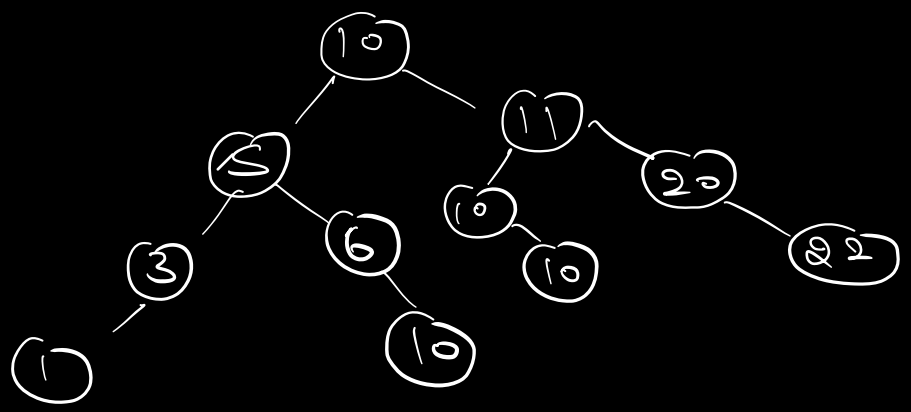
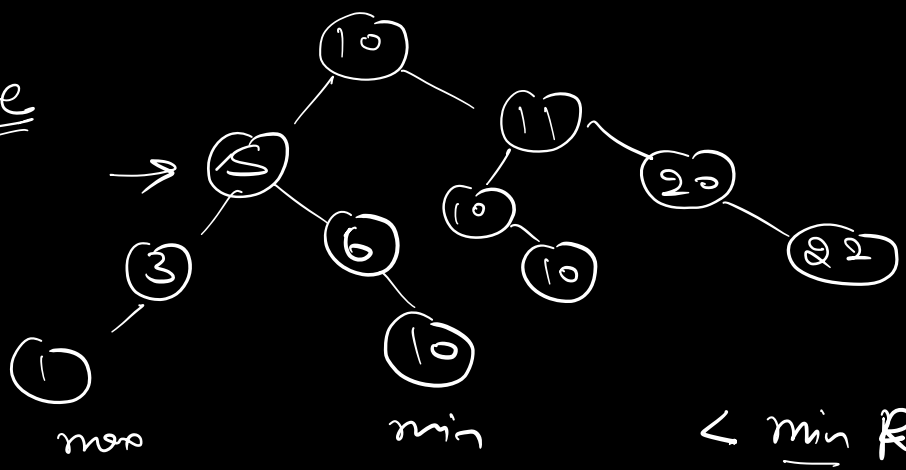
Q ⇒ Given a binary tree. Check if it is a BST.

1) Inorder Traversal ⇒ Sorted.



