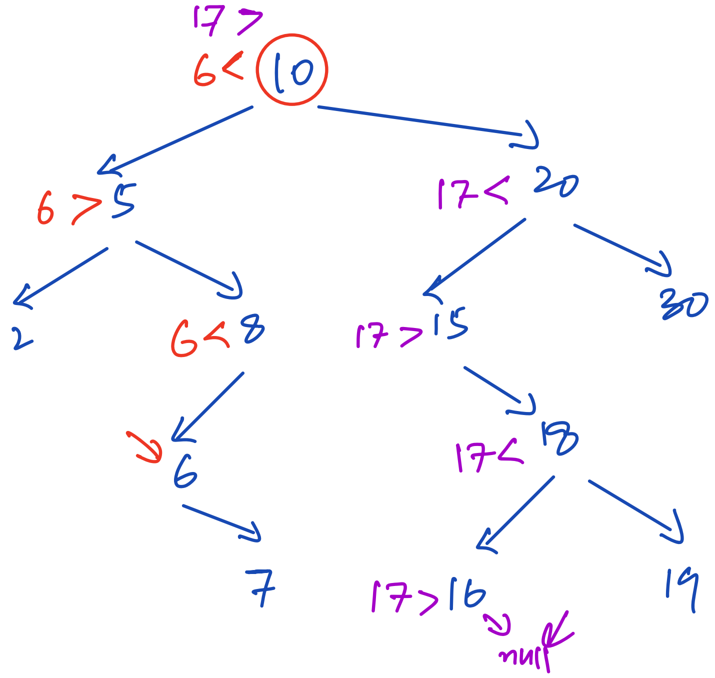
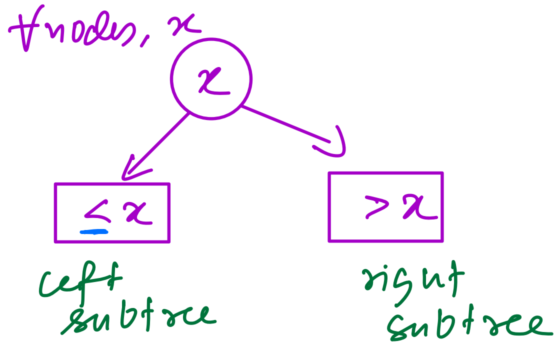
Trees 3 : BST

Binary Search Tree



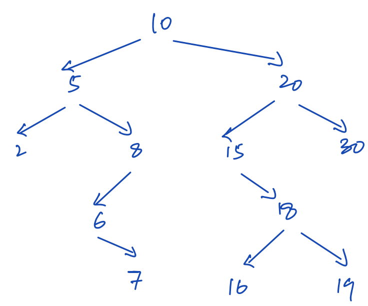


Equality can be exactly on one side.

In our eg., equality will be in left subtree.

searching

find (6) → true
find (17) → false } TC = O(H)

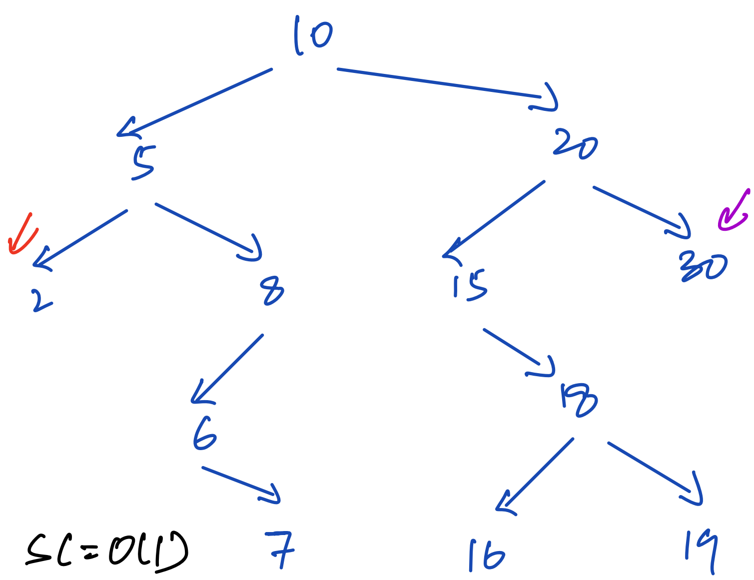


Ques → find smallest element in BST ?

