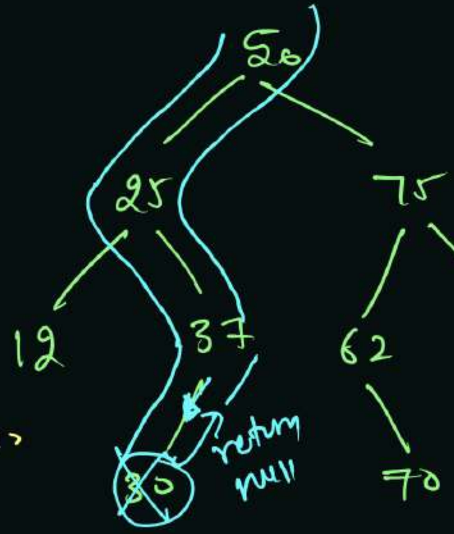
Remove Node From BST

Saturday, 26 June 2021 2:34 PM

if (data is found) →

1. leaf Node] return null
2. Node have only left child] return left child of node.
3. Node have only right child] return right child of node.
4. In the middle, have both left and right child

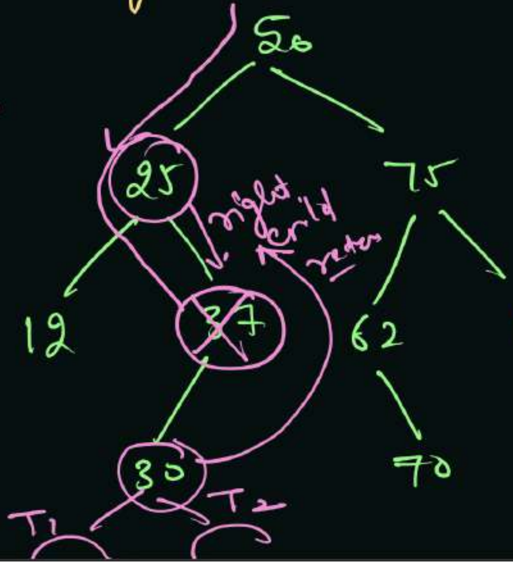
data = 30



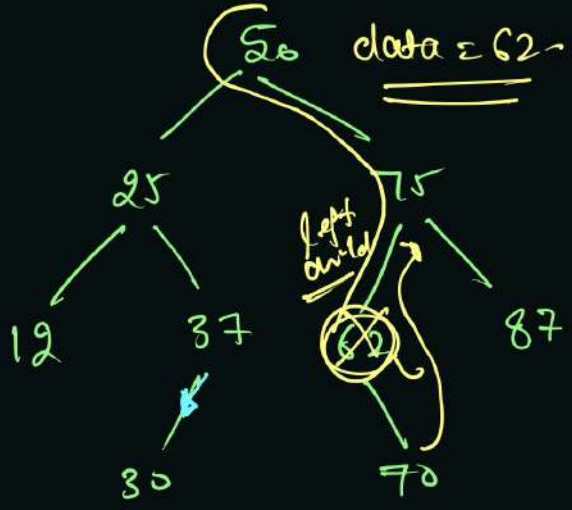
30 is leaf Node

data = 37

have left child



data = 62



have right child

Node having both left and right child

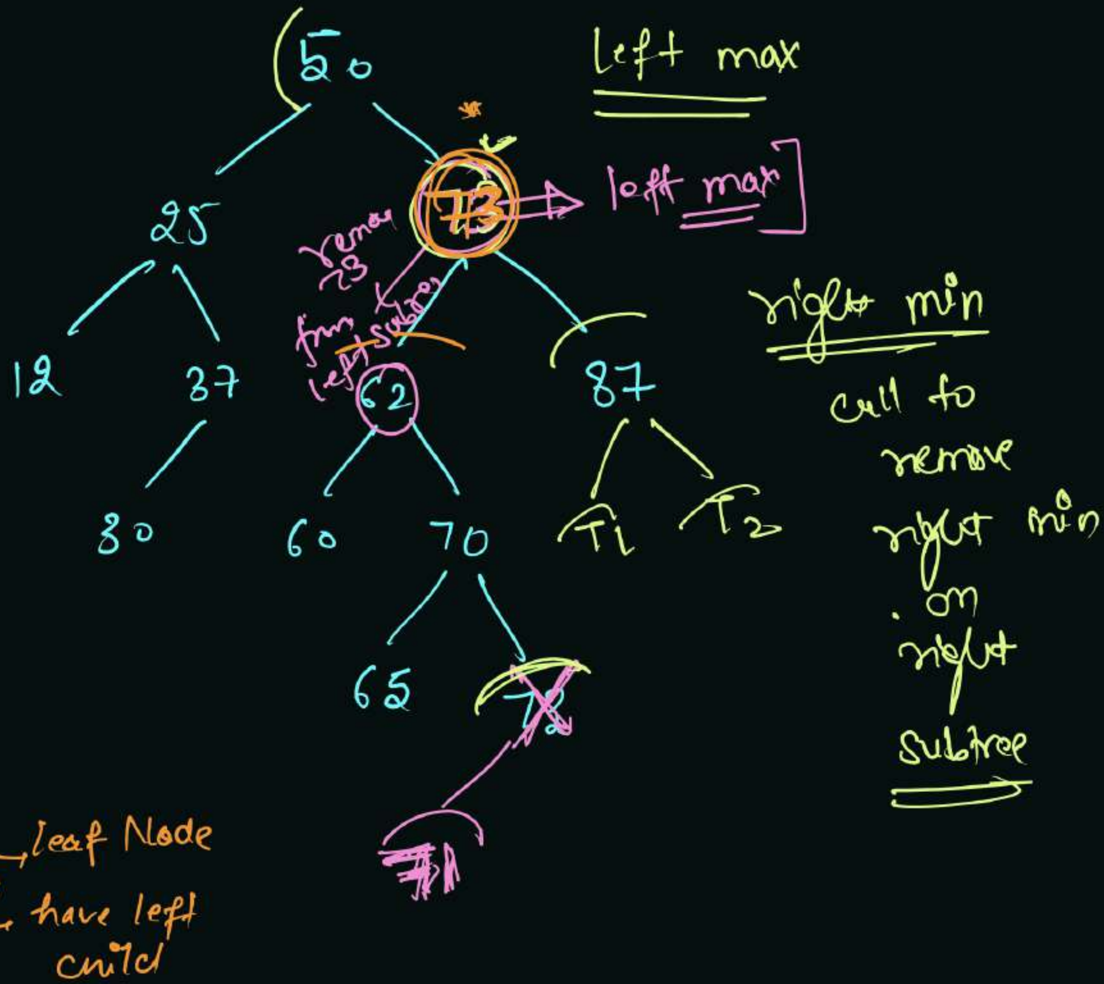
data = 75.

BST → property -] Maintain
Maintain] BST property.