✗ LNR ⇒ 1, 3, 5, 6, 10, 10, 10, 11, 20, 22

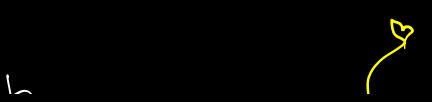
↓ Pre
NLR



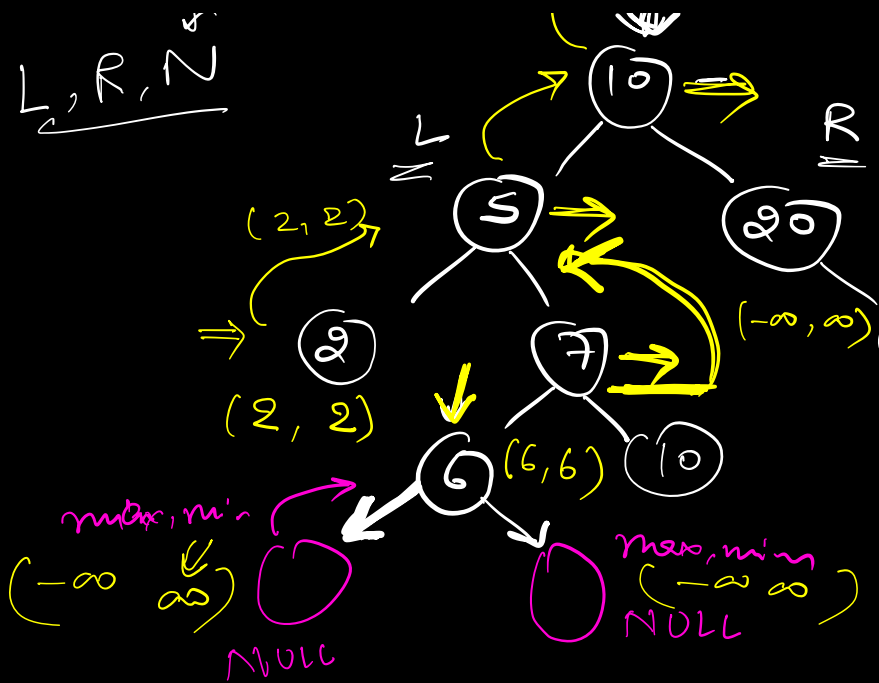
L to Rot

< min RST

≥ max LST



L, R, N



bool isBST
= true
false

<max, min>

min = min (left.min , root.value);

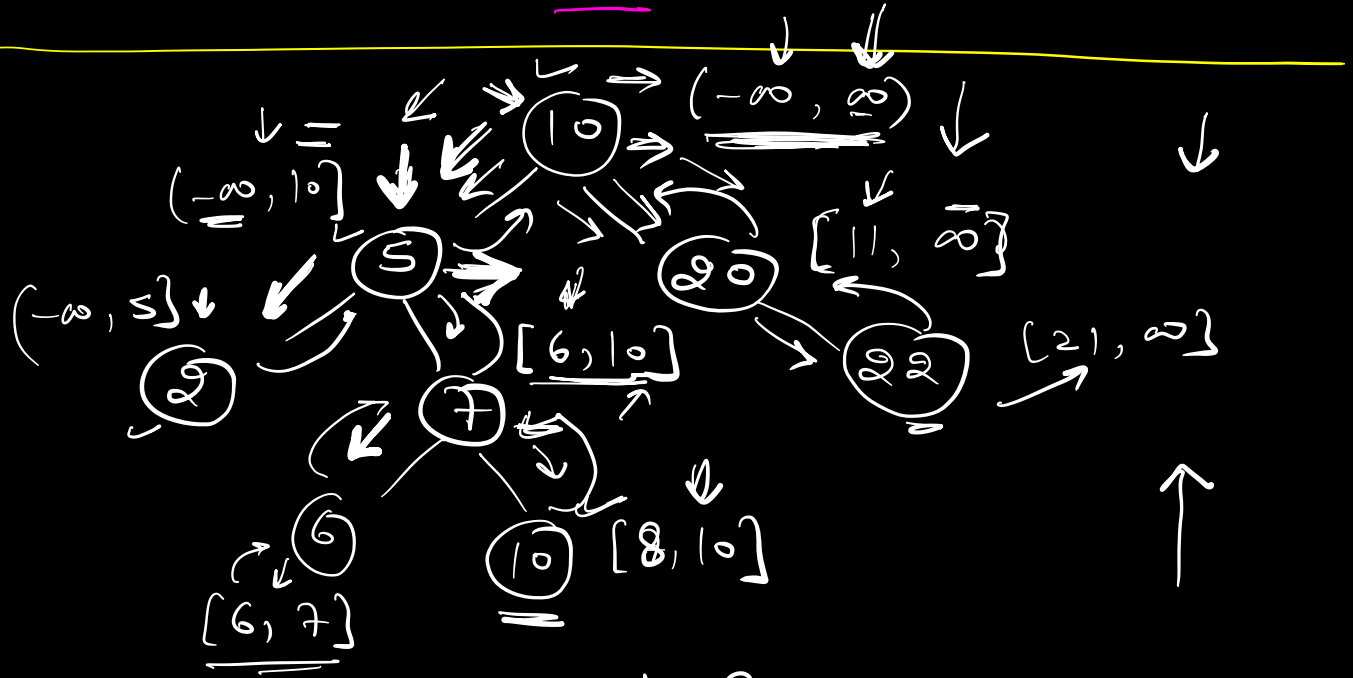
max = max (root.value, right.max);

Code

```
bool isBST = true;
pair <max, min> isTreeBST (root) {
    if (root == NULL) {
        return { INT_MIN, INT_MAX };
    }
    if (isBST) {
        pair l = isTreeBST (root.left);
        pair r = isTreeBST (root.right);
```

