- Today's Content

Invert Binary Tree

Equal tree partition

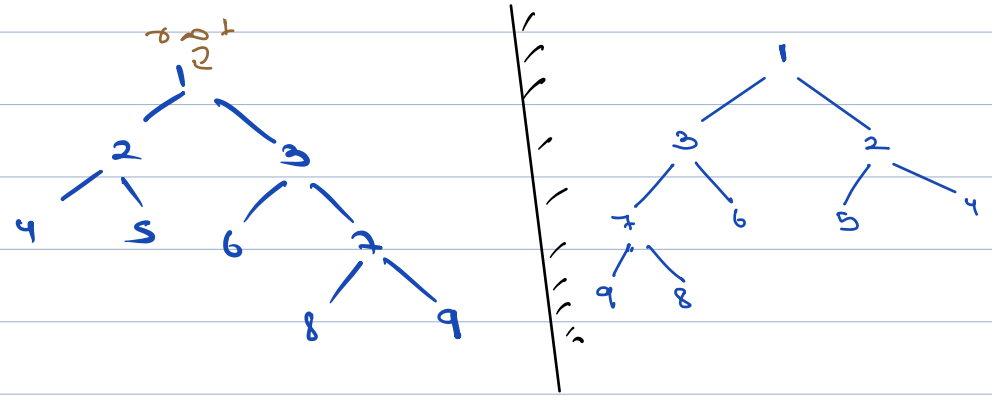
Next pointer B.T.

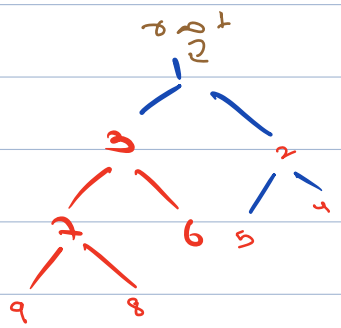
Root to leaf path sum = k.

Diameter of B.T.

Ques) Invert a B.T.

I/P →



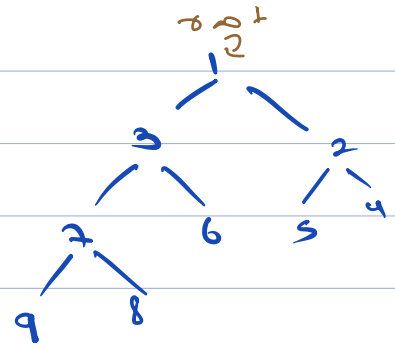


Sol:- At, nodes swap l & r child.

```
void invert (root) {
```

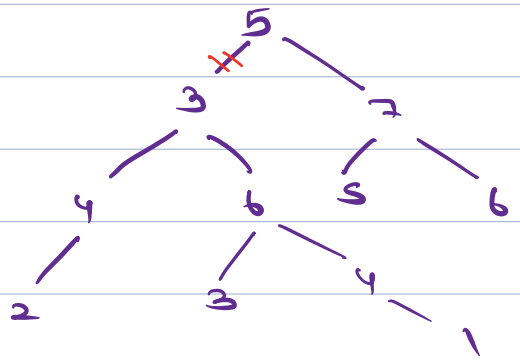
T.C → O(n)
S.C → O(H)

```
    if (root == null) return;

    temp = root.left;
    root.left = root.right;
    root.right = temp;
    invert (root.left);
    invert (root.right);
}
```


1
3
2
7
6
5
9
8

Equal Tree Partition

Ques) Check if it is possible to remove an edge from b.t, s.t, the sum of resultant two trees is equal.

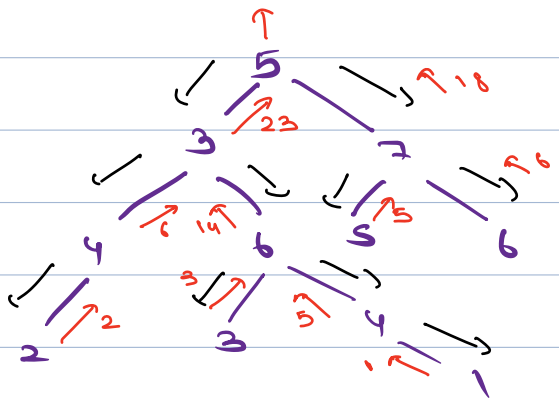


Ans True.

