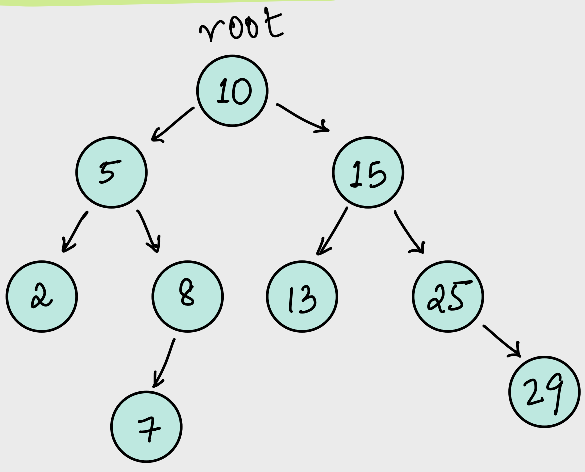
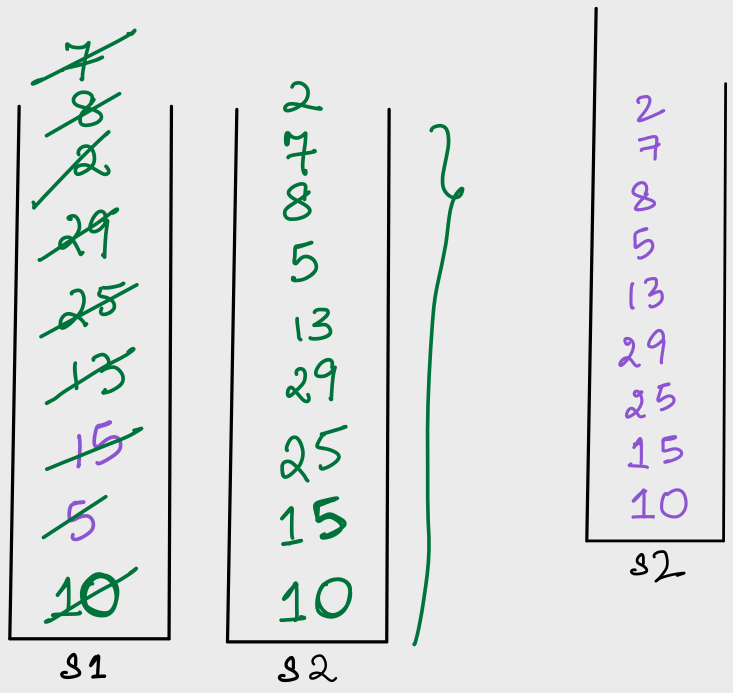
Que 1 Iterative Posforder

LRD





Posforder →

2 7 8 5 13 29 25 15 10

Note: If we pop from S2 it should give the answer.

```
void postorder( Node root) {
    stack <Node>  S1, S2 ;
    S1.push(root)
    while ( S1.size() > 0 ) {
        Node t = S1.top ( )
        S1.pop()
        S2.push(t)
        if ( t.left) {
        |   S1.push(t.left)
        }
        if (t.right) {
        |   S1.push(t.right)
        }
    }
    // print stack 2. => TODO
}
```

TC : O(N)