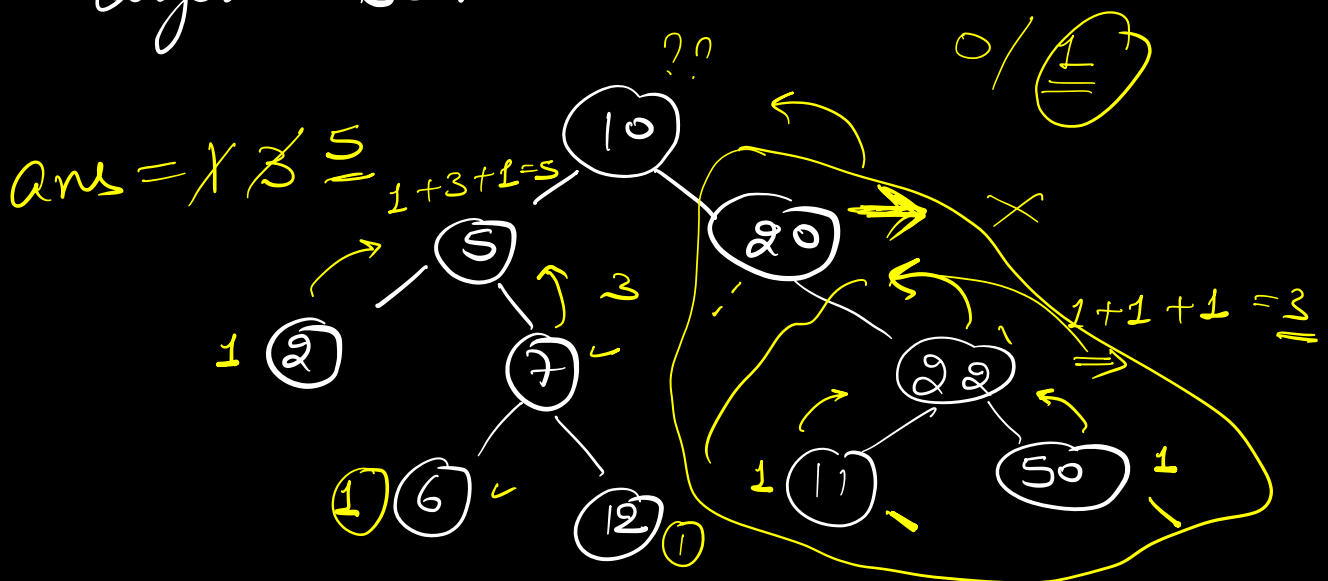
```
if ( root.value < left.max ||
     root.value >= right.min ) {
        isBST = false;
}
return { max(root.value, r.max),
         min(root.value, l.min);
}
```

T.C. = O(N)



NLR => pre

Q Given a Binary tree.
Return the root node of the largest BST



nodes = 1 + # nodes in LST
+ # nodes in RST

Traversal ⇒ Post order

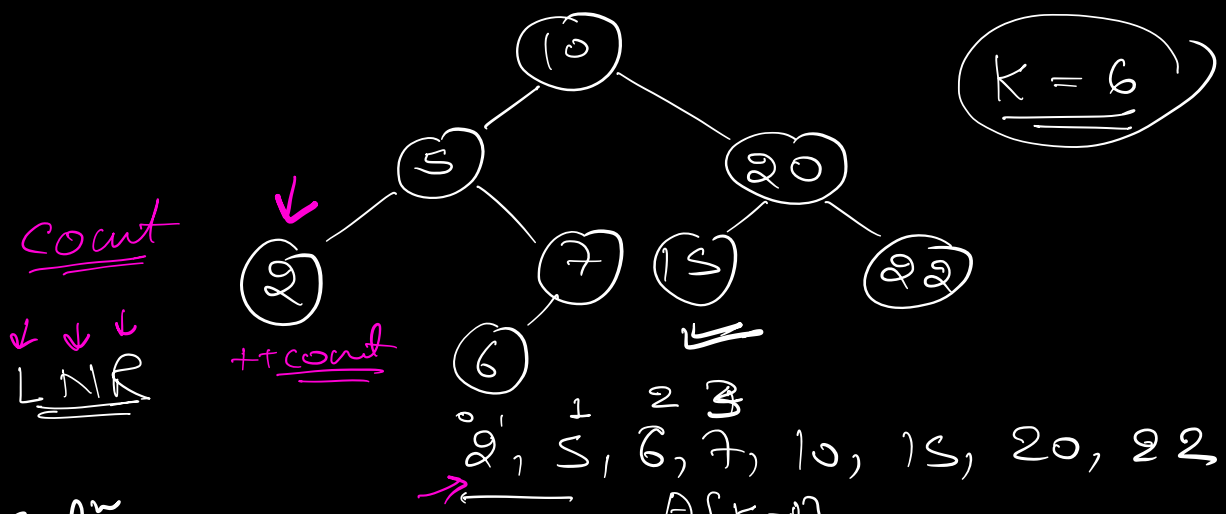
Information to propagate upwards ⇒ (max, min, # nodes)

C++
struct

Java
class ≤ max, min, count.