ans = 2

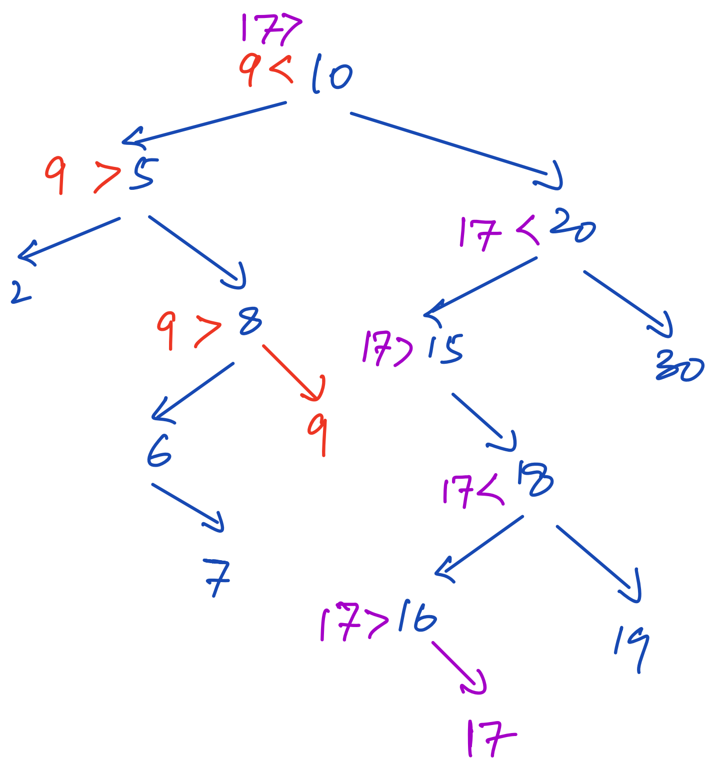
```
if(root == null) return -1   invalid value
temp = root
while ( temp.left != null)
    temp = temp.left
return temp.data
```

TC = O(H)   SC = O(1)



find largest element → 30

Insertion → search + insert



insert(9)

insert(17)

```
newNode = new TreeNode(x)
if (root == null)   return newNode   // new root
temp = root
while (true) {
    if (x <= temp.data) {   // go left
        if (temp.left == null) {
            temp.left = newNode
            return root
        }
```

```
        temp = temp.left
    }
    else { // go right
        if(temp.right == null) {
            temp.right = new Node
            return root
        }
        temp = temp.right
    }
}
```

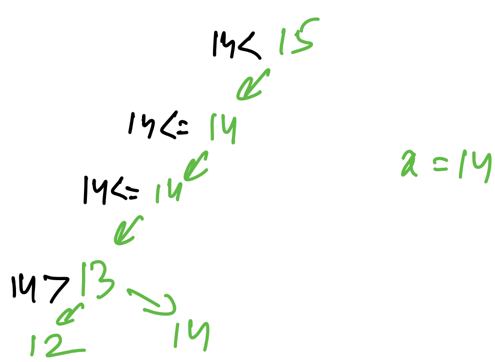
TC = O(H)
SC = O(1)

