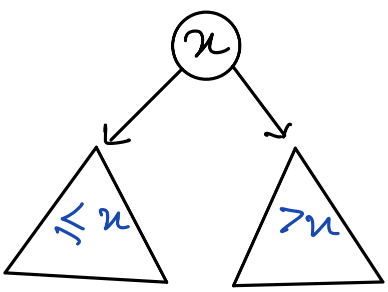
Welcome ☺

Agenda : BST
Operaⁿ
2 quesⁿs

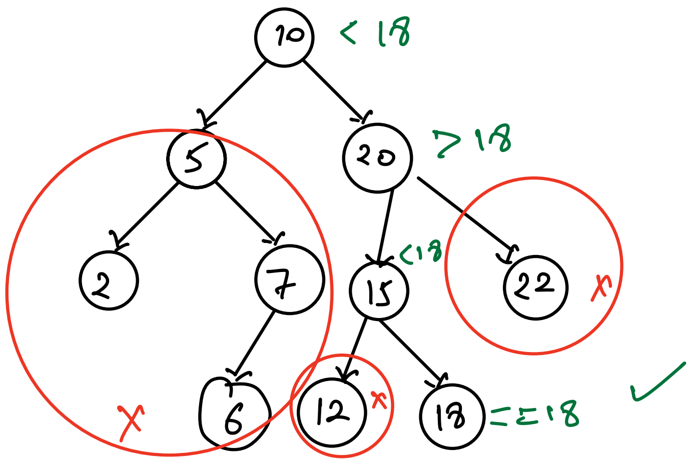
Binary Search Tree

↳ Searching data in organised dataset using divide & conquer



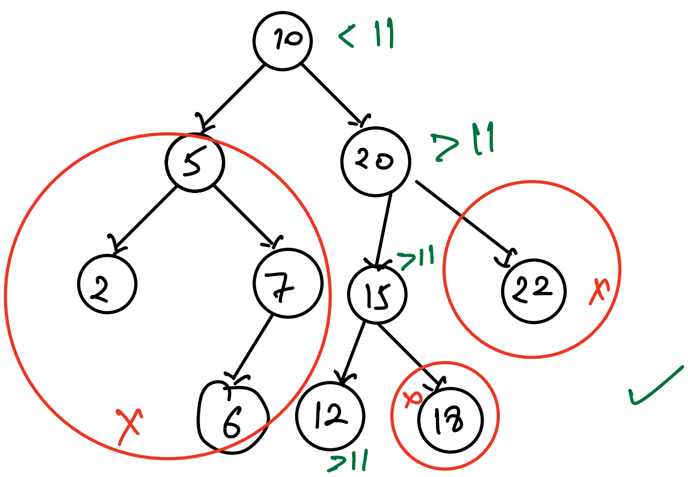
∀ nodes → all data on the left subtree ≤ x
→ all data on the right subtree > x

equality on left side



1) Searching

Find (18) ⇒ 10 → 20 → 15 → 18



Find (11) ⇒ 10 → 20 → 15 → 12 → NULL.

T.C ⇒ O(H)

S.C ⇒ O(1)

Q Find smallest element → left most node.

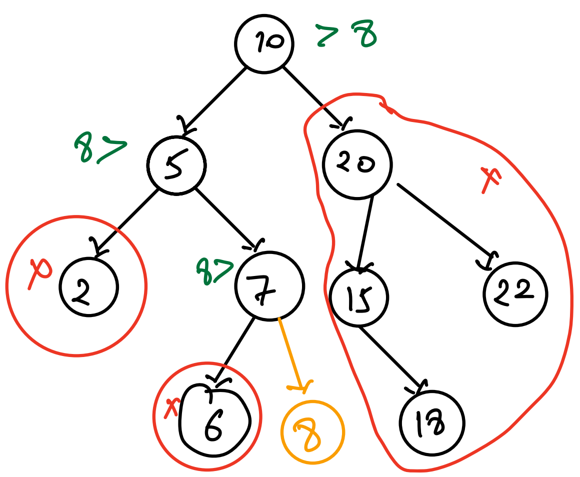
```
temp = root
while ( temp.left != NULL)
{
    temp = temp.left)
}
return temp.data.
```