T.C. = O(N)

Q Given a BST & K
Find the K^{th} smallest element.



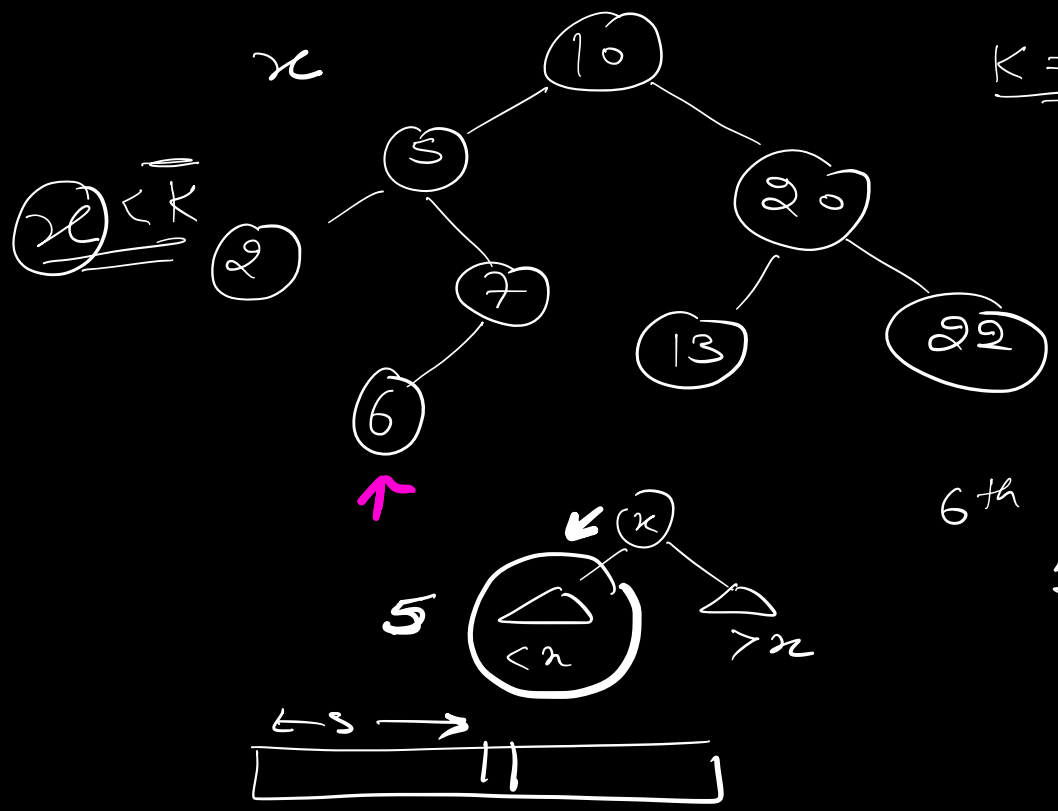
Soln 1) Convert it to a sorted array

⇒ Inorder Traversal

T.C. = O(N)
S.C. = O(N) ⇒ Array

inorder (root)

root.left;
node
root.right
count++;
if (count == K)