steps ① max from left subtree

② Set the data of node as left max

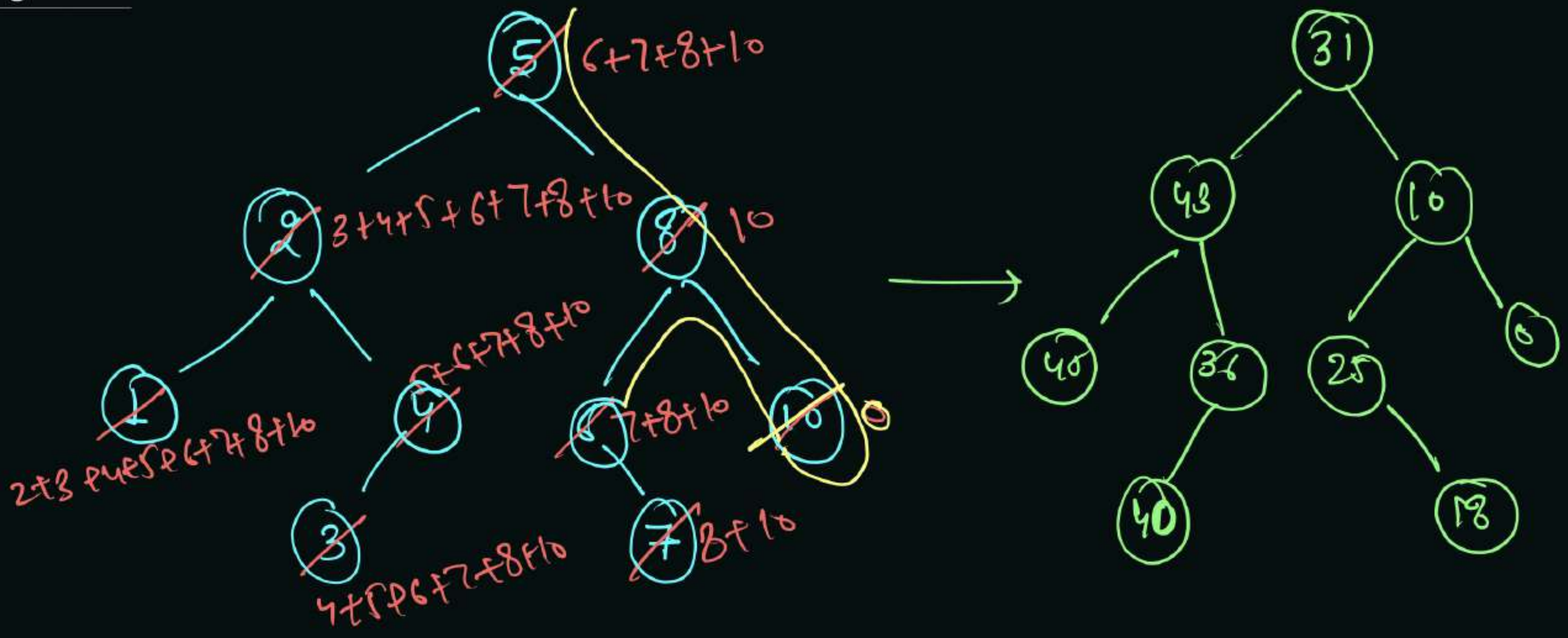
③ Remove max, from left subtree



Replace with Sum of Larger

Saturday, 26 June 2021 2:34 PM

sum = ~~0~~ 10



Space → O(1) Except recursive Space

Time → O(n) → Single traversal on Tree

InOrder of BST → Sorted, (Increasing order)

Inorder of BST → Sorted (Increasing)

ReverseInorder of BST → Sorted (Decreasing)

arr → 10 20 30 40 50

normal $O(n^2)$

140 120 90 50 0

resol →

left to Right $O(n^2)$

140 120 90 50 0 ← array

reverse Inorder → $O(n)$

Right to left 150 ~~140~~ ~~120~~ ~~90~~ ~~50~~ ~~0~~ ← sum

LCA of BST

Saturday, 26 June 2021 2:34 PM

LCA → (55, 65) → 62.

space → recursive space

time → O(height)

55, 65

find find

↓

find LCA

✓ node.data < {d1, d2}

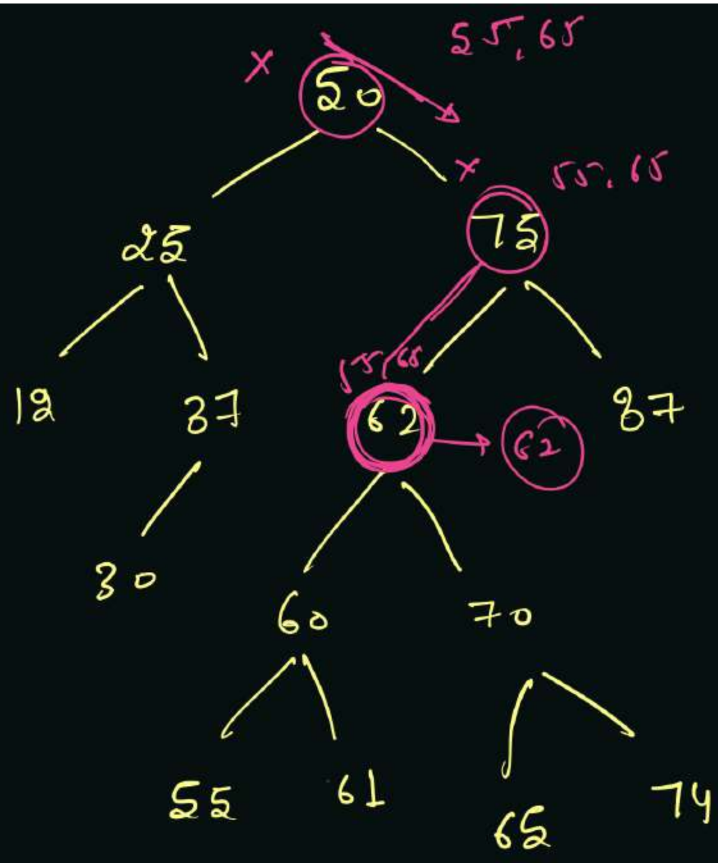
↳ Right side (LCA)