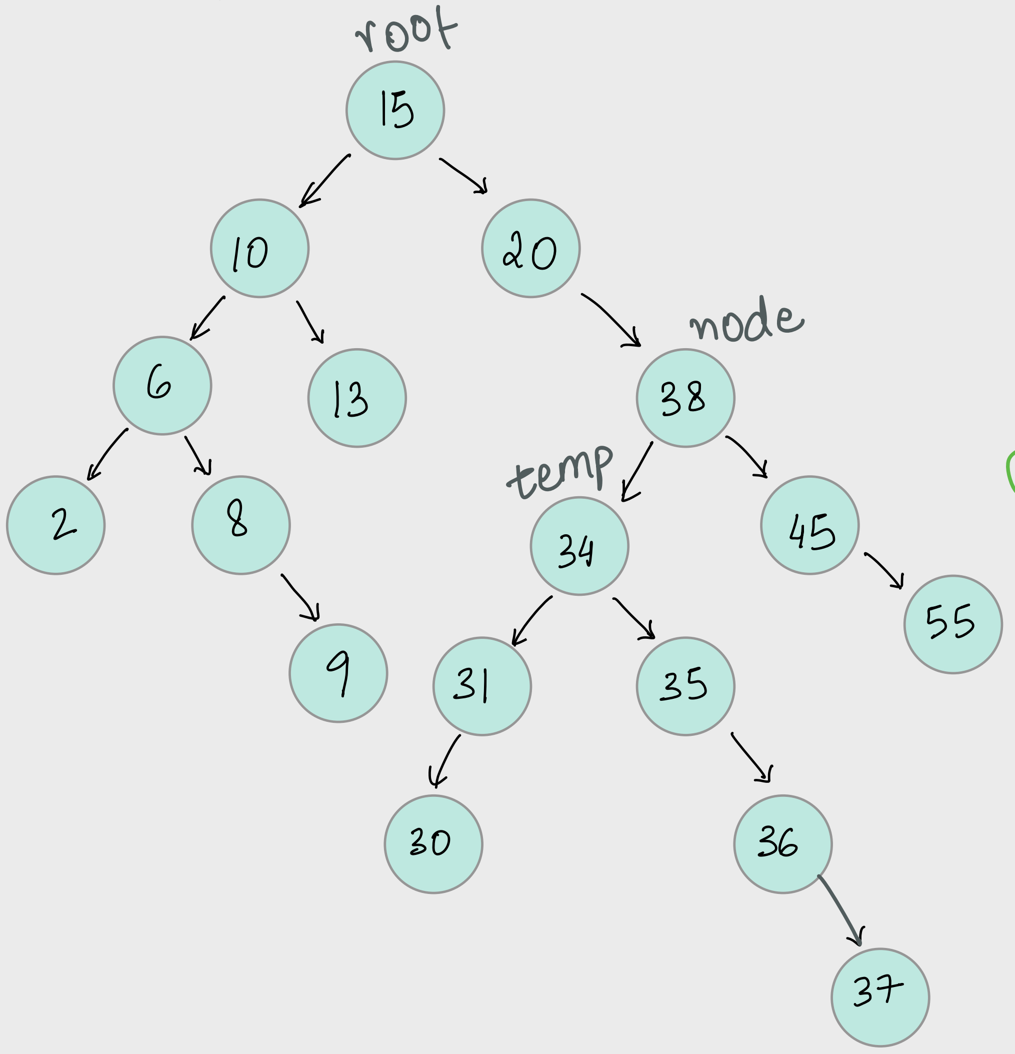
SC : O(N)

DIY / TODO

Do it using single stack.

Q. Given node, get rightmost node of inorder in LST.

Binary Search Tree



15 ⇒ 13

38 ⇒ 37

6 ⇒ 2

floor .

```
temp = node.left
while (temp.right != NULL)
{
    temp = temp.right
}
return temp.data
```