$x \geq K$ k^{th} smallest in left subtree

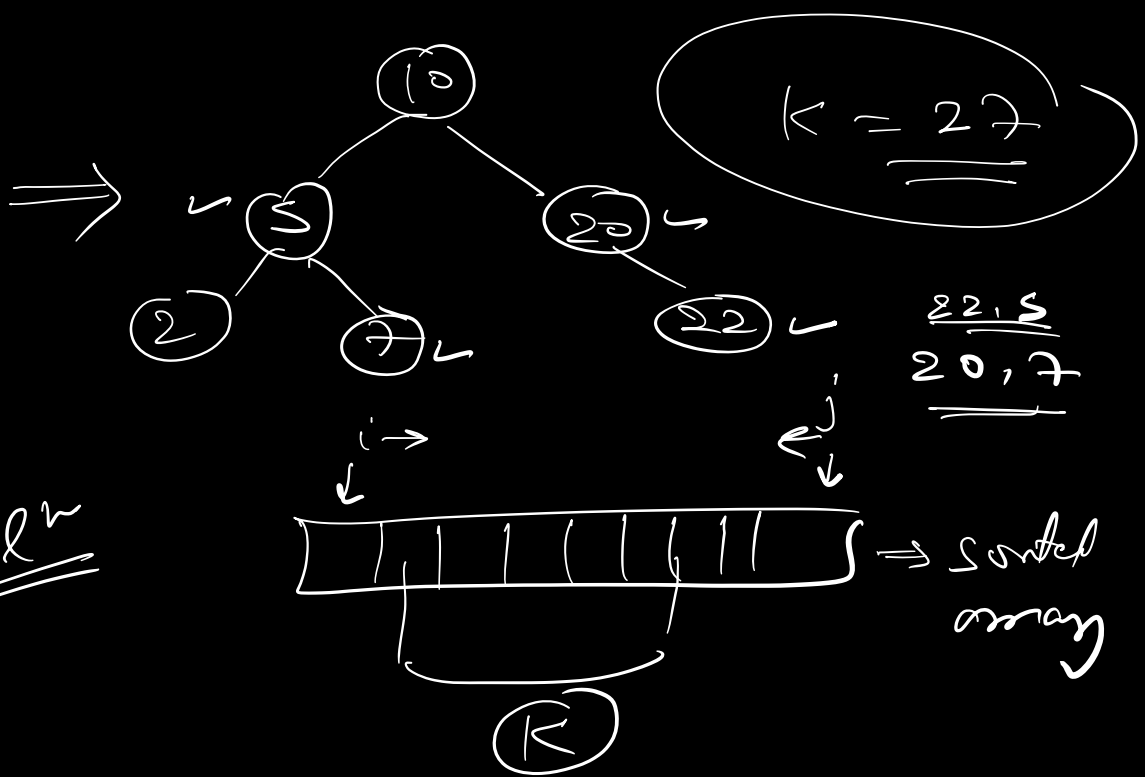
$x = K-1$ ⇒ root

$x < K$ ⇒ right subtree

```
node {
    int value;
    int size;
}
```

$T = O(N + H)$

Q Given a BST.
Find 2 nodes with a given sum K.



Sol^{n}

(1) (i) Inorder to create sorted array
(ii) Use 2 pointer approach

$$T.C. = O(N+N) = O(N)$$

$$S.C. = O(N) \times$$

$$O(1)$$

(constraint

$\Rightarrow$ Solve in constant space.

2) find a pair

(i) $N \Rightarrow {}^{N}C_2 = O(N^2)$

(ii) (i, j) $i + j = K$

$\forall$ element i, find $(K-i)$ in the tree

$$j = K - i$$

$$T.C. = O(N \times H)$$

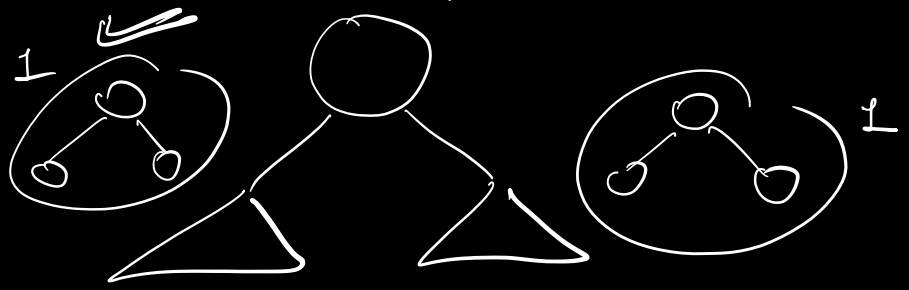
H.W. Can this approach ever fail ?!

Q Given a sorted array
Construct a balanced BST

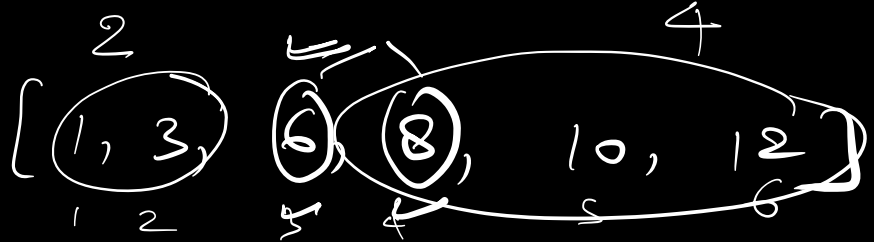
A ⇒ [1, 3, 5, 8, 10, 15, 20]

mid

Soln



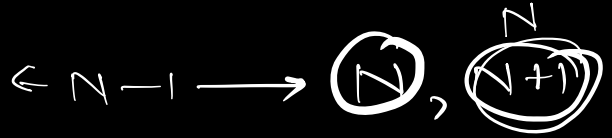
Equal



does not matter

[ ]

