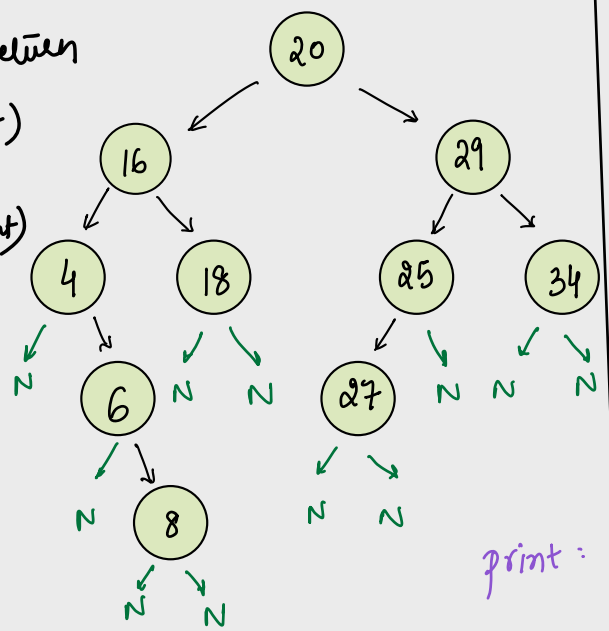
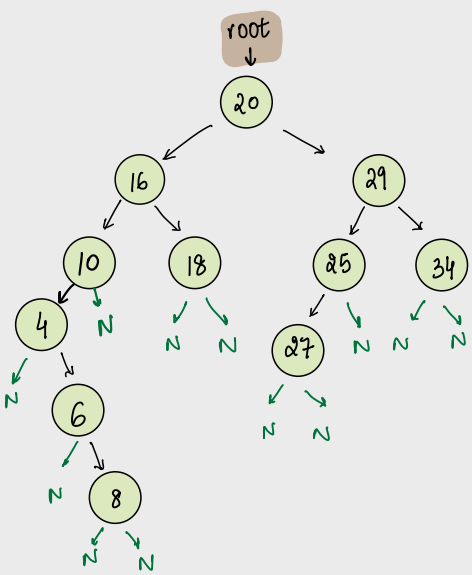
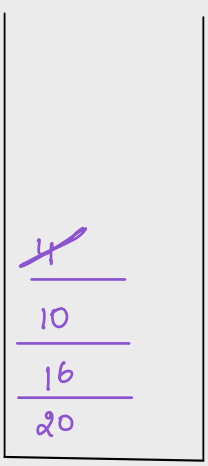
Inorder (LDR)

```
void inorder (Node root){
    1  if(root == NULL) return
    2  inorder (root.left)
    3  print (root.data)
    4  inorder ( root.right)
}
```



print : 4 6 8 16 18 20



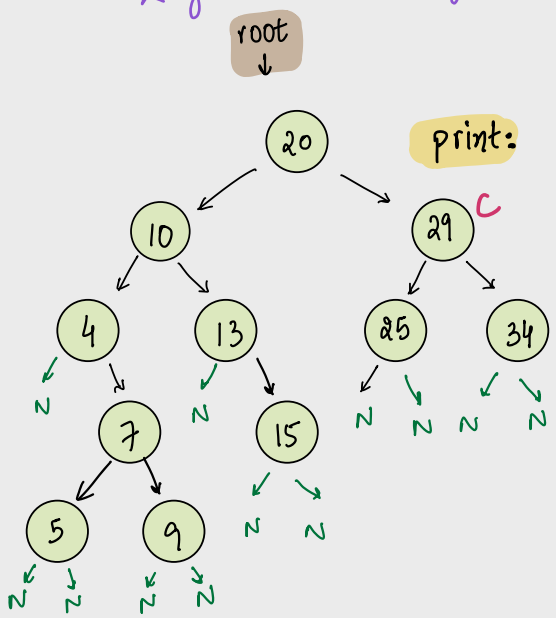


Insert till NULL, come back 4 (remove 4 from stack) print it Go to right

4 ,

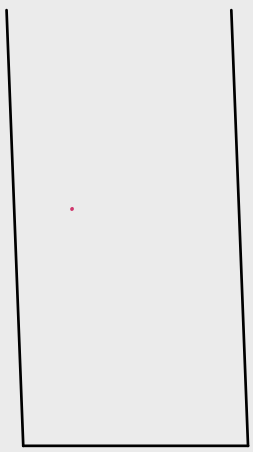
Idea: ① Till you get a NULL on left side, keep inserting into the stack.

