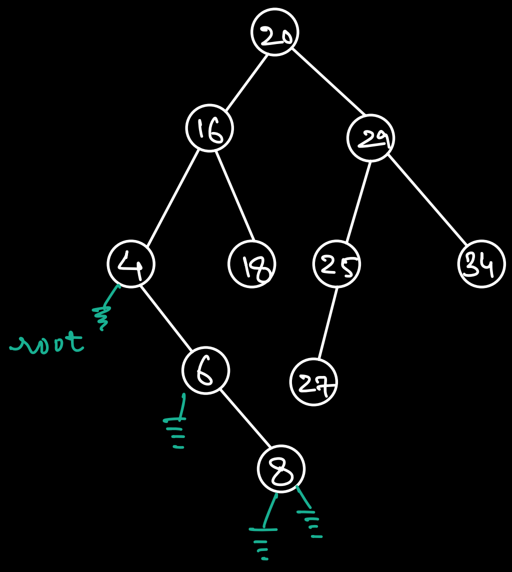
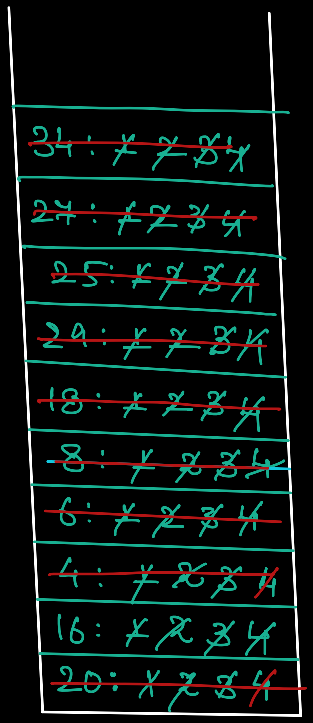
Inorder :-

Left Root Right

```
void InOrder ( root) {
        1) if( root == Null)
                return ;
        2) InOrder ( root.left);
        3) print (root. data)
        4) InOrder ( root. right);
}
```



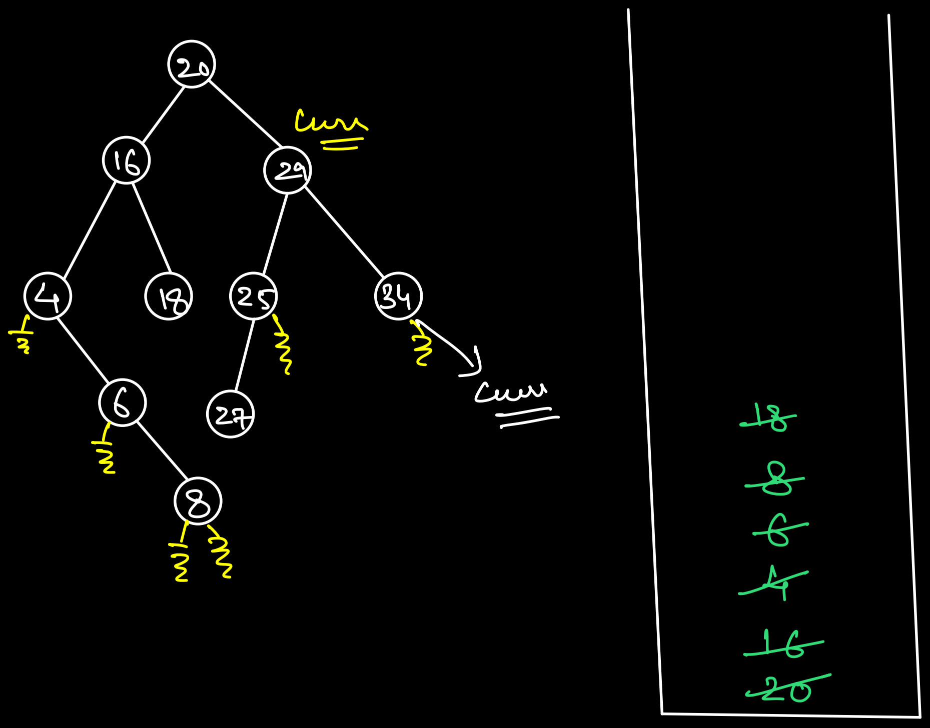


4, 6, 8, 16, 18, 20, 27, 25, 29, 34

1. Till we get a NULL on the left side, keep on inserting in the Stack.

2. If root == NULL, get the top element from Stack, print it & move towards right.

Q. Inorder traversal in iterative way.



4 6 8 16 18 20 27 25 29 34

```
void    inorderIterative ( root ) {
        curr = root;
        Stack < Node >  st ;
        while ( curr != Null  || st.size() > 0 ) {
              if (curr != Null ) {
                     st.push(curr)
                     curr = curr.left;
              }
              else {
                   Node temp = st.top()
                   st.pop();
                   print(temp.data);
                   curr = temp.right;
              }
        }
}
```

