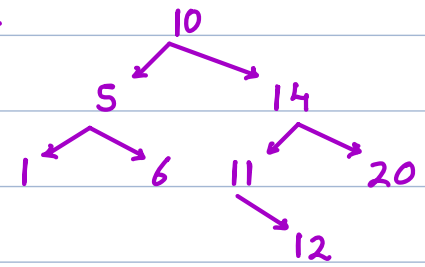
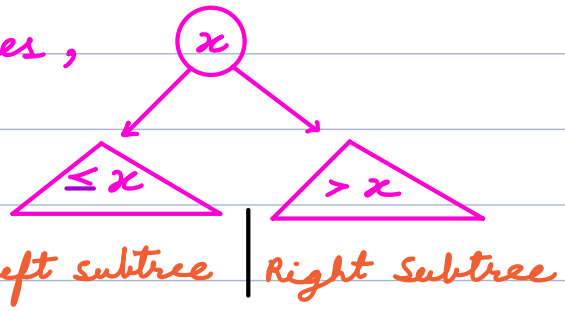
Binary Search Tree

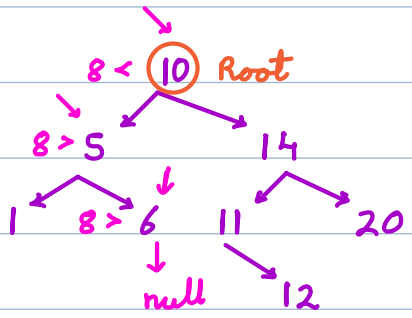
Eg →



∀nodes,



Searching



search (11) 10 → 14 → 11 ✓

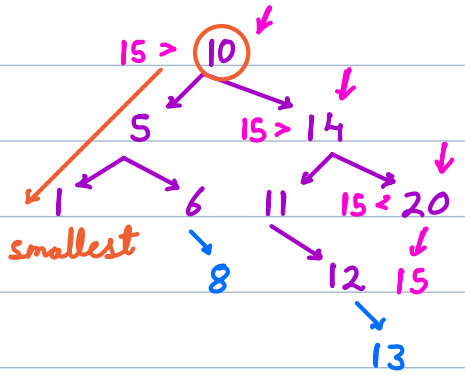
search (8) 10 → 5 → 6 → null ✗

```
while (root != null) {
    if (root.data == X) return true
    if ( x < root.data) root = root.left
    else                root = root.right
}
return false
```

TC = O(H)

SC = O(1)

Insertion



insert (8) 10 → 5 → 6 → 8

insert (13) 10 → 14 → 11 → 12 → 13

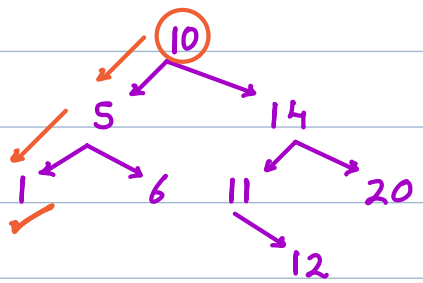
```
// insert X
```

```
while (root != null) {
    if ( x <= root.data) {
        if (root.left != null) root = root.left
        else { root.left = new node(x)
               break }
    } else {
        if (root.right != null) root = root.right
        else { root.right = new node(x)
               break
             }
    }
}
```

TC = O(H)

SC = O(1)

Q → Find smallest element in BST.




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

TC = O(H)

SC = O(1)

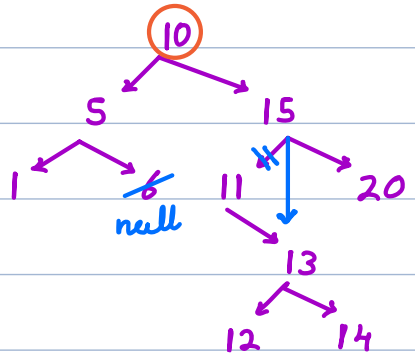
Q → Find largest element in BST.

```
temp = root
while ( temp.right != null)
    temp = temp.right
}
return temp.data
```

TC = O(H)

SC = O(1)