Obs-1 :- If total sum of the Tree is S, both the subtrees would have sum $S/2$.

Obs 2 :- If total sum is odd, return false.

> Check if there's a subtree with sum = $S/2$



Total Sum = 46

```
s = sum(root);
if (s%2 == 1) { return false }
```

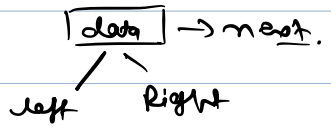
T.C → O(n)
S.C → O(H)

```
boolean ans = false;
int check(root) {
    if (root == null) { return 0 }
    l = check(root.left);
    r = check(root.right);
    if (l == s/2 || r == s/2) {
        ans = True;
    }
    return l + r + root.val;
}
```

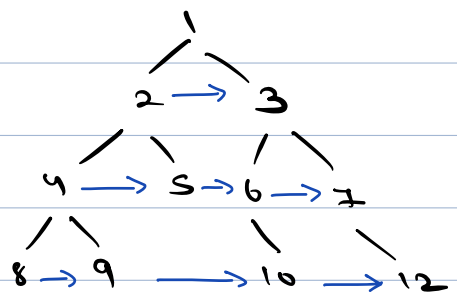
```
class node {
    int data;
    node left;
    node right;
    Node next;
}
```

Ques). [Next pointer in B.T]

Initially each node's next pointer points to null.



update each node's next pointer to point to the next node in same level.




```
void levelOrder(Node root){
    Queue<Node> q;
    q.add(root);
    while(q.size()>0){
        int n= q.size();
        for(i=1; i<=n; i++){
            Node front = q.peek();
            q.remove();
            if(front.left!=null){
                q.add(front.left);
            }
            if(front.right!=null){
                q.add(front.right)
            }
        }
    }
}
```

```
if(i!=n){
    front.next= q.front();
}
```

T.C → O(n)

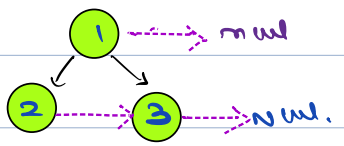
S.C → O(n)

Ques Fill next in **Perfect Binary Tree**.

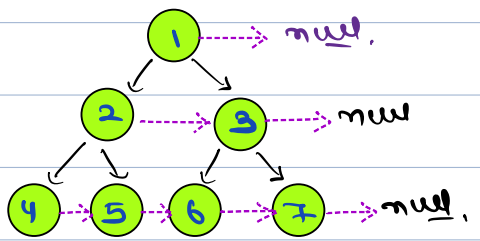
Expected S.C → O(1).

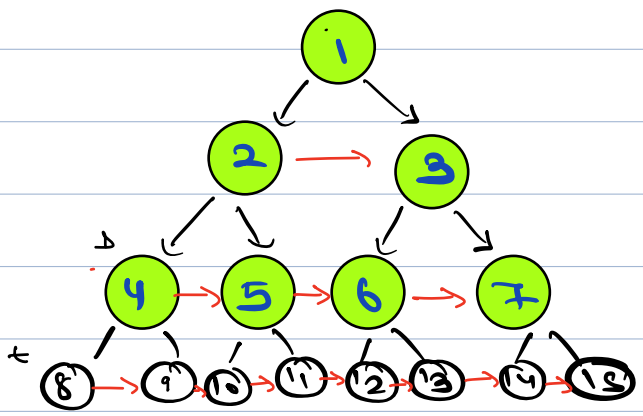
```
class Node {
    int data;
    Node left, right, next;
    Node(x) {
        data = x
        left = null
        right = null
        next = null
    }
}
```

Ex1 :-



Ex2 :-





T.C → O(n)
S.C → O(1)

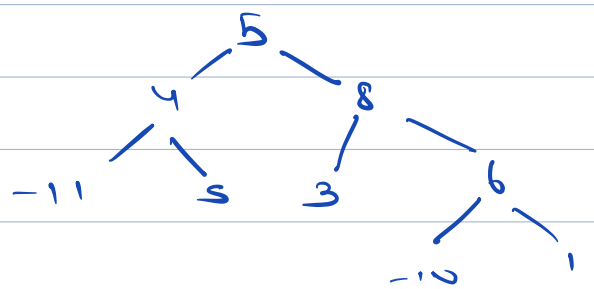
```
while (t.left != null) {
    node s = t;
    while (t != null) {
        t.left.next = t.right;
        if (t.next != null) {
            t.right.next = t.next.left
        }
        t = t.next;
    }
    t = s.left;
}
```