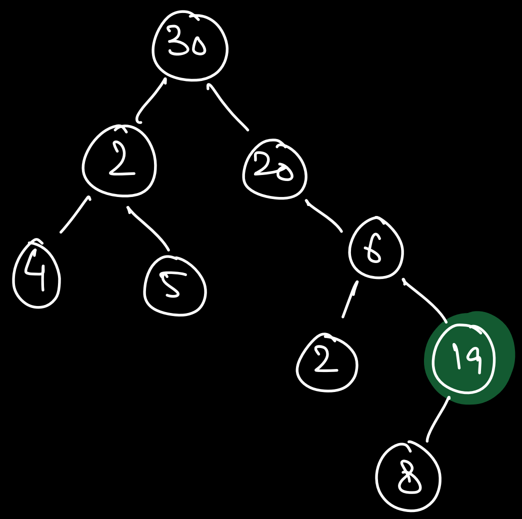
TC : O(N)

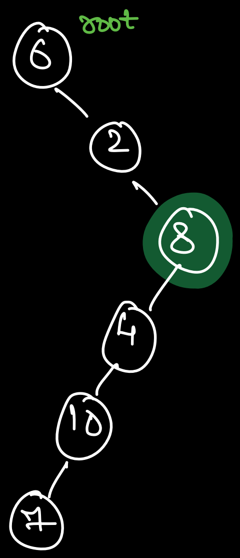
SC : O(N)

HW

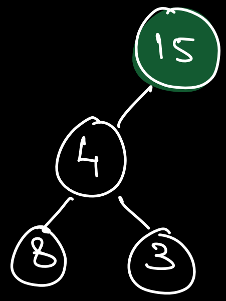
Pre Order
PostOrder } ✓

Q. Given a tree, find the last inorder node that we print.





⇒ Keep going on right side until we get a NULL.



⇒ Right most node.

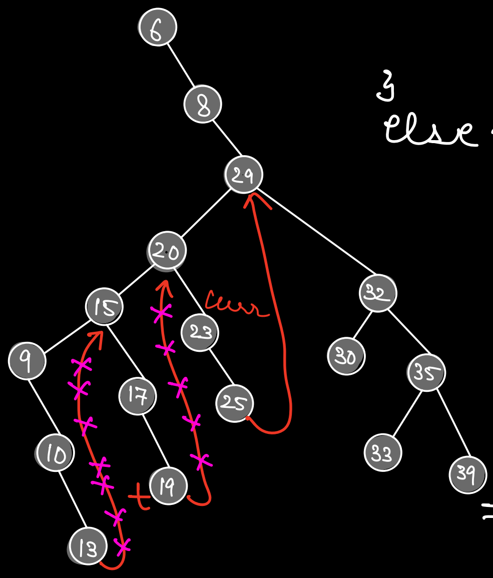
Morris Inorder Traversal

↓

Expected SC : O(1)

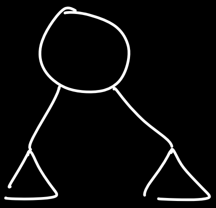
6 8

```
inOrder (Node root) {
    Node curr = root;
    while (curr != NULL)
        if (curr.left == NULL) {
            print (curr.data)
            curr = curr.right;
        }
        else { // curr.left != NULL
            temp = curr.left;
            while (temp.right != NULL &&
                    temp.right != curr) {
                temp = temp.right;
            }
            if (temp.right == NULL) {
                // Visiting curr Node 1st time
                temp.right = curr;
                curr = curr.left;
            }
            else { //temp.right = curr, i.e visiting
                   // curr node 2nd time.
                temp.right = NULL
                print (curr.data);
                curr = curr.right;
            }
        }
}
```



$$\boxed{\begin{array}{l} TC : O(N) \\ SC : O(1) \end{array}}$$

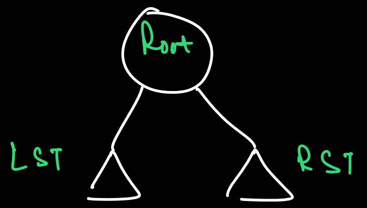
Q: Given a Binary Tree, check if it is a BST or not.



∀ nodes :

LST < Root.val < RST

(I)



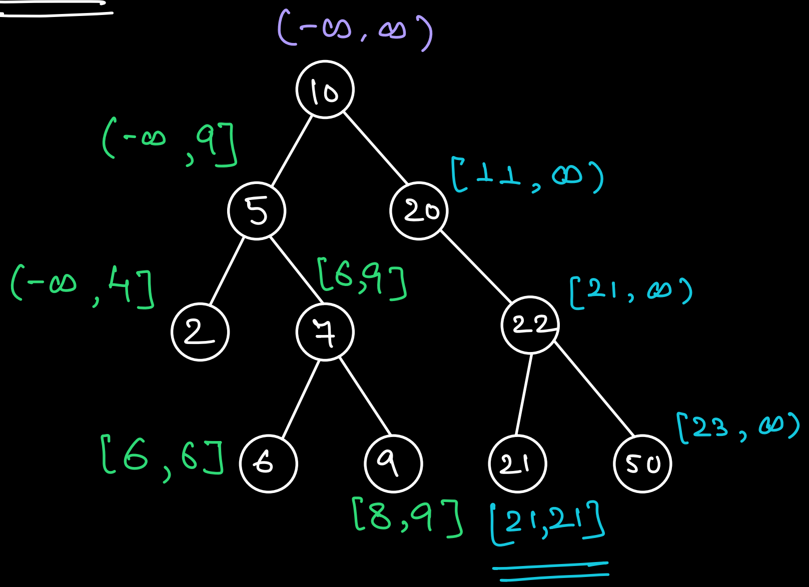
⇒ Inorder : $\underbrace{\text{LST} \quad \text{Root} \quad \text{RST}}_{\text{Sorted.}}$

- Store the inorder traversal of BT
- If it is sorted in ascending order return true, else return false.