---

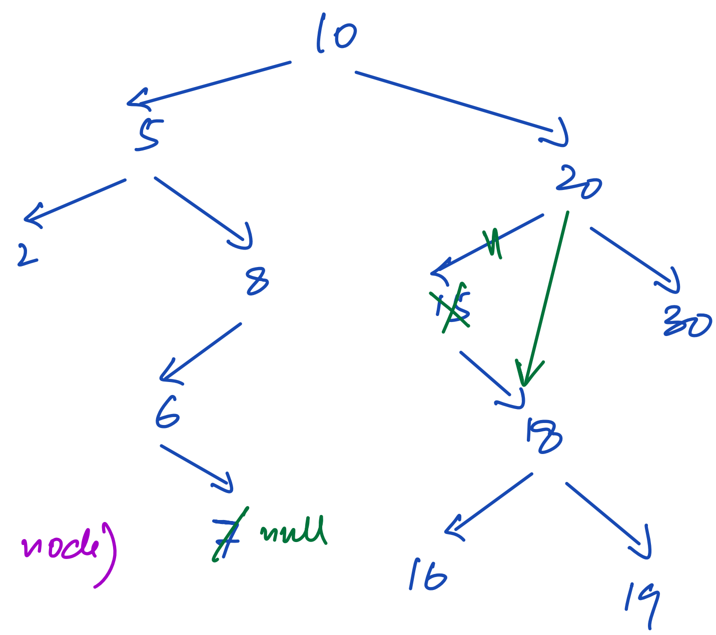
use recursion

```
TreeNode insert( root, x ){
    if (root == null)   return new TreeNode(x)
    if( x <= root.data) {
        root.left = insert( root.left , x)
    }
    else {
        root.right = insert( root.right , x)
    }
    return root
}
```

TC = O(H)   SC = O(H)



Deletion in BST



Steps 1. Search for node to delete and stop at its parent.

2. Delete Cases

a) Node has 0 children (leaf node)

delete(7)

update the link of parent to null

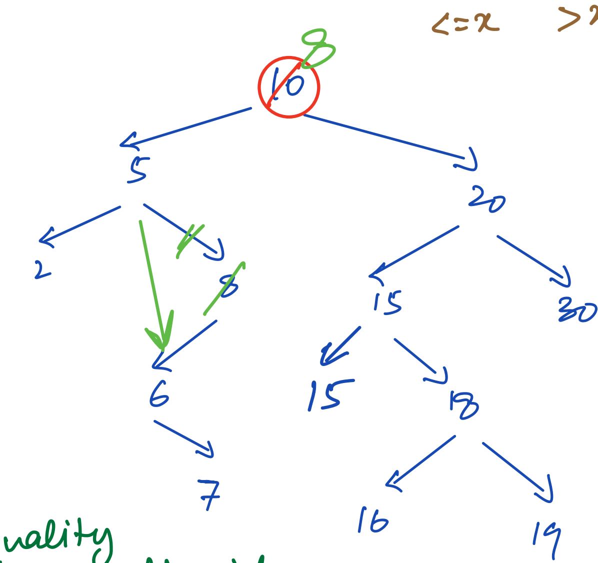
b) Node has 1 child

delete(15)

update the link of parent to the only child

c) Node has 2 children

delete(10)



i) Replace node to delete.
with

largest element of left subtree ✓ ∵ equality on left side

smallest element of right subtree ✗

both are correct if distinct elements are present

ii) Replaced node can be deleted via case (a) or (b).

$TC = O(H) + \text{either } (O(1) \text{ or } O(H))$

$= O(H)$

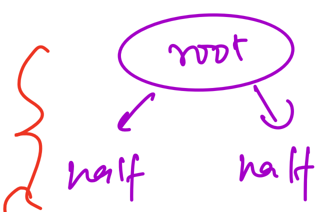
$SC = O(1)$ for iterative / $O(H)$ for recursive