T.C → O(H)

S.C → O(1)

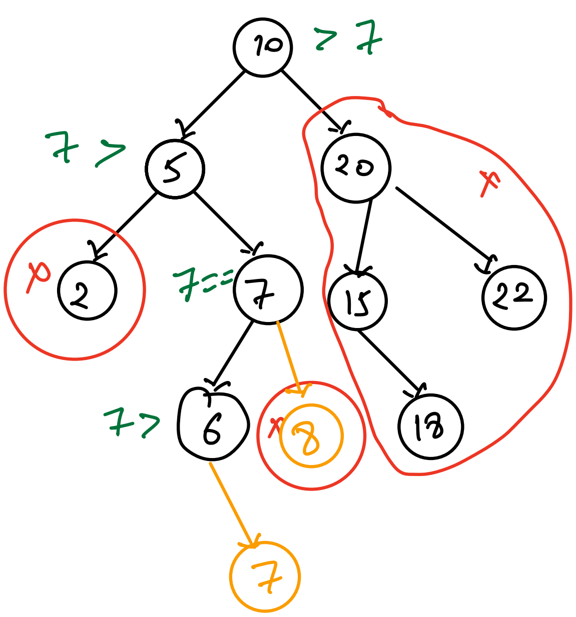
Q Find greatest element → right most node.

Insertion in BST



insert (8) ⇒ searching for 8 & insert it as leaf node.

⇒ Always try to insert elements as leaf nodes to avoid complexity. But it is not compulsion.



insert (7)

T.C → O(H)

S.C → O(1)

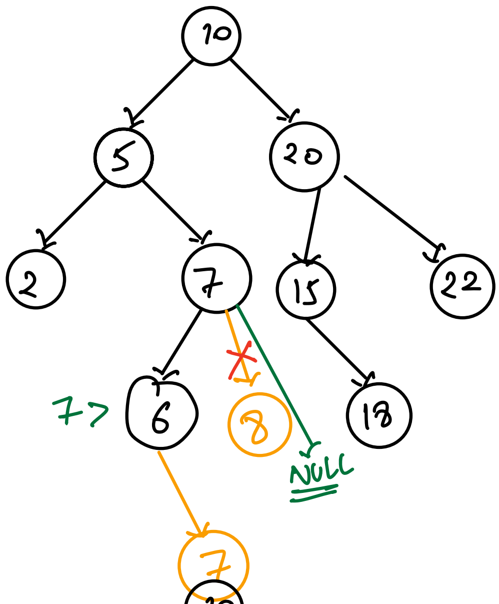
Code

```
nn = new Node ( X )
if( root == NULL) return nn
temp = root
while ( temp != NULL)
{
    if( temp.data < X )
    {
        if( temp.right == NULL) {
            temp.right = nn
            return root
        }
        temp = temp.right
    }
    else
    {
        if( temp.left == NULL) {
            temp.left = nn
            return root
        }
        temp = temp.left .
    }
}
```

T.C → O(H)

S.C → O(1)

Deletion in B.S.T



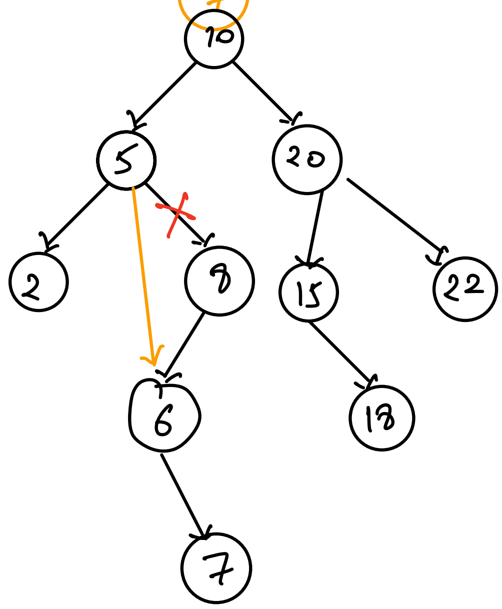
1) Search for the node to delete

T.C ⇒ $O(H)$

2) a) If node to be deleted is a leaf node.