TC : O(N)

SC : O(N)

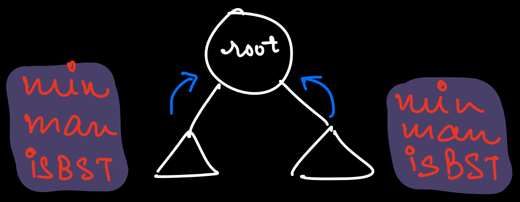
(II) PreOrder



```
                          -∞   +∞
bool  isBST ( root ,  l ,  r ) {
    if (root == NULL)  return  true;
    if (root.val >= l && root.val <= r){
        bool lst = isBST(root.left, l, root.val-1);
        bool rst = isBST(root.right, root.val+1, r);
        return lst && rst;
    }
    return false;
}
```

Preorder

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

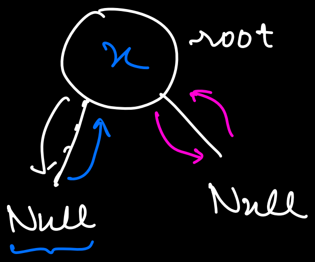


```
Class TreeInfo {
    int min;
    int max;
    bool isBST;
    ______
    ______
}

TreeInfo isBST(root) {
  if (root == NULL) {
      return new TreeInfo(+∞, -∞, true);
  }
  TreeInfo l = isBST(root.left);
  TreeInfo r = isBST(root.right);
  if (l.isBST && r.isBST &&
          root.val > l.max && root.val < r.min){

      return new TreeInfo(min(root.val,
                              l.min
                              r.min), max(root.val
                                          l.max
                                          r.max), true)
  }
  return new TreeInfo(min(root.val, max(root.val
                          l.min         l.max
                          r.min),       r.max), false);
}
```

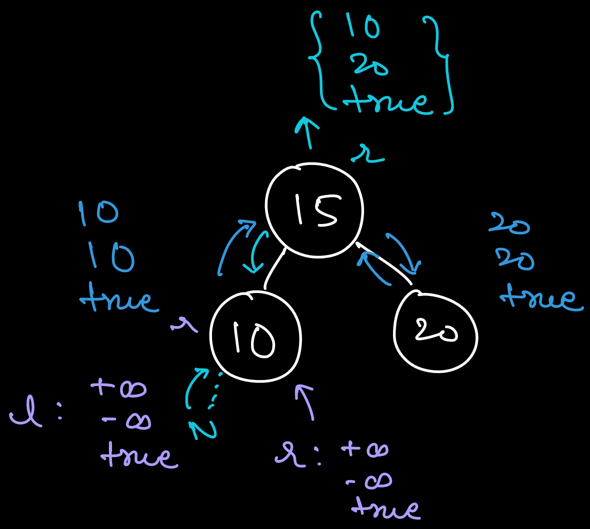
MIN → +∞
MAX → −∞
isBST → true

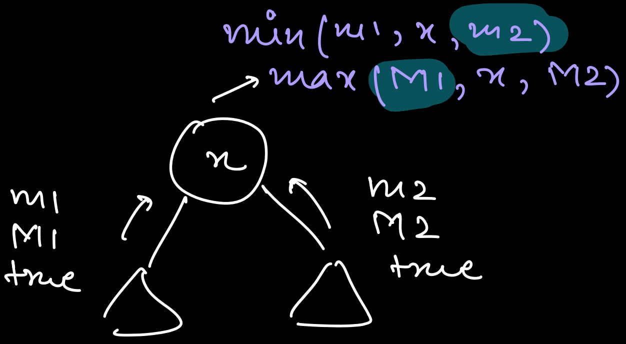


+∞ ← MIN
−∞ ← MAX
true

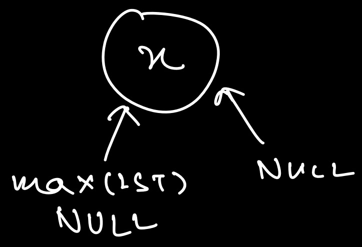
TC : O(N)
SC : O(N)

min
max
isBST





$x > M1$
&&
$x < m2$



$x > \max(LST)$

$\Downarrow$

$-\infty$

$x < \min(RST)$

$\Downarrow$

$\boxed{+\infty}$

$$\left\{ \begin{array}{l} \min = +\infty \\ \max = -\infty \\ isBST = true \end{array} \right\}$$