✓ {d1, d2} < node-data

↳ Left side (LCA)

✓ ↳ {d1 ≤ node.data ≤ d2 || d2 ≤ node.data ≤ d1}

else } → LCA is node.data



Print In Range

Saturday, 26 June 2021 2:35 PM

lo = 26 → 61

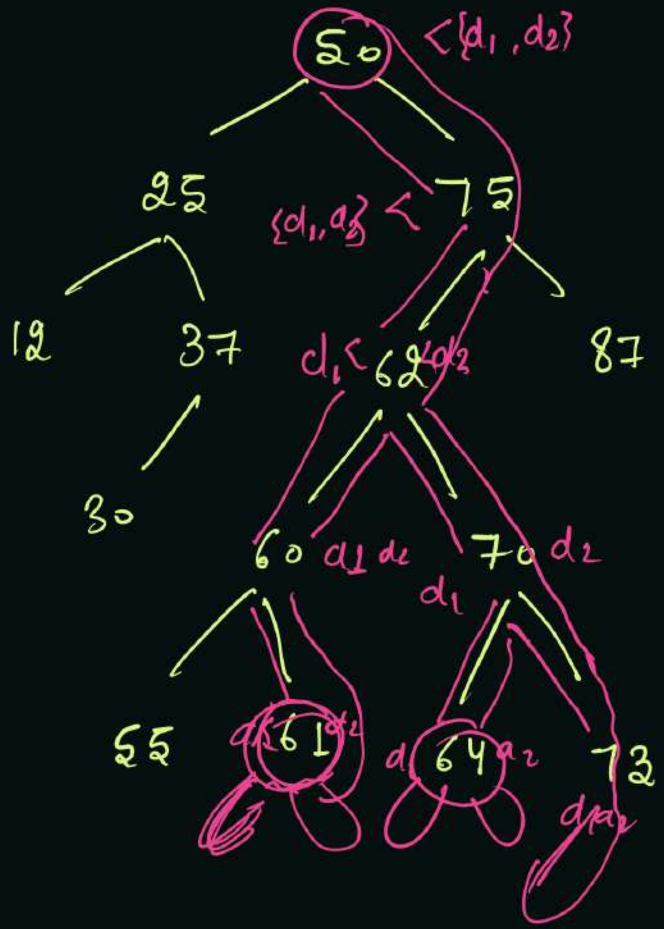
upper = 72 → print value

Time comlexity → less than O(n) but also depend on Range.

output →

[25 30 37 50 55 60 61 62 64 70]