|height of left − height of right| <= 1

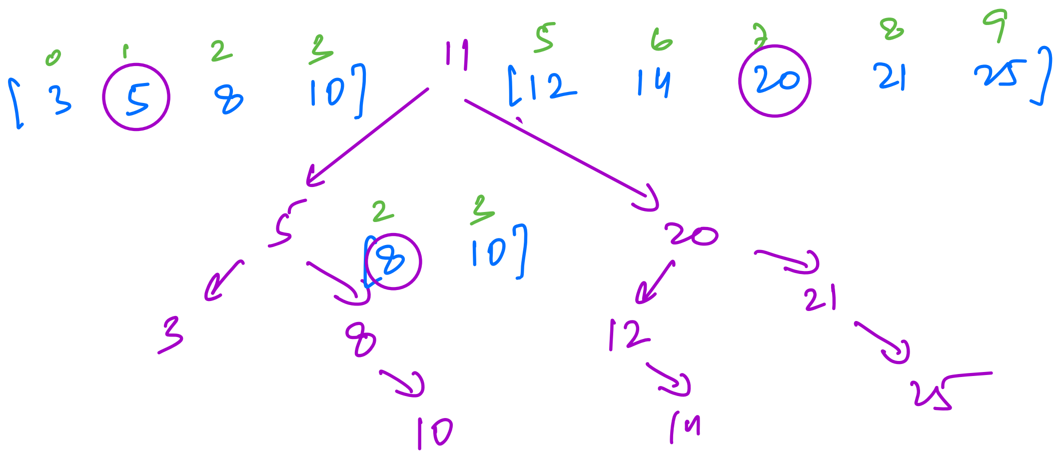
Ques → Construct height balanced BST from sorted array.

eg → A = [3 5 8 10 11 12 14 20 21 25]

0	1	2	3	4	5	6	7	8	9
3	5	8	10	11	12	14	20	21	25



∀ nodes ⇒ recursion



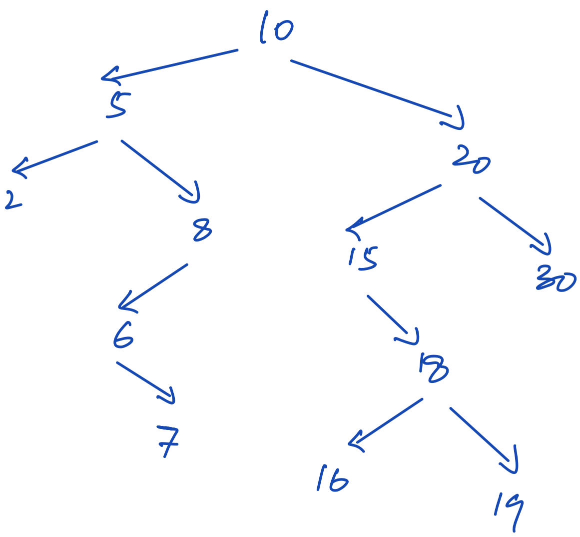

```
TreeNode build(A, l, r) {      // l → 0, r → N-1
    if (l > r) return null
    mid = (l+r)/2
    root = new TreeNode(A[mid])
    root.left = build(A, l, mid-1)
    root.right = build(A, mid+1, r)
    return root
}
```

$TC = O(N)$

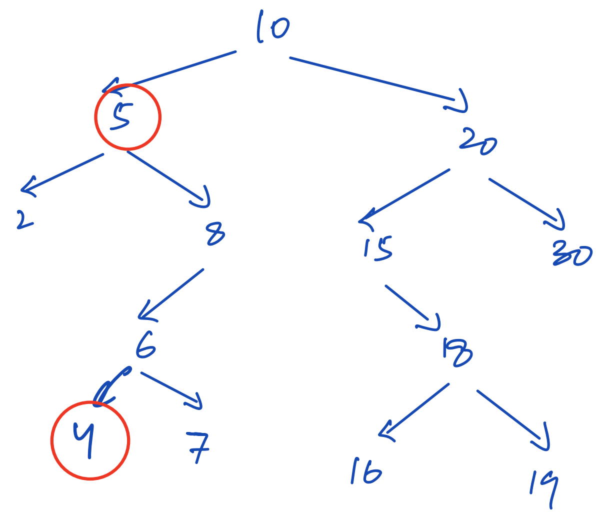
$SC = O(\log_2 N)$

$\boxed{H = \log_2 N}$

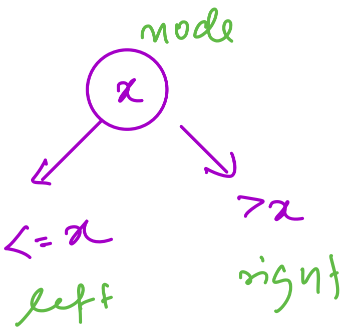
Ques → check if the given binary tree is a BST ?