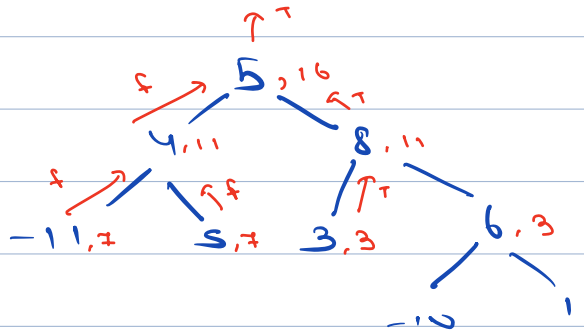
Ques. check if given binary Tree, has any one root to leaf path sum = k.

k = 16 → True.

k = -2 → True

k = 9 → True

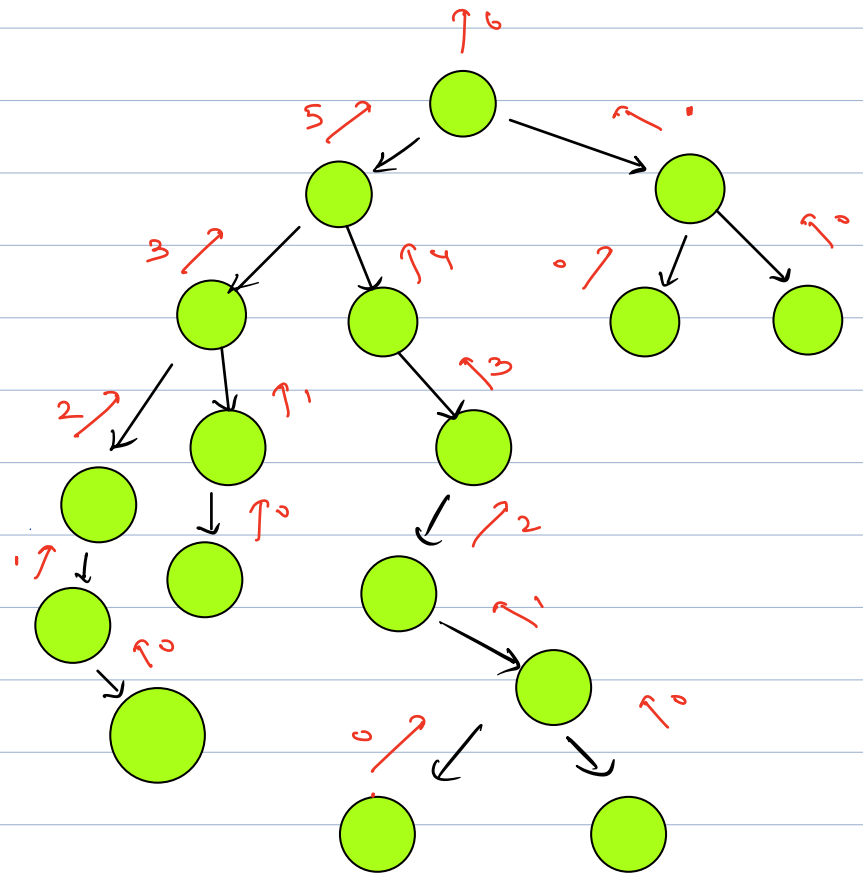




T.C -> O(n), S.C -> O(H)

```
boolean check(root, k) {
    if (root == null) { return false }

    if (root.left == null && root.right == null) {
        return root.data == k
    }

    return check(root.left, k - root.data) ||
           check(root.right, k - root.data);
}
```