Deletion in BST

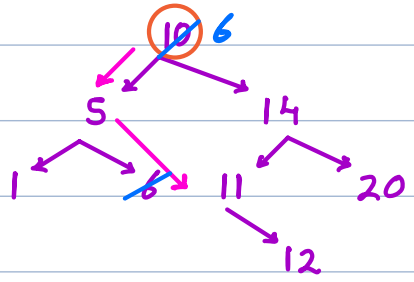


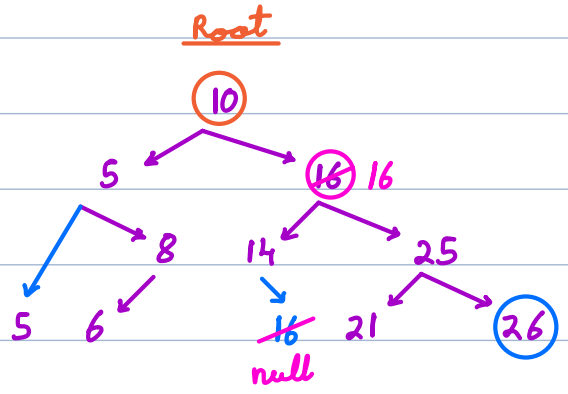
// search for node to delete

1) Node with 0 children (leaf)
   → update it to null
   delete (6)

2) Node with 1 child → delete (11)
   → replace the node to delete with its only child.

3) Node with 2 children → delete (10)
   → find max in left subtree (let say x)
   → delete x from its position
   → replace current node with x



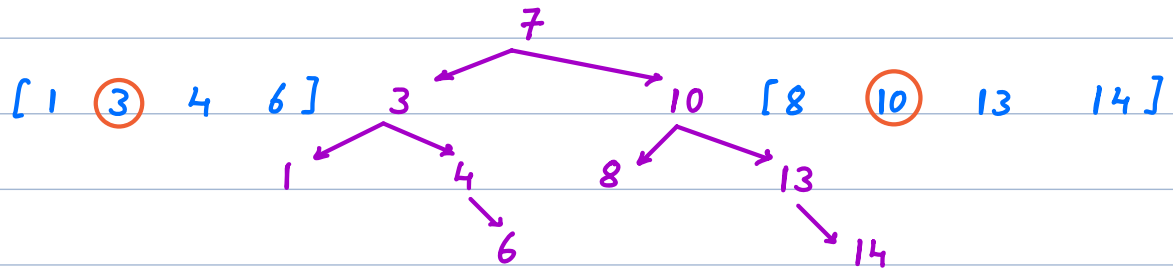


insert (16) ✓
search largest → 26
delete (1) ✓
delete (2) ✓
delete (16) ✓

TC = O(H)

Q → construct a height balanced BST from sorted array of unique elements.

	0	1	2	3	4	5	6	7	8
A = [	1	3	4	6	7	8	10	13	14]




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

TC = O(N) SC = O(log(N))

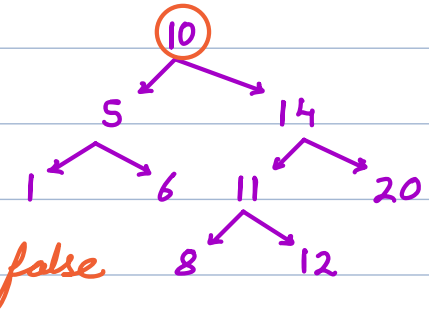
Q → check if the given binary tree is a BST.

MS

Sol 1 → L N R

<= >

Inorder traversal of BST is sorted. ✓

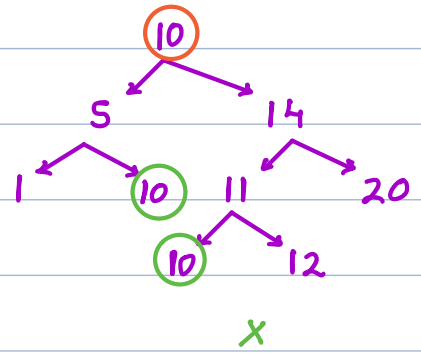


Ans = false

For given binary tree find inorder traversal & check if sorted → yes, BST
→ no, not a BST

↳ unique elements ✓

1 5 10 10 10 11 12 14 20 ✓



X

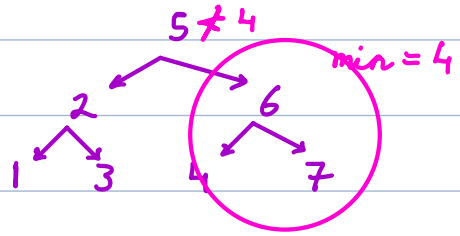
Sol 2 → ∀ nodes, x

left max

all data in left subtree <= x &&

all data in right subtree > x ⇒ BST ✓

right min



Max isBST = true

Min

```
Pair maxMin(root) {
    if (root == null) return {Int_Min, Int_Max}
    L = maxMin(root.left)
    R = maxMin(root.right)

    if (L.Max > root.data || R.Min < root.data)
        isBST = false

    return { max(root.data, max(L.Max, R.Max)),
             min(root.data, min(L.Min, R.Min))}
}
```

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