floor → largest value ≤ given val

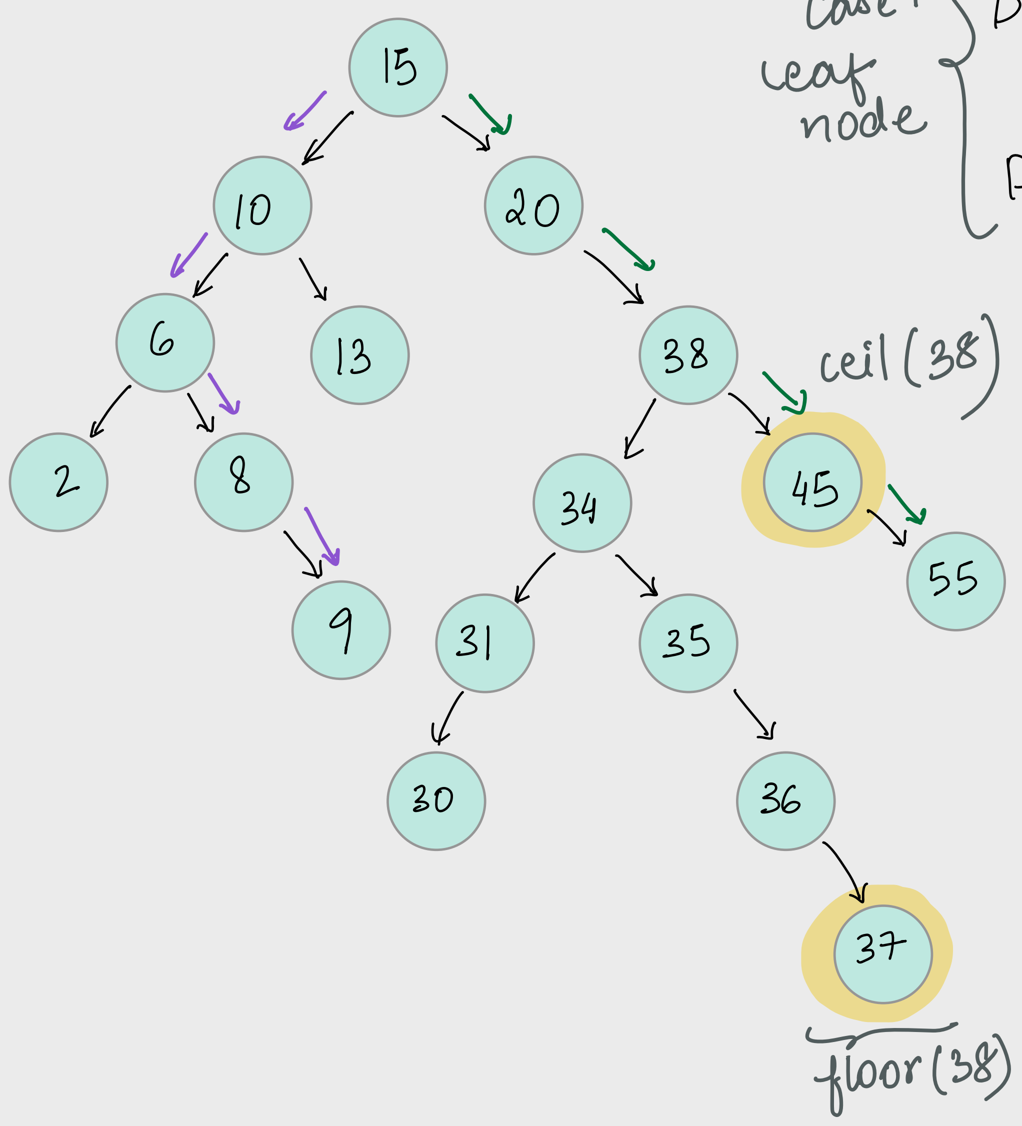
ceil → smallest val ≥ given val

DIY.

Deletion in BST

{Replacing the data is allowed if needed}

After deleting, return root Node.

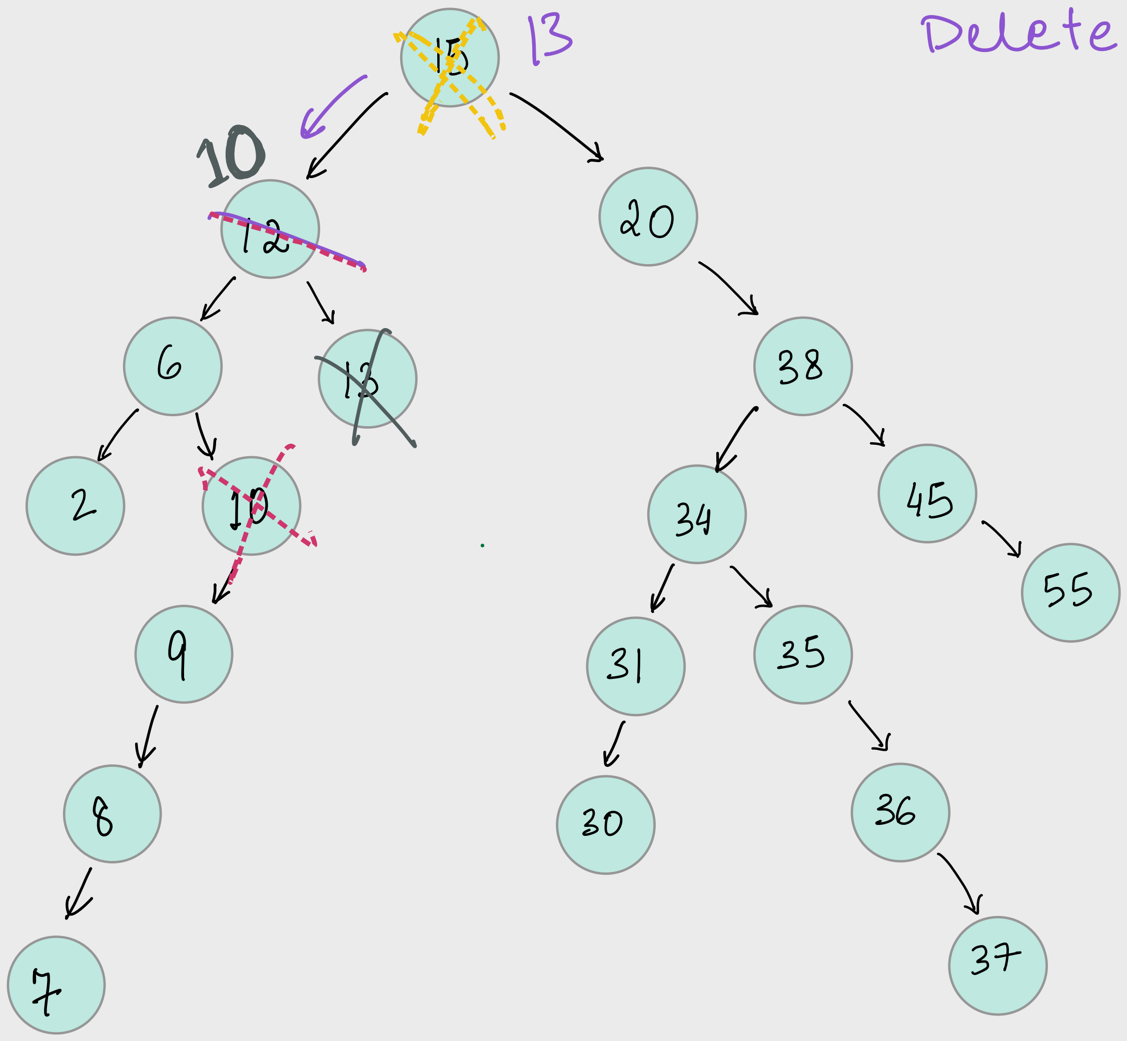


case 1 leaf node
- Delete 9 — 8.right = NULL
- Delete 55 — 45.right = NULL

case2 One child
- Delete 20 — 15.right = 20.right
- Delete 45 — 38.right = 45.right