Height of a B.T. → In terms of Edges



I, R

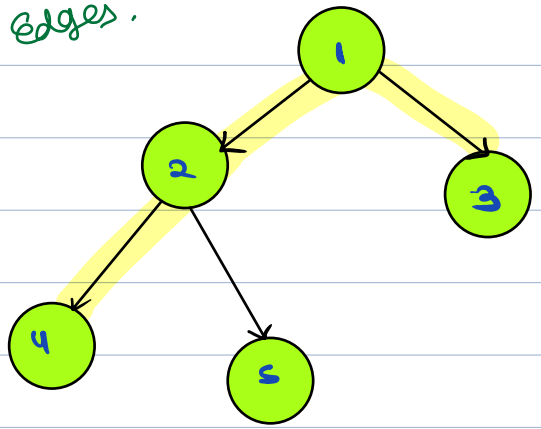
T.C → O(N)
S.C → O(H)

```
int height(Node root) {
    if (root == null) { return -1; }
    lh = height(root.left);
    rh = height(root.right);
    return max(lh, rh) + 1;
}
```

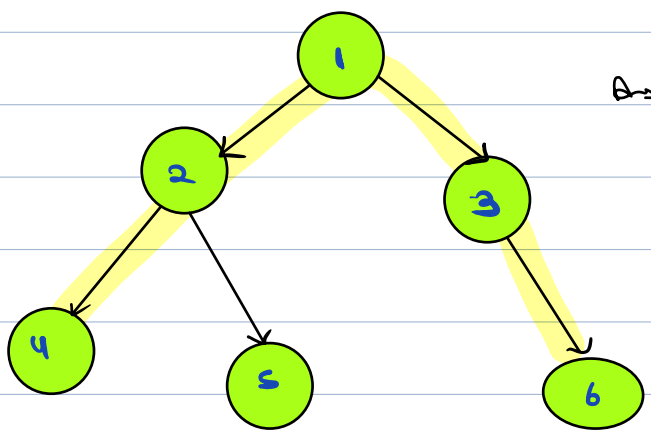
3

Diameter :- It is no. of edges on longest path b/w two nodes in a binary tree.

diameter in terms of edges.

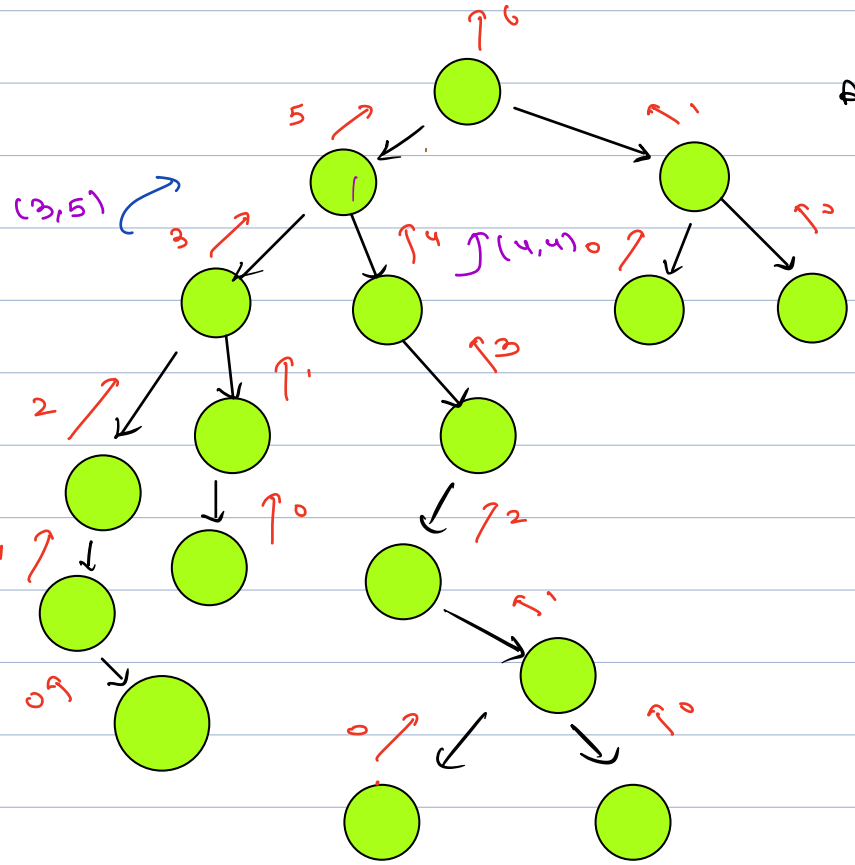


Ans = 3.



