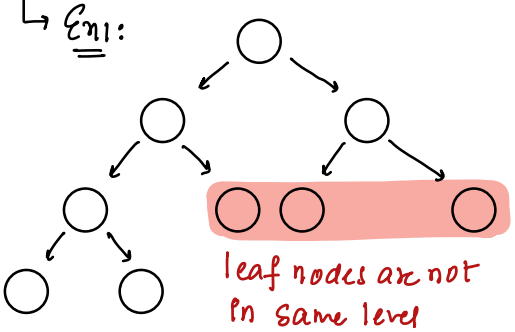
Todays Content: 7:05 AM

- → Perfect Binary Tree
- → fill right in Perfect binary Tree
- → CDLL
- → Merge 2 CDLL
- → BST → CDLL
- → Morris Inorder traversal

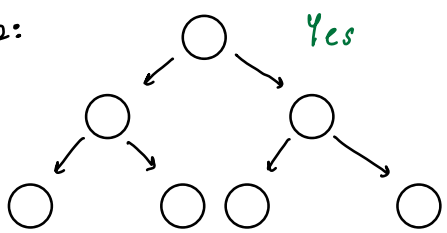
Perfect binary Tree:

A binary tree is a perfect binary tree in which all non leaf nodes have 2 children & all leaf nodes are at same level

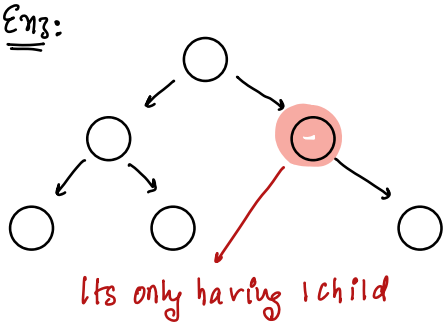
Ex1:



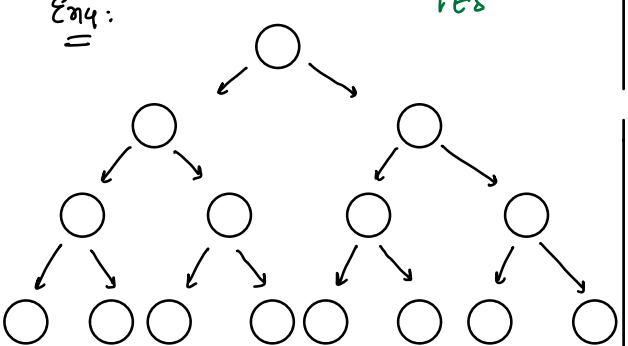
Ex2: Yes



Ex3:



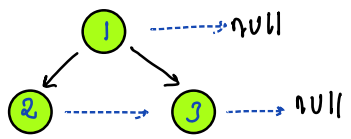
Ex4: Yes



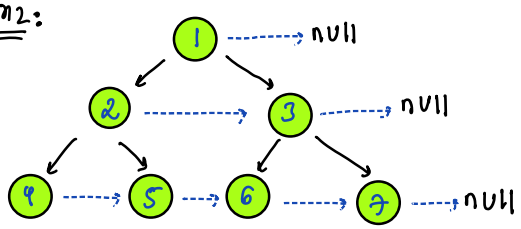
Fill next in Perfect Binary Tree : Every node should point to next node horizontally (Expected SC: O(1))

```
class Node {
    int data
    Node left, right, next;
    Node(n) {
        data = n
        left = null
        right = null
        next = null
    }
}
```