Case 3 Both children
- 38 : we can replace 38 with either 37 or 45.

1. Find floor of 38
2. Replace 38 with 37
3. Delete 37.



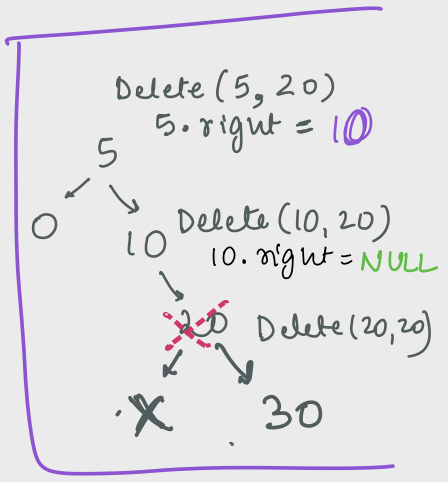
Delete 12

Find floor(12) → 10
Replace 12 with 10
Delete 10.

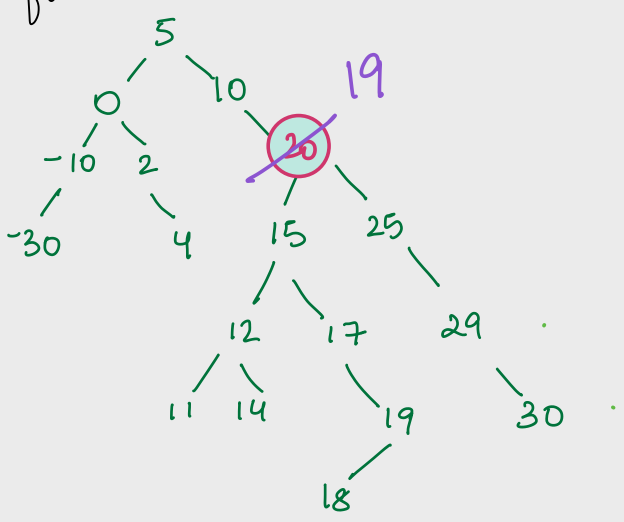
6.right = 10. left

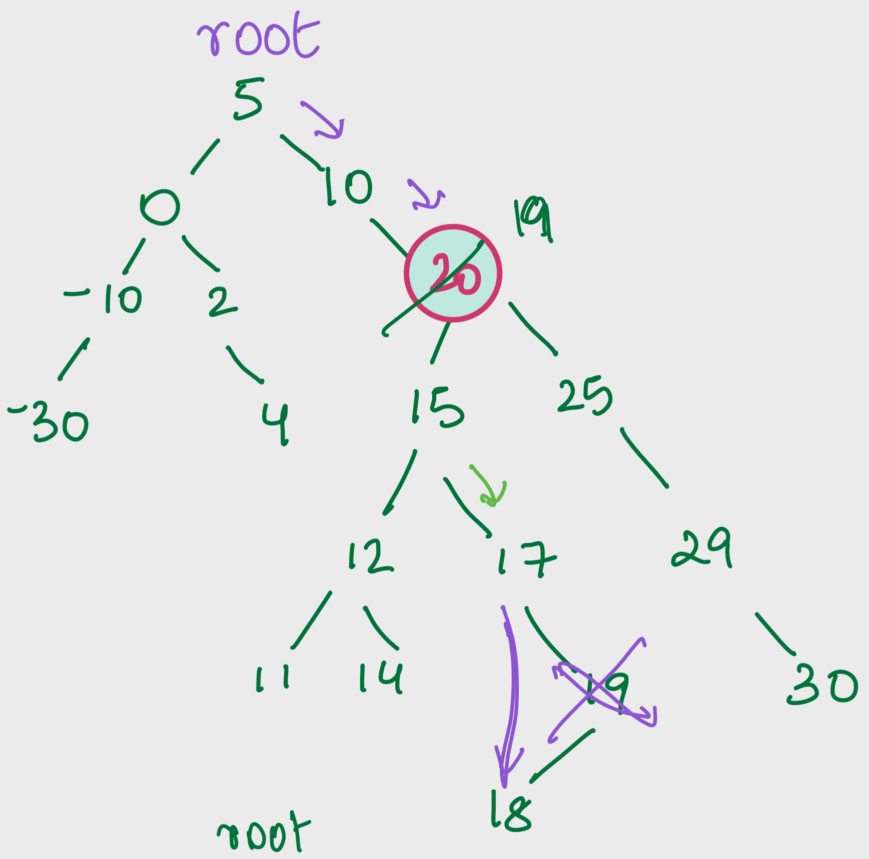
10 cannot have any child on right
Because then floor of 12 will not be 10 but the rightmost node