Ans = 4.

diameter = 1 + 3 + 2.



Ans = 9

```
diameter = -1;
```

```
int height(Node root) {
    if (root == null) return -1;
    lh = height(root.left);
    rh = height(root.right);
    diameter = max(diameter, lh+rh+2);
    return max(lh,rh)+1;
}
```

3

Pair { int height;
int dia;
}

3

Pair height (Node root) {

if (root == null) { return new Pair(-1, 0); }

lp = height (root.left);

rp = height (root.right);

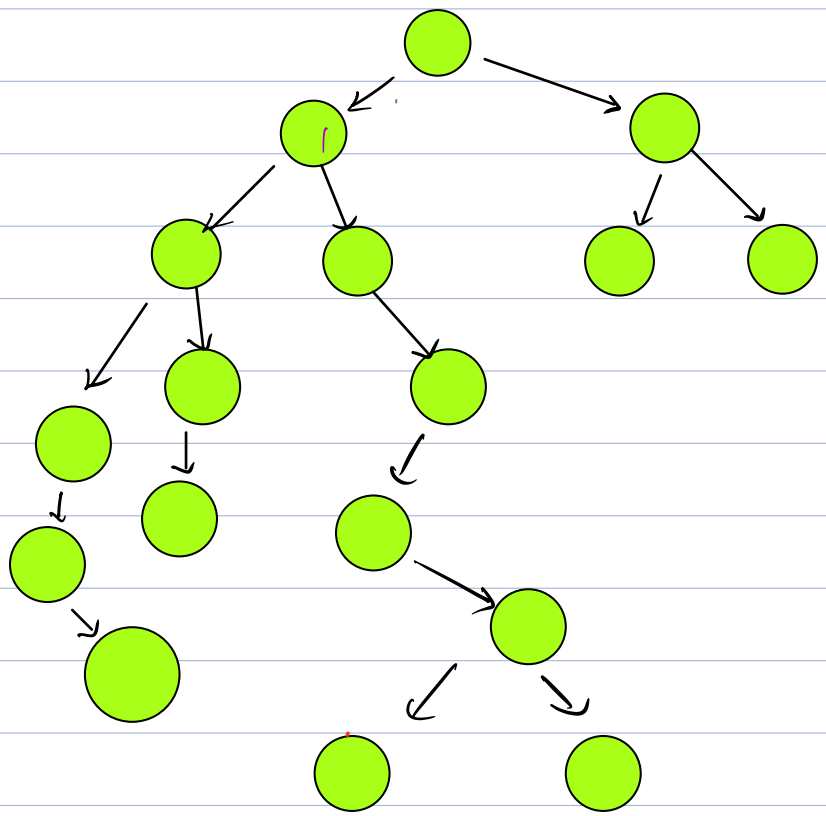
mydia = Max (lp.dia, rp.dia, (lp.h + rp.h + 2));

return new Pair (Max (lp.h, rp.h) + 1, mydia);

3

T.C → O(n)

S.C → O(h)



int gvar = 0;