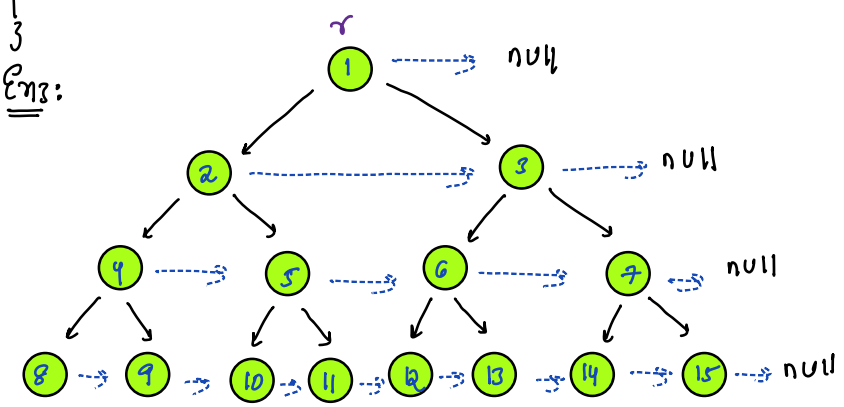
Ex1:



Ex2:



Ex3:




```
Node fillNext(Node root) {
    if(root==null) {return}
    Node t = root;
    while(t.left != null) {
        Node s = t
        while(t != null) { // fill next only for 1 level
            t.left.next = t.right
            if (t.next != null) {
                t.right.next = t.next.left
            }
            t = t.next
        }
    } t = s.left
```

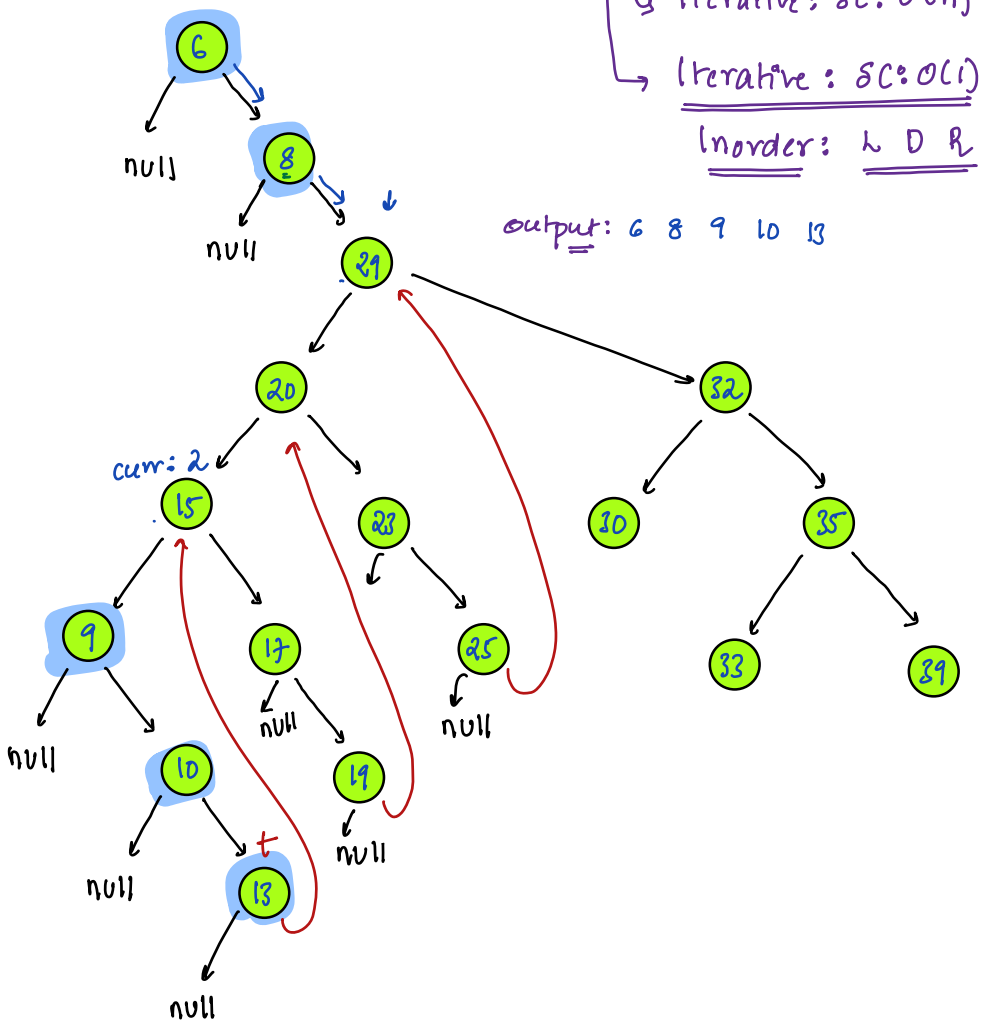
TODO: If not a perfect binary Tree how to do ?