② If root == NULL, get the top ele of stack, print it & go to the right.



Iteratively: Need a stack

print: 4 5 7 9 10 13 15 20

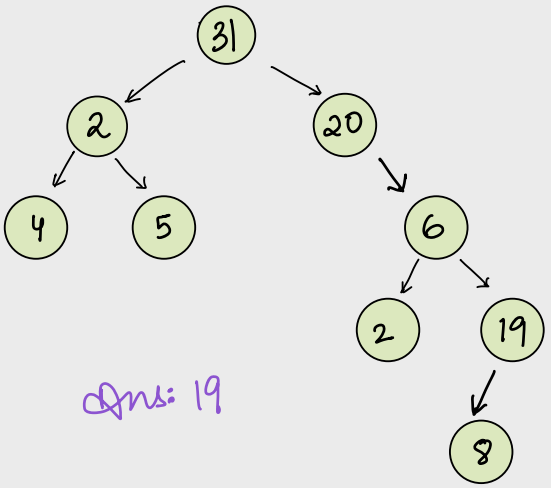


```
void inorderIter(Node root){
    Stack<Node> s;
    Node curr = root
    while (curr != NULL || s.size()>0){
        if (curr != NULL){
            s.push(curr)
            curr = curr.left
        }
        else{
            Node temp = s.top()
            s.pop()
            print(temp.data)
 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

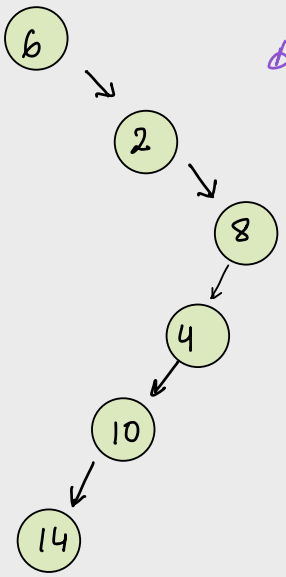
TC: O(N)
SC: O(H)

Que. With inorder traversal on a tree, last node we print

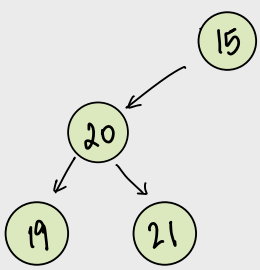
LDR



Ans: 19



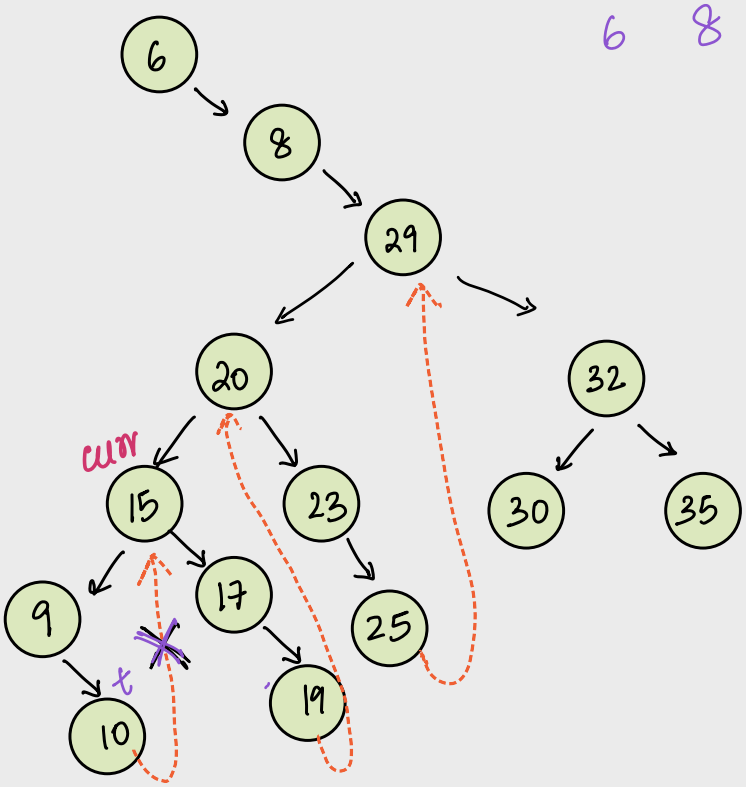
Ans: 8



Ans: 15

Morris Inorder Traversal

LDR



6 8 9 10 15

t.right != NULL

t.right != curr

t.right != curr

if (t.right == curr) {

t.right = NULL

Print (curr)

curr = curr.right.

Pseudocode