Delete 15:
floor(15) → 13
Replace 15 with 13
Delete 13



```
Node Delete(Node root, int K) {
    if (root == NULL) return NULL
    if (root.data == K) {
        //leaf node
        if (root.left == NULL && root.right == NULL) {
            return NULL
        }
        // If single child
        if (root.left == NULL || root.right == NULL) {
            if (root.left == NULL) {
                return root.right
            }
            if (root.right == NULL) return root.left
        }
        //Both children
        int x = floor(root)
        root.data = x
        root.left = Delete(root.left, x)
        return root
    }
    else if (root.data > K) {
        root.left = Delete(root.left, K)
    }
    else {
        root.right = Delete(root.right, K)
    }
    return root
}
```

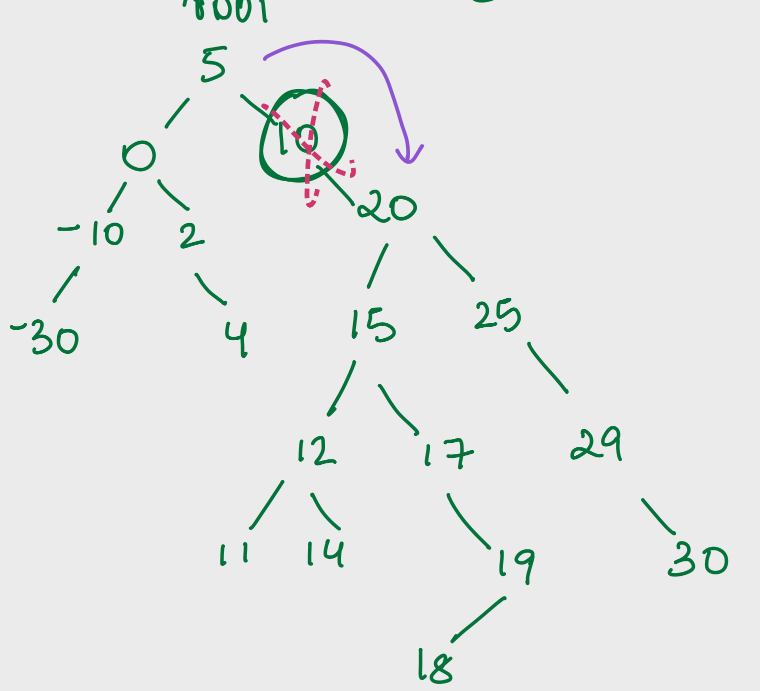




~~(19, 19)~~
~~(17, 19)~~
~~(15, 19)~~
~~(20, 20)~~ 19
~~(10, 20)~~
~~(5, 20)~~

17.right = 18
15.right = 17
19.left = 15
10.right = 19
5.right = 10

5



~~10, 10~~
~~5, 10~~

5.right = 20

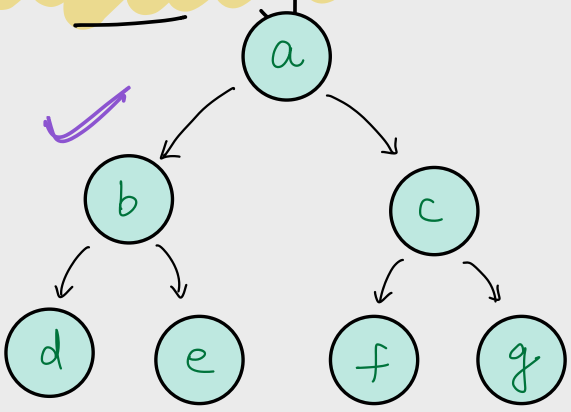
5

TC: O(H)

SC: O(H)