delete(8)

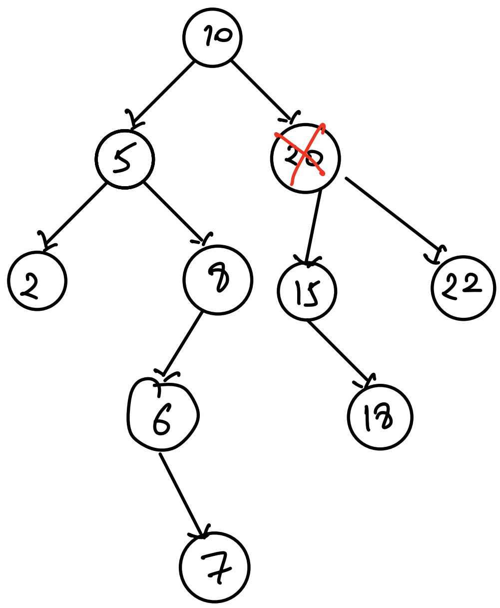
⇒ parent points to NULL



b) If node to be deleted has 1 child.

delete(8)

→ parent points to single child.



c) If node to be deleted has 2 child

delete(20)

a) Find greatest ele. in left subtree of the node to be deleted. ⇒ x

b) Remove x from its pos^n.

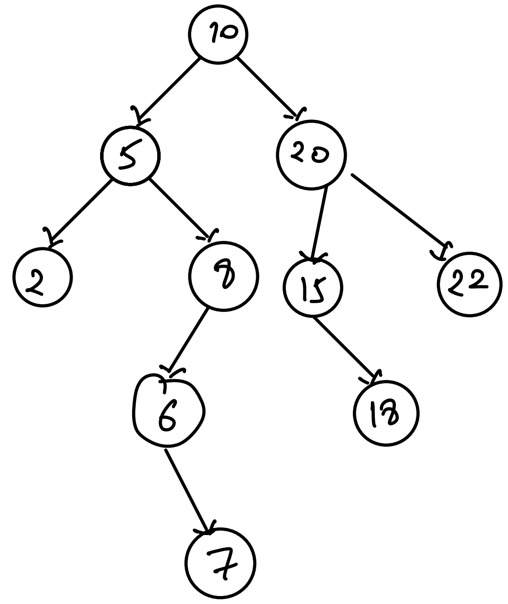
c) Replace the node to be deleted with x.

LNR

Inorder traversal of B.S.T

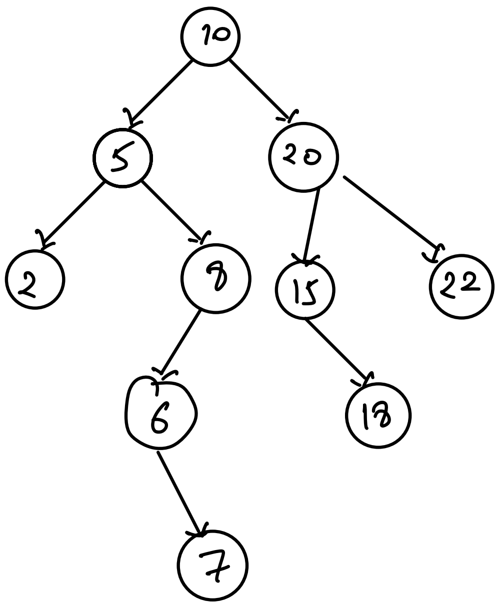
$\Downarrow$

Sorted data.