```
void morrisInorder(Node root){
    Node curr = root
    while(curr != NULL){
        if(curr.left == NULL){
            print(curr.data)
            curr = curr.right
        }
        else{
            Node temp = curr.left
            while(temp.right != NULL && temp.right != curr){
                temp = temp.right
            }
            if(temp.right == NULL){
                temp.right = curr
                curr = curr.left
            }
            if(temp.right == curr){
                temp.right = NULL
                print(curr.data)
                curr = curr.right
            }
        }
    }
}
```

T && T = T
T && F = F
F && T = F
F && F = F

// connect the right most node of LST to curr

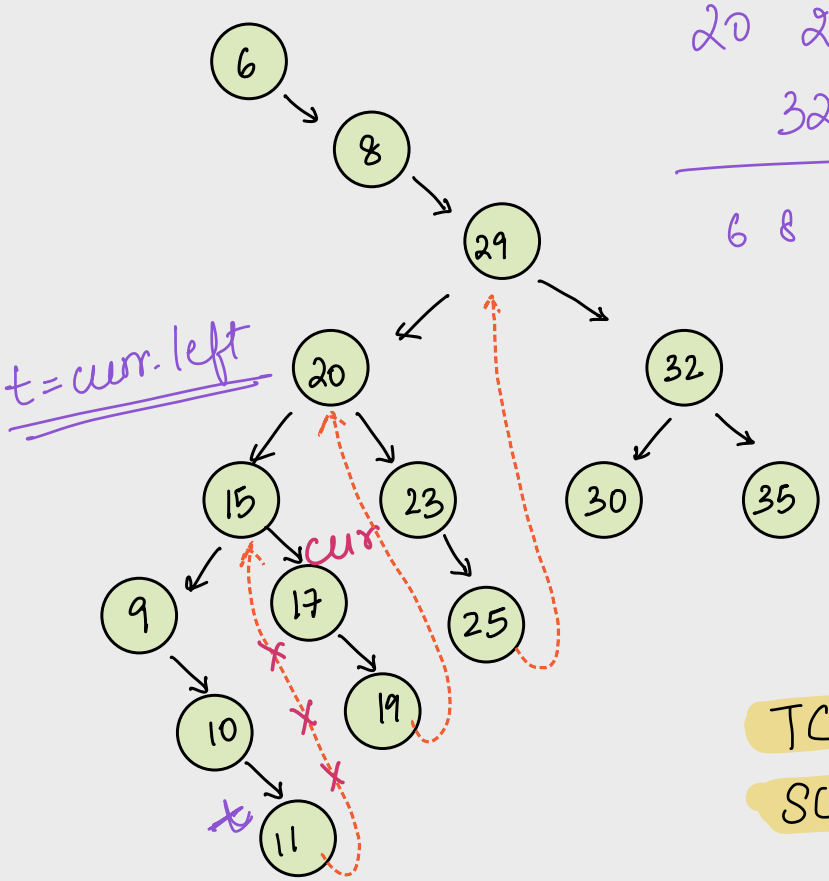
// break the cycle

DRY RUN

6 8 9 10 11 15 17 19
20 23 25 29 30
32 35

---

6 8 9 10 11 15



t = curr.left

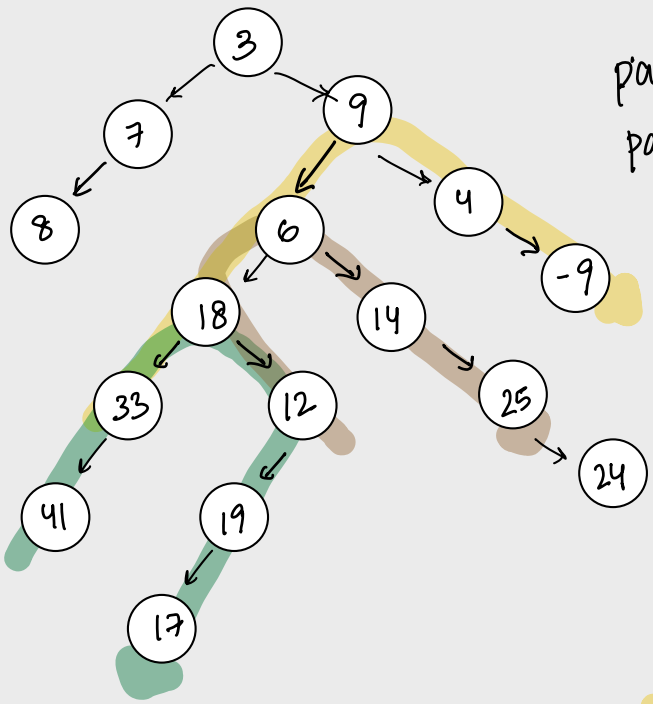
HW.
Try isBST using this approach.

TC: O(N)
SC: O(1)

Break : 10:40

Q. (LCA)

Given 2 values in BT, find shortest path between



41 & 17 : 41 33 18 12 19 17

path(41) : 3 9 6 18 33 41

path(17) : 3 9 6 18 12 19 17

Least / lowest common ancestor

(18)

12 & 25 : 12 18 6 14 25

path(12) : 3 9 6 18 12

path(25) : 3 9 6 14 25

LCA : (6)

33 & -9 : 33 18 6 9 4 -9

path(33) : 3 9 6 18 33

path(-9) : 3 9 > 4 -9

LCA : (9)

a & b

path(a) from root

path(b) from root

stop at the last common node b/w them

traverse arrays to get the ans.

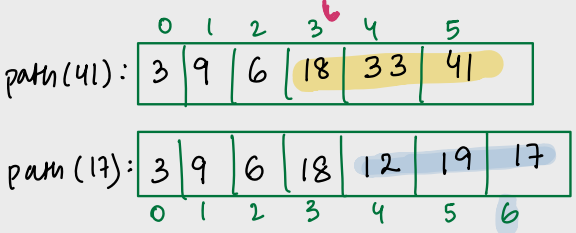
```
List<int> P1, P2