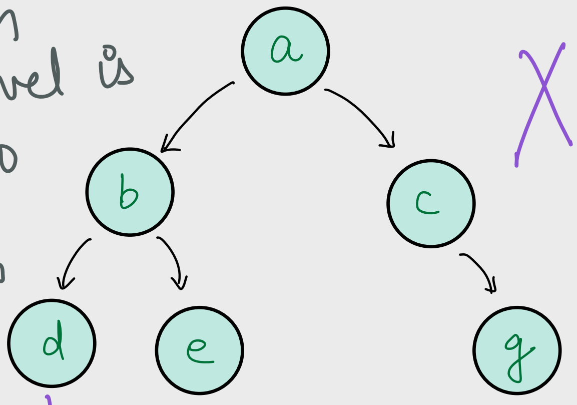
Break: 10 :40

Perfect Binary Tree

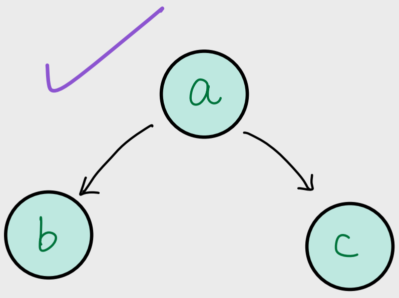


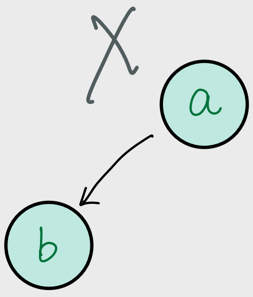
Each node has 2 children or not children

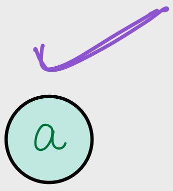
Only last level is allowed to have 0 children



All the levels must be filled.



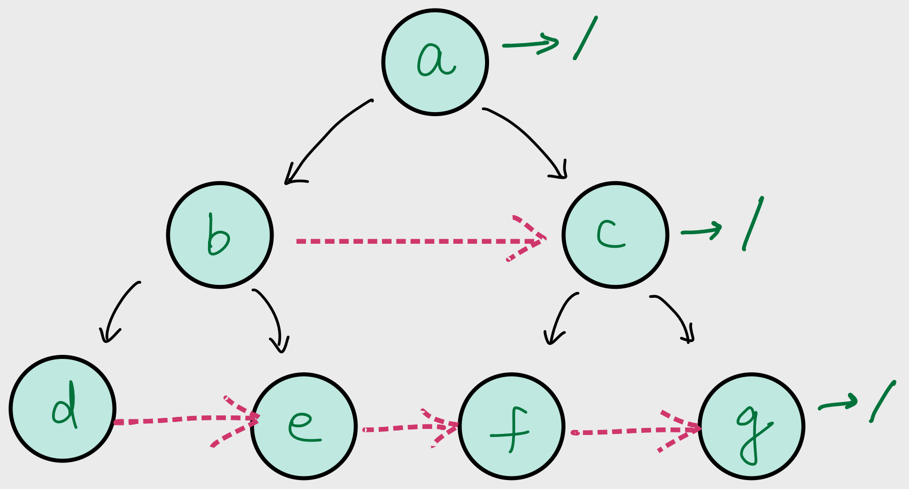




Amazon

Que. Given a perfect tree,

```
class Node {
    Node left
    Node right
    Node side
}
```



Approach 1 :

We can do level order traversal using Queue {

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

~~a~~ ~~NULL~~ ~~b~~ ~~c~~ ~~NULL~~ ~~d~~ ~~e~~ ~~f~~ ~~g~~ ~~NULL~~ }

prev	curr
a	~~a~~ NULL

a.side = NULL

~~NULL~~ ~~b~~	~~NULL~~ ~~b~~ C

b.side = C

~~c~~	~~c~~ NULL

c.side = NULL

~~NULL~~ ~~d~~	~~NULL~~ ~~d~~ e

d.side = e

e	~~e~~ f

e.side = f

f	~~f~~ g

f.side = g

prev	curr
NULL	~~NULL~~ a

```
if (prev == NULL)
        prev = curr
else
        prev.side = curr
        prev = curr
```