ans = true



ans = false



left node right (Inorder)

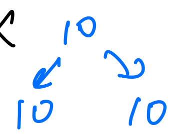
⇒ sorted order

BST inorder traversal ⇒ sorted ✓

Inorder traversal sorted ⇒ BST

→ $\boxed{\text{Distinct elements}}$ ✓

→ Duplicate elements on both sides ✗

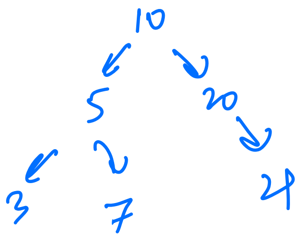


TC = O(N)

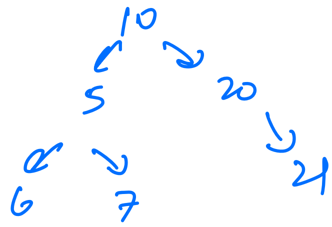
SC = O(H) → next class → Inorder traversal in SC = O(1)

Sol. → ∀ nodes → node.data >= left child

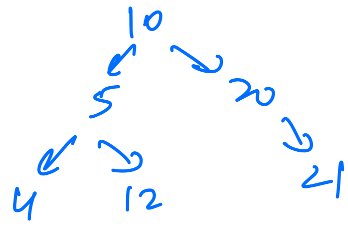
node.data < right child



ans = true



ans = false

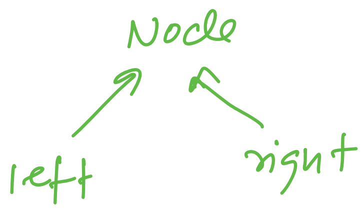


ans = ~~true~~ → false

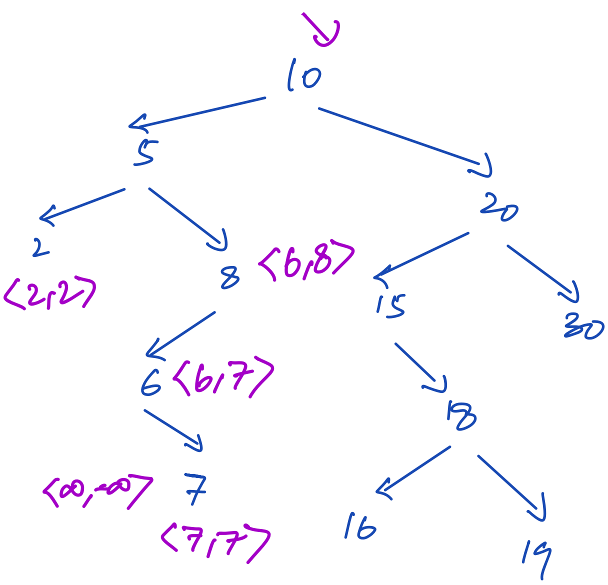
∀ nodes x

$x.data >= \boxed{\text{left subtree data}}$ → max in left subtree

$x.data < \boxed{\text{right subtree data}}$ → min in right subtree



postorder



```
        isBST = true

{min, max}
pair travel(root) {
    if (root == null)   return {INT_MAX, INT_MIN}

    L = travel(root.left)
    R = travel(root.right)

    if (root.data < L.max || root.data >= R.min) {
            isBST = false
    }
    minRoot = min(L.min, R.min, root.data)
    maxRoot = max(L.max, R.max, root.data)

    return {minRoot, maxRoot}
}
```

TC = O(N)

SC = O(H)