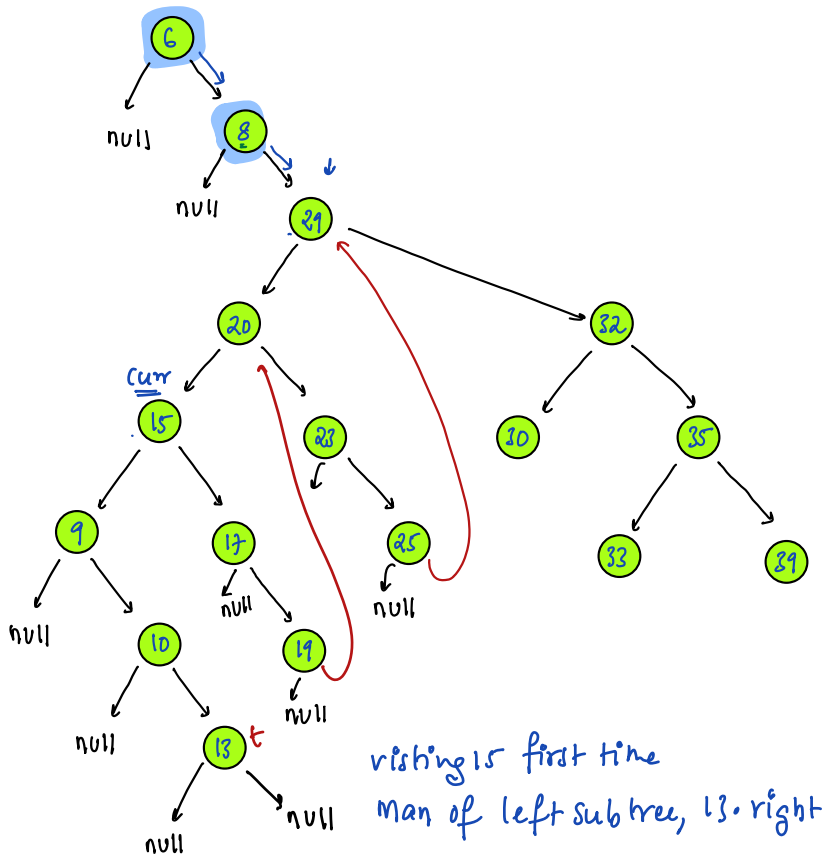
TC: O(N)
SC: O(1)

→ Morris Inorder traversal :

↳ Inorder : → Recusive : SC: O(H)
→ Iterative : SC: O(H)
→ Iterative : SC: O(1)