~~g~~	~~g~~ NULL

g.side = NULL

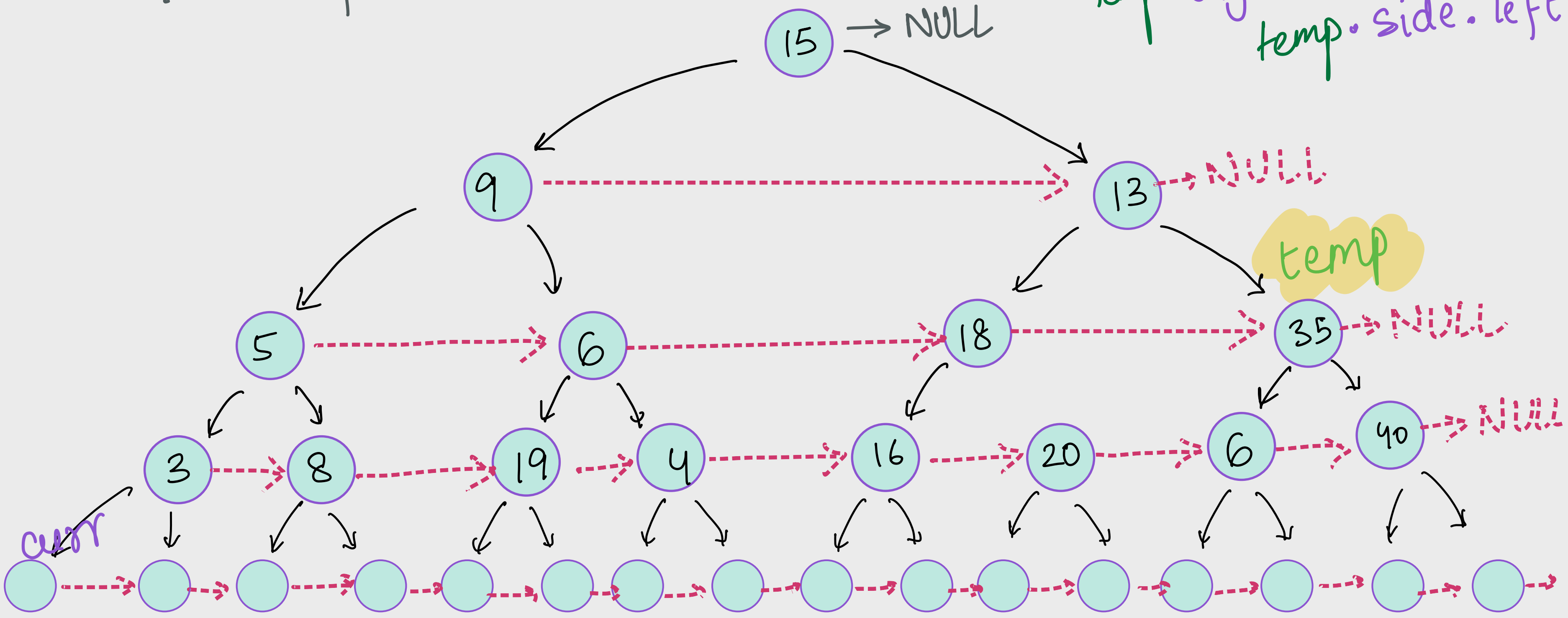
NULL NULL

Use only constant extra space

root

temp.left.side = temp.right

temp.right.side = temp.side.left



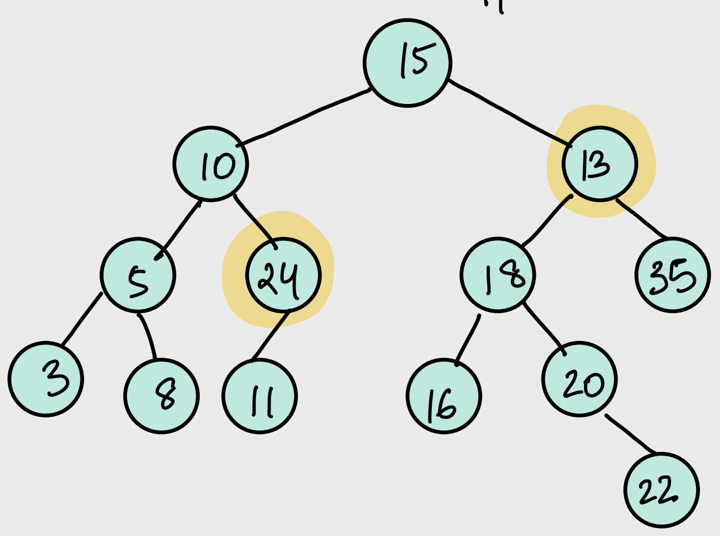
```
void connectSides ( Node root) {
    curr = root

    while(curr.left != NULL){
        temp = curr
        while(temp != NULL){
            temp.left.side = temp.right
            if (temp.side){
                temp.right.side = temp.side.left
            }
            Temp = temp.side
        }
        curr = curr.left
    }
}
```

TC : O(N)

SC : O(1)

Q. Find 2 swapped nodes of BST.



Left to Right: Data ↑

if (A[i] > A[i+1])