2N

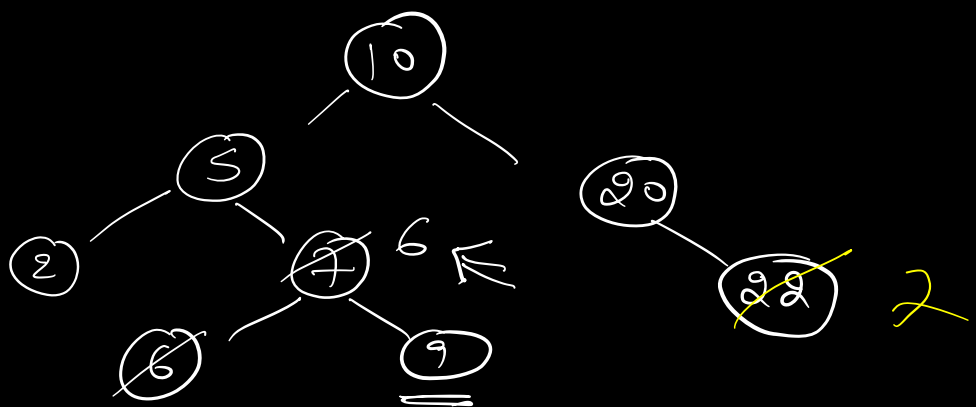


(N+1) − (N = 1)

N − (N−1) = 1

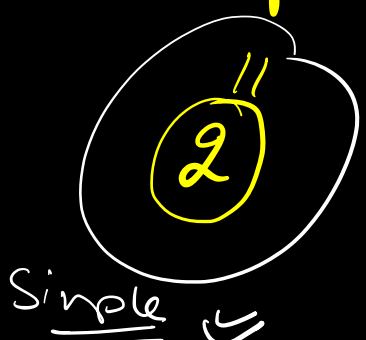
= (1)

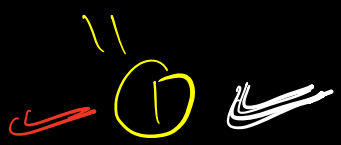
Q Given a BST (unique nodes)
2 nodes in the BST are swapped
find the 2 nodes



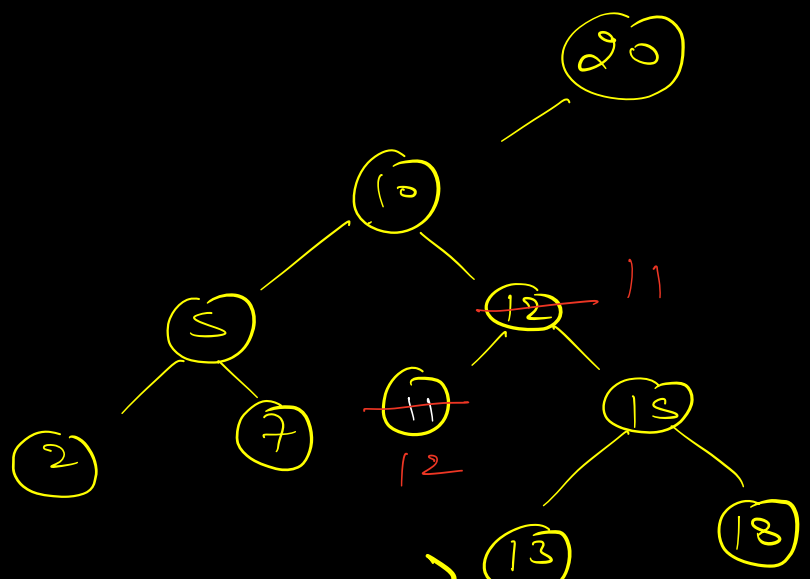
Sol^{n}

① from root to leaf check if BST node is broken





↳ Inorder successor



Inorder Successor

2, 5, 7, 10, 12, 11, 13, 14, 15, 18

(2)

$>$

2, 5, 7, 10, 11, 13, 12, 14, 15, 18

greater

less

(Post order)

T.C. = O(N)

Alternate approach

1) Inorder Traversal

2 faulty elements ⇒ as it is

1 faulty element ⇒ faulty element, prev element

T.C. = O(N)

S.C. = O(N) = O(1)

H.W.

O(H) ⇒ recursive stack

X