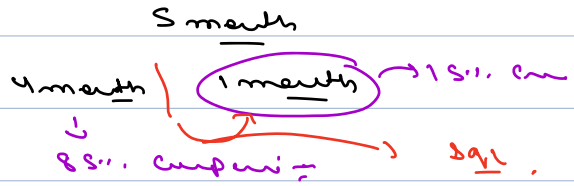
xyz ( ) {

}

DSA len by 1 month

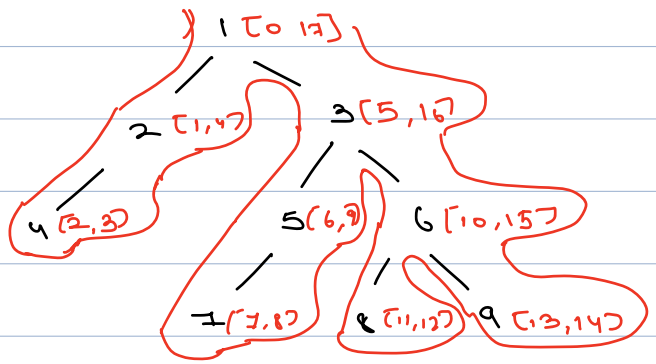
Adv. DSA

Tried,



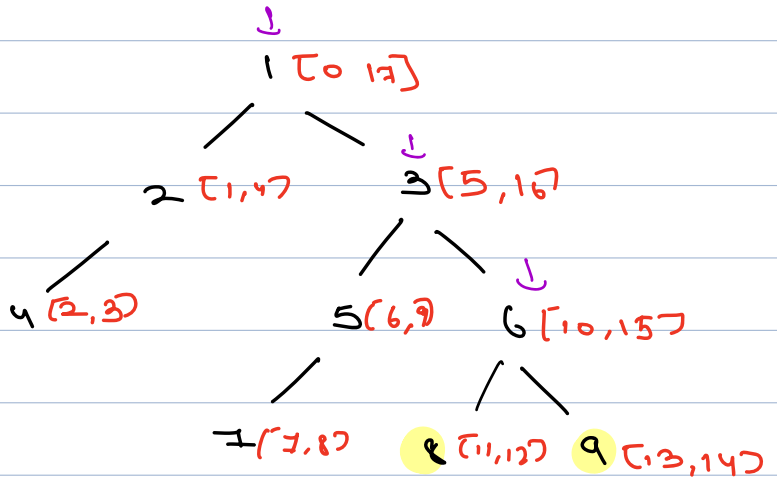
In time - Out time Concept

t = 0 1 2 3 4 5 6 7 8




```
T=0;
void Traversal (root) {
    if (root == null) {return;}
    in(root) = T;
    T++;
    Traversal (root.left);
    Traversal (root.right);
    out(root) = T
    T++;
}
```

T.C -> O(n)



in(x) < in(y)
out(x) > out(y)
} x is ancestor of y.

T.C → O(H)

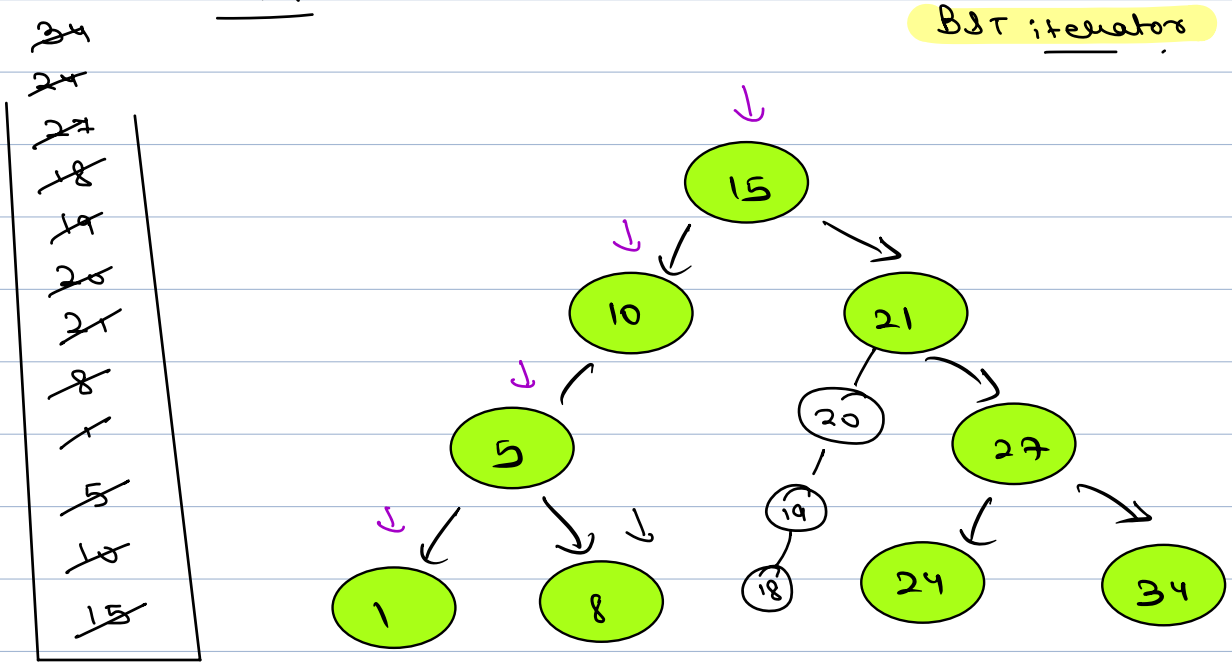
```
curr = root;
while (curr != null) {
    if (curr.left is ancestor of x & y) {
        curr = curr.left;
    }
    else if (curr.right is ancestor of x & y) {
        curr = curr.right;
    }
    else {
        return curr;
    }
}
```

Q queries :-

O(n) + O(h) * Q
for Queries.