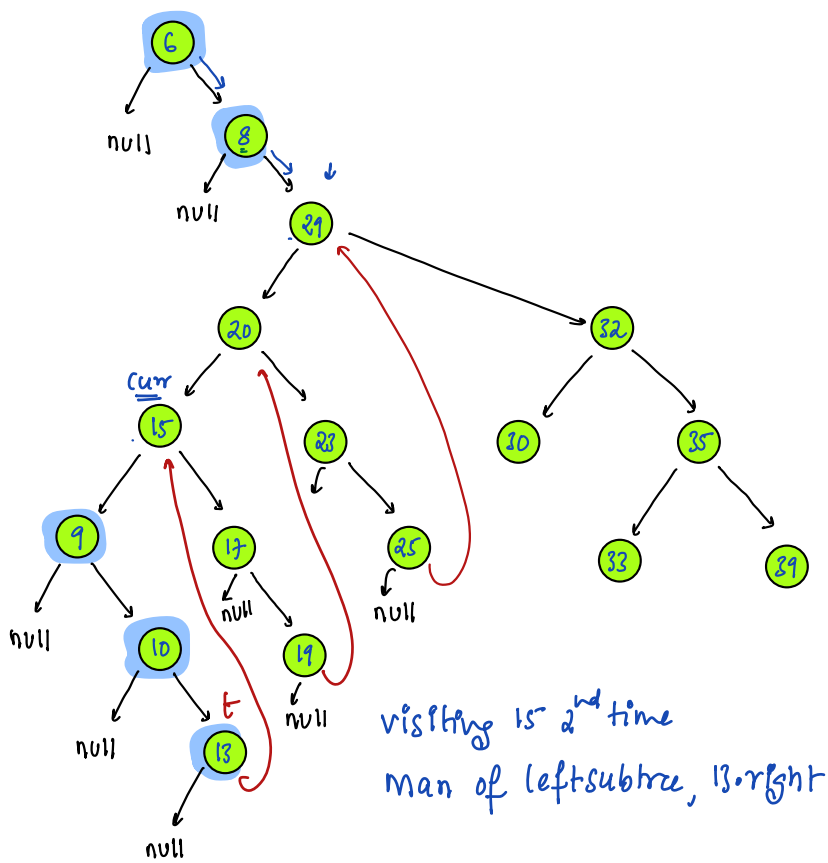
Inorder: L D R





visiting 15 first time

Man of left subtree, 13.right = null



visiting 15 2nd time

Man of leftsubtree, 13.right = cur

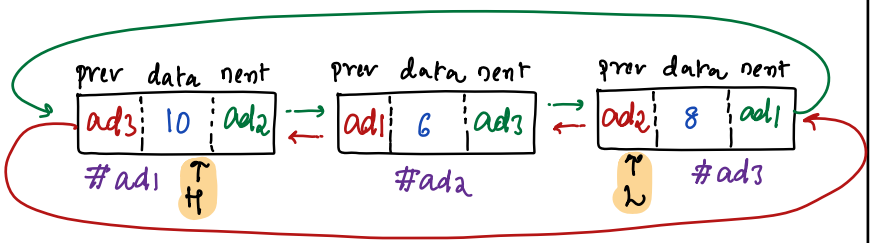
```
void inorder(Node root){   TC: O(N)  SC: O(1)
                           → Exact code will for any Binary Tree
                           → No need of any code change
    Node curr = root;
    while(curr != null){
        if(curr.left == null){ // no left, ✗ ✗ ✗
            print(curr.data)
            curr = curr.right
        }
        else{
            obs1: if curr visits a node for first time
                  right child of rightmost of its LST = null
            obs2: if curr visits a node for 2nd time
                  right child of rightmost of its LST = curr
            Node t = curr.left // We are iterating in LST to get rightmost
            while( t.right != null  or  t.right != curr){
                t = t.right
            }
            if(t.right == null){ // curr visited node 1st time
                t.right = curr
                curr = curr.left
            }
            else{ // curr visited node 2nd time
                print(curr.data)
                curr = curr.right
                t.right = null
            }
        }
    }
}
```

10:40 → 10:50 pm

```
class Node {
    int data

    Node(int x) {
        data = x
        prev = null
        next = null
    }
}
```