// calc path of a from root in P1
// cal path of b from root in P2        } see in session 1 notes.

for(i=0; i<min(P1.size(), P2.size()); i++){
    if(P1[i] != P2[i]){
        break
    }
}
i = i-1   // index of LCA
for(j = P1.size()-1 ; j>=i ; j--){
    ans.add(P1[j])
}
for(j= i+1 ; j<P2.size() ; j++){
    ans.add(P2[j])
}
```

41 & 17:

41 33 18 12 19 17



P1 [P1.size()-1 ... i]

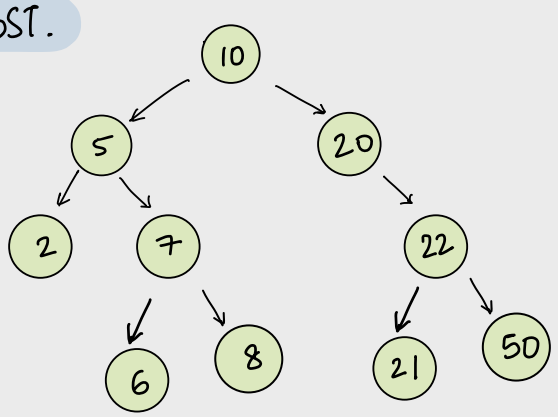
P2 [i+1 ... P2.size()-1]

TC: O(N+N+H)

O(N+H) ≅ O(N)

SC: O(H)

LCA in BST.



LCA (6,2) = 5

LCA (7,22) = 10

LCA (21, 50) = 22

LCA (2, 8) = 5

LCA (7, 8) = 7

LCA (10, 22) = 10

```
int LCAinBST (Node root, n1, n2) {
    if (root == NULL) { return 0 }

    if ((root.val >= n1 && root.val <= n2) ||
        (root.val <= n1 && root.val >= n2))
        return root.val

    if ( root.val > n1 && root.val > n2) {
        return LCAinBST (root.left, n1, n2)
    }
    else if ( root.val < n1 && root.val < n2) {
        return LCAinBST (root.right, n1, n2)
    }
    else
        return root.val
}
```

TC : O(H)

SC : O(H)

Circular Doubly Linked list

```
CNode combine (Node n1, Node n2) {
|
|
|
|
|
|
|
|
}
```

Q. Given BST, convert into circular DLL.