2 5 6 7 8 10 15 18 20 22

Q Check if the given binary tree is a B.S.T



→ ∀ nodes,

$\boxed{\text{if } n.\text{data} \geq \text{all nodes in left subtree}}$ ⇒ left_max ≤ n

&

$\boxed{n.\text{data} < \text{all nodes in right subtree}}$ ⇒ right_min > n

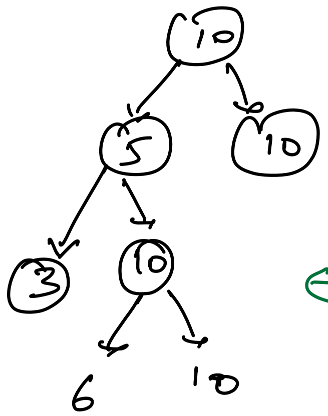
Code

```
// { max, min }
isBST = true.
pair max_min( root)
{
    if( root == NULL) return { INT_MIN , INT_MAX }
    L = max_min( root.left)        Left
    R = max_min( root.right)       Right.
    if ( L.max > root.data || R.min <= root.data ) {
```

```
        isBST = false
    }
    return { max(root.data, L.max, R.max),
             min(root.data, L.min, R.min) }
}
```

$T.C = O(N)$ $S.C = O(H)$

Q Can inorder traversal give sorted data for non BST.



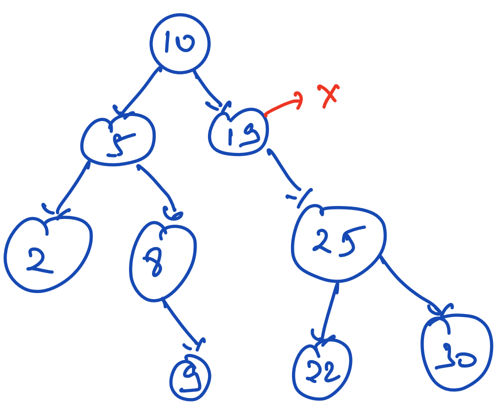
3 5 6 10 10 10 10

Sorted ✓

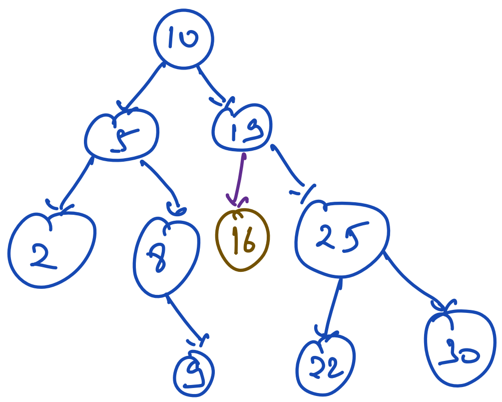
X BST

Balanced B.S.T

$\forall$ nodes abs(height(left) − height(right)) ≤ 1



X not balanced BST

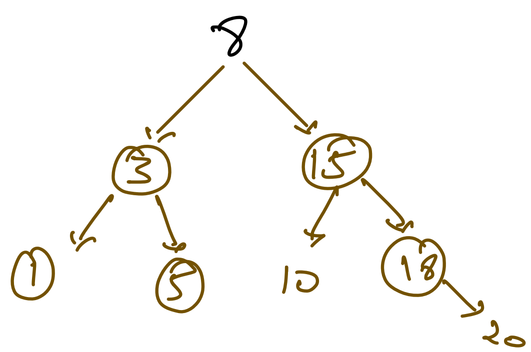


✓ Balanced BST.

Q Construct balanced BST from sorted array of unique elements.

mid.

eg: 1 3 5 $\boxed{8}$ 10 15 18 20



Code

```
Node build ( A[], L, R ) {
    if ( L > R ) return NULL
    mid = L+R/2
    root = newNode ( A[mid] )
    root.left = build ( A[], L, mid-1 )
    root.right = build ( A[], mid+1, R )
    return root
}
```

T.C → O(N)

S.C → log(N)

↓

recursion space