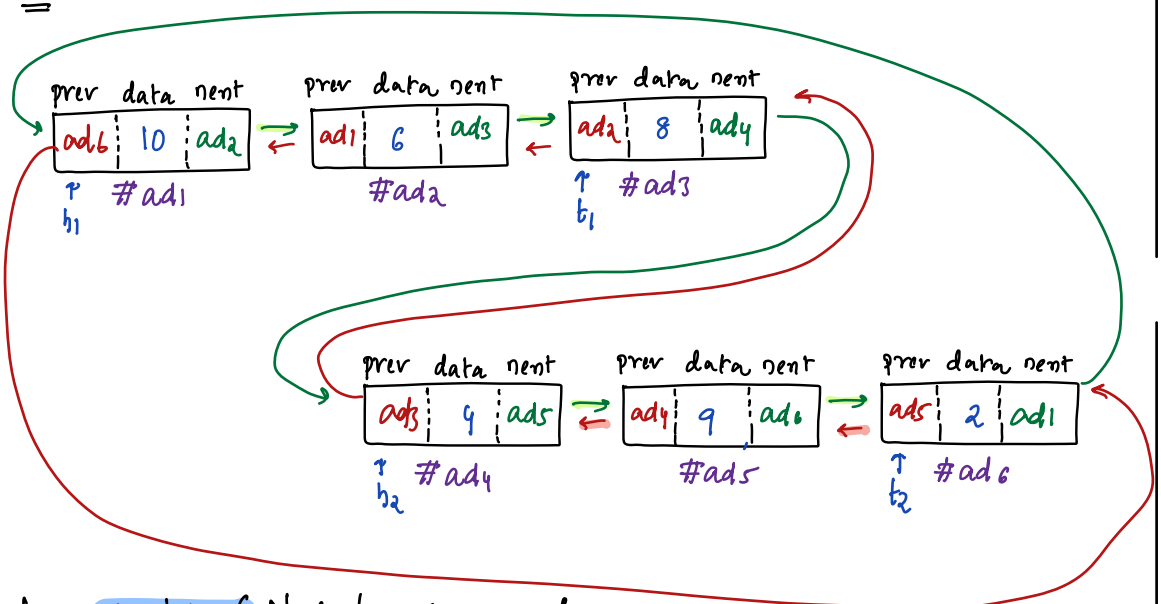
Circular double linked list:



Note: In Circular double linked list, head node is given

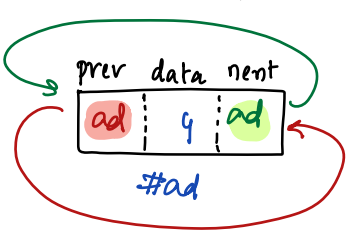
→ 1st node prev = last node & last node next = first node

Ex1: Given 2 circular double linked list combine them



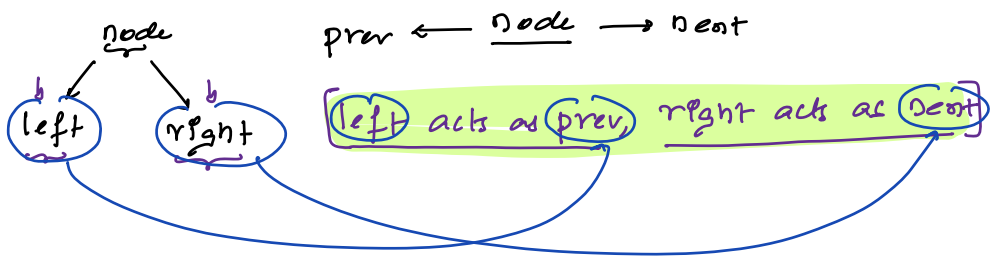

```
Node combine(Node h1, Node h2) {        TC: O(1)  SC: O(1)
    if (h1 == null) { return h2 }
    if (h2 == null) { return h1 }
    Node t1 = h1.prev
    Node t2 = h2.prev
    t1.next = h2
    h2.prev = t1
    h1.prev = t2
    t2.next = h1
    return h1
}
```