⇒ A[i] is out of order

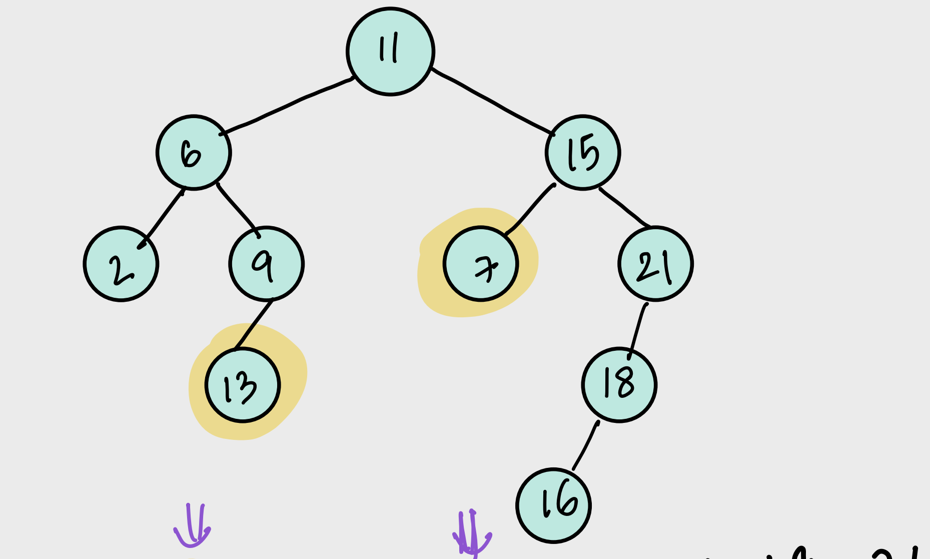
Right to left: Data ↓

if(A[i] < A[i-1])

⇒ A[i] out of order

Inorder: 3 5 8 10 11 24 15 16 18 20 22 13 35

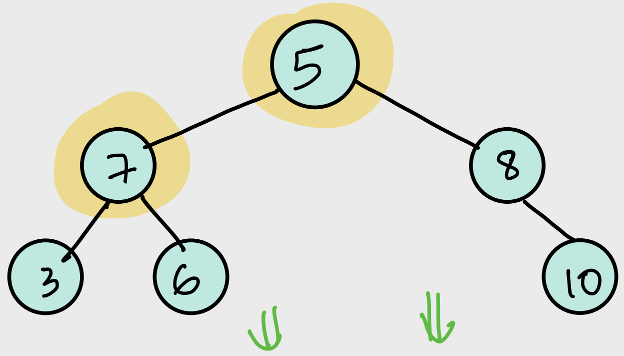
< < < < < > > <



⇓ ⇓

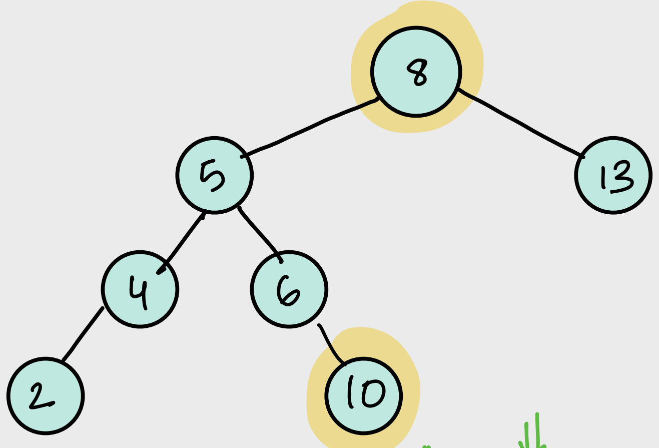
2 6 13 9 11 7 15 16 18 21

< < > > < < < <



⇓ ⇓

3 7 6 5 8 10

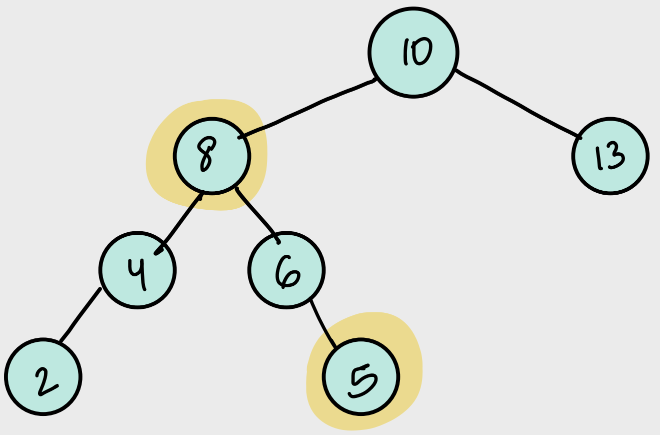


⇓ ⇓

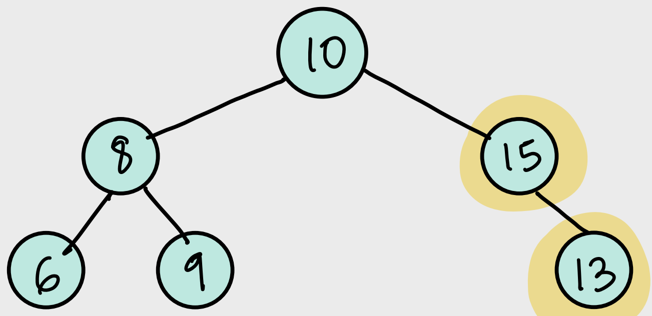
2 4 5 6 10 8 13

< < < < >

> <



2 4 8 6 5 10 13



⇓ ⇓

6 8 9 10 15 13

< < < < >

Find inorder traversal

Try with Morris