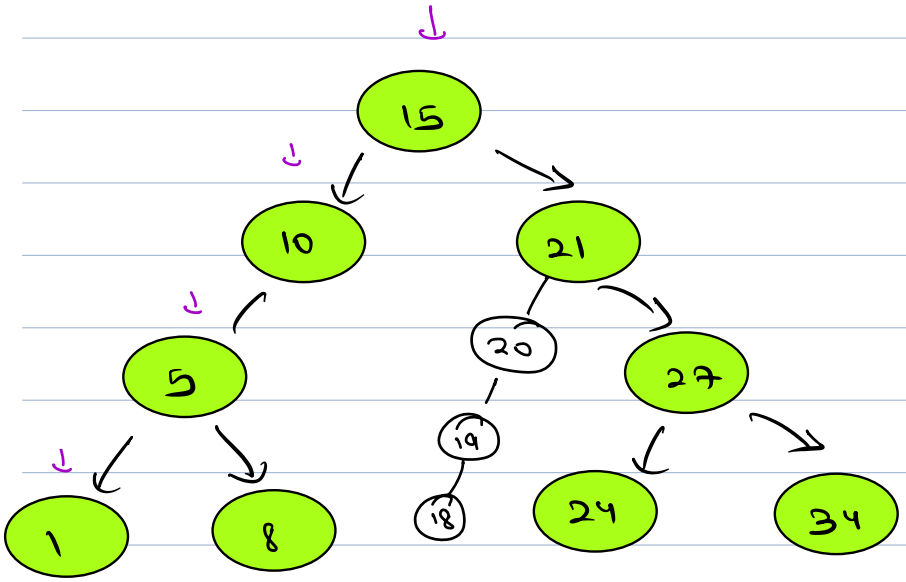
LDR

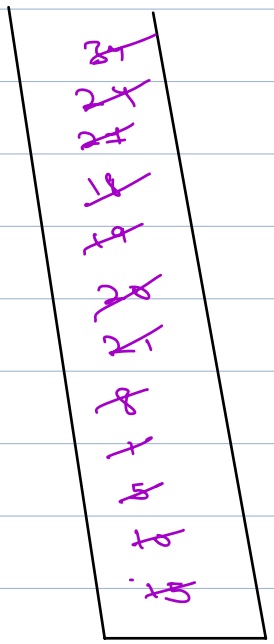
BST iterator



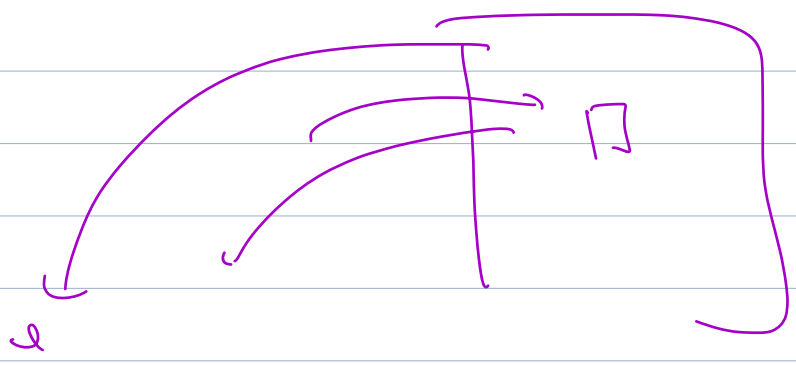
1 5 8 10 15 18 19 20 21 24 27 34

```
Next() {

    Stack <> ( );
    while ( curr != null ) { st.push(curr); curr = curr.left; }
    Node temp;
    while (st.size() > 0) {
        temp = st.top(); st.pop();
        if (temp.right != null) {
            ...
        }
    }
    return temp;

hasNext() {

}
```





1 5 8 10 15 18 19 20 21 24 27 34



q = root();

r = RemoveMin()

if r < q