Note: Single node in
↳ Circular Double linked list

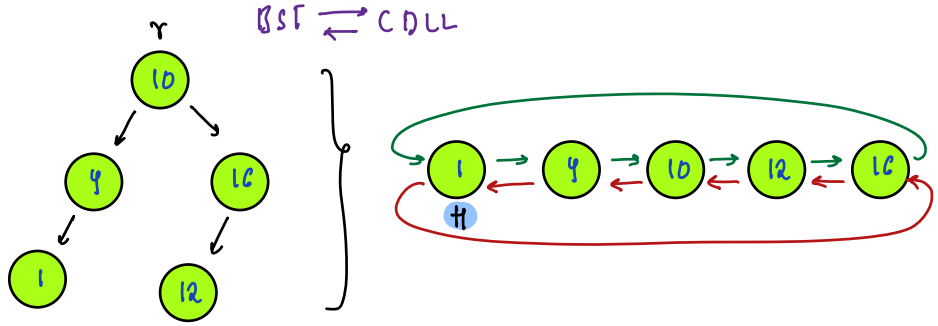


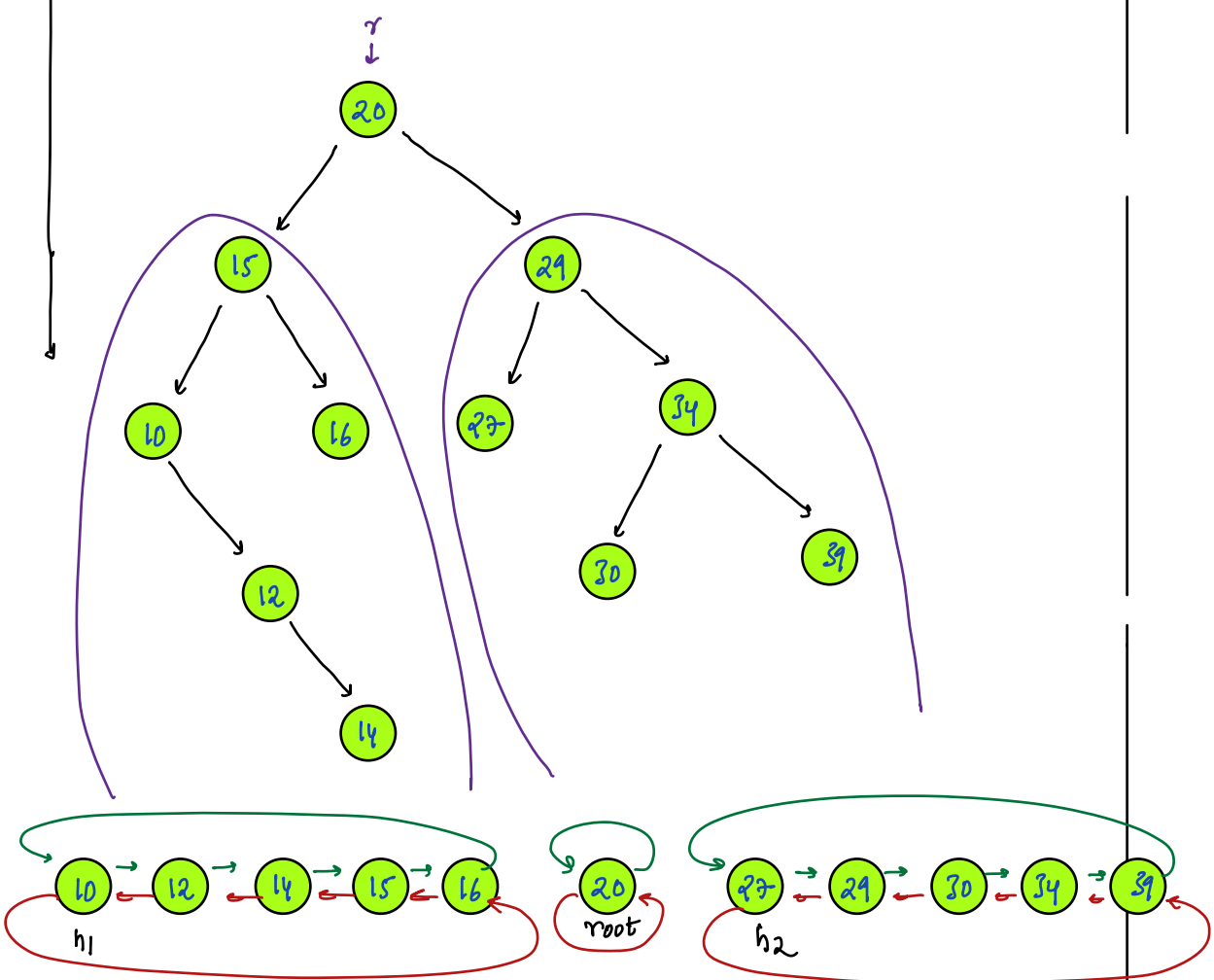
// Obs: prev & next are itself

38) Given a BST convert into a {sorted circular double linked}q return q head node



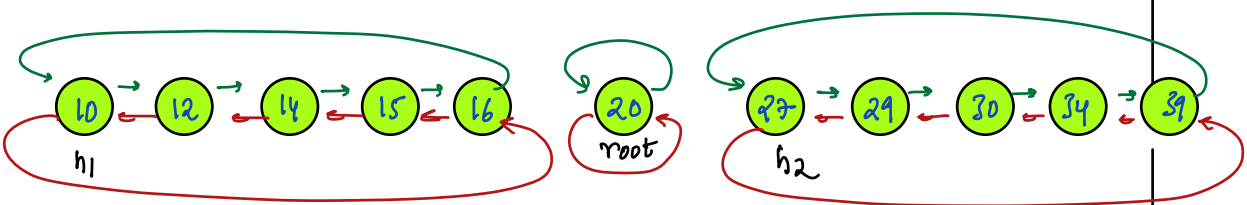
Ex1:



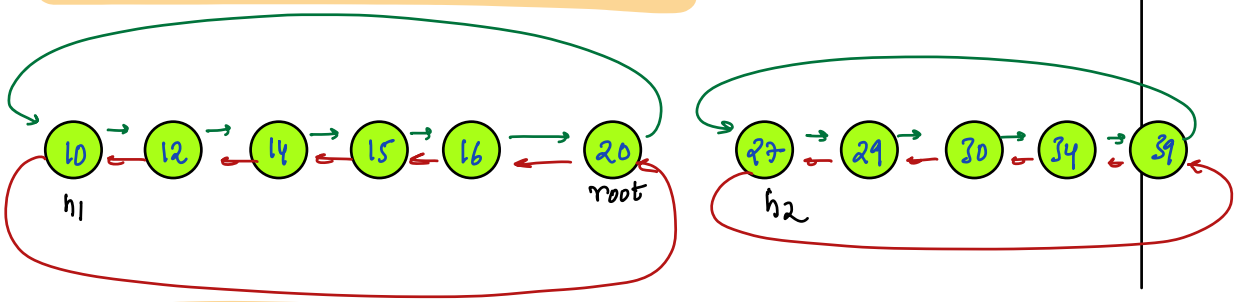


Ass: Given root of BST, convert into SCDLL & return head node.

```
Node BST2CDLL(Node root){   TC: O(N)  SC: O(H)
    if(root==null){ return null}
    Node h1 = BST2CDLL(root.left) // LST → SCDLL & return head node
    Node h2 = BST2CDLL(root.right) // RST → SCDLL & return head node
    root.right = root
    root.left = root          } //Convert root → CDLL
    h1 = combine(h1, root) // combine 2 CDLL & return head of CDLL
    h1 = combine(h1, h2) // combine 2 CDLL & return head of CDLL
    return h1
}
```



combine above 2 CDLL



combine above 2 CDLL