61 62 64 70



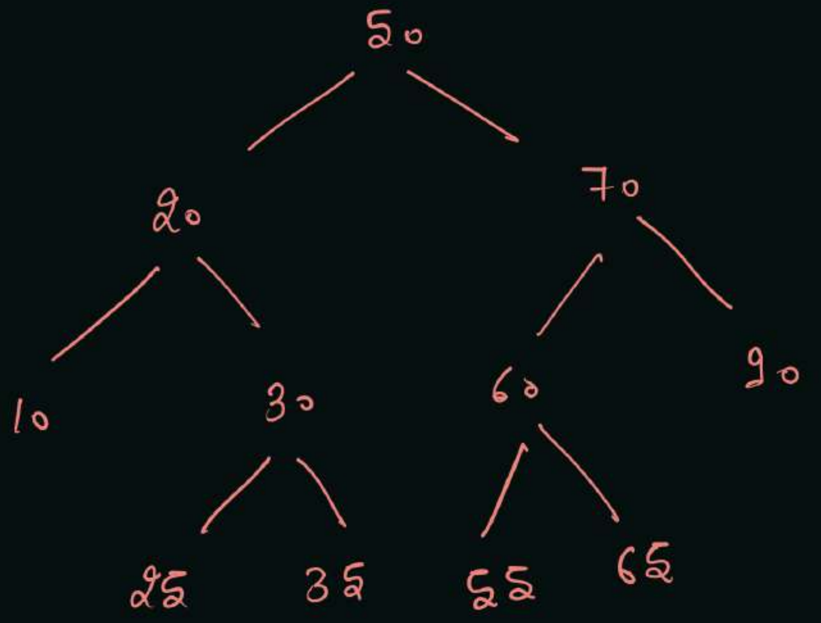
Target Sum Pair In BST

Saturday, 26 June 2021 2:35 PM

target = 100

print all pairs from which we can achieve target ⟹ 100

Ex - 10	90
20	80
30	70



	Time	space
Method 1	$O(nh)$	$O(h)$
Method 2	$O(n)$	$O(n)$
Method 3	$O(n)$	$O(h)$

→ Time is optime but required more space

→ Time is optimise as well as space optimise.

Method 1 →   Space → $O(h)$ Time : $O(nh)$

For find. original root

InOrder.

val = ~~10~~ ~~20~~ ~~25~~ ~~30~~ ~~35~~ ~~50~~ ~~55~~ ~~60~~ ~~65~~ ~~70~~ 90

val2 = $\underbrace{\text{target - val 1}}$ } if present in the tree.

$\cancel{90}$ → find.

Traversal → $O(n)$
find → $O(h)$
Total = $O(nh)$

~~80~~ ~~75~~ ~~70~~ ~~65~~ ~~50~~ ~~45~~ ~~40~~ ~~35~~ ~~30~~ 10

Space → (Recursive) $O(h)$

val1	val2.
✓ 10	90
✓ 30	70
✓ 35	65

] val1 < val2

65	35
70	30
90	10

] Repitited pair
val2 ✗ val1] Repitition

Repeated

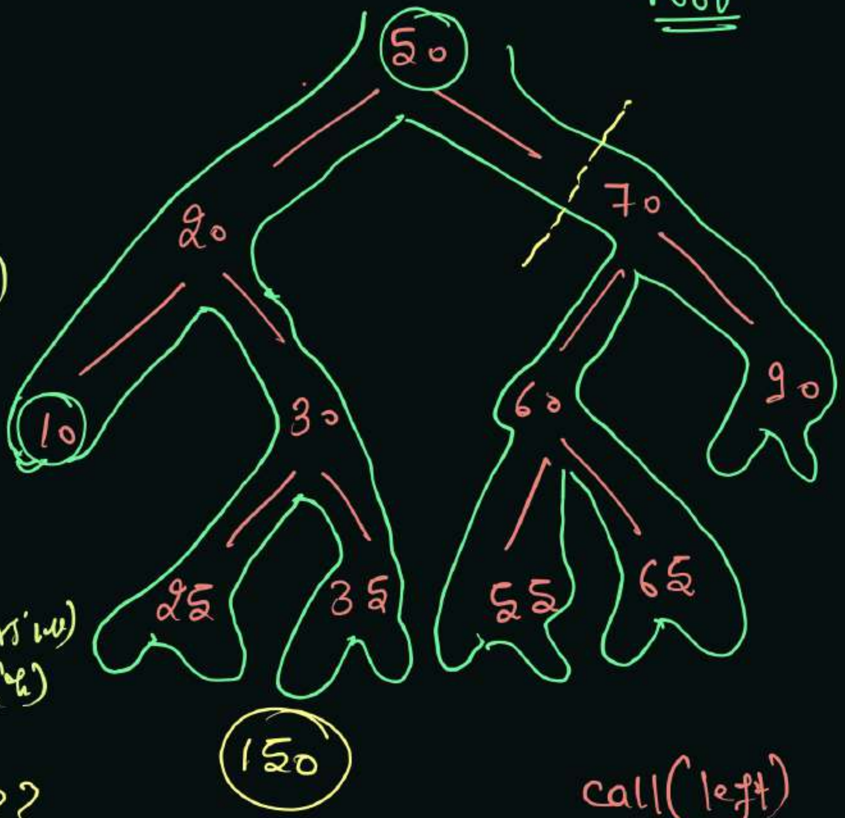
How to avoid these repeated pair ??

make a find only when val2 > val1

call(left)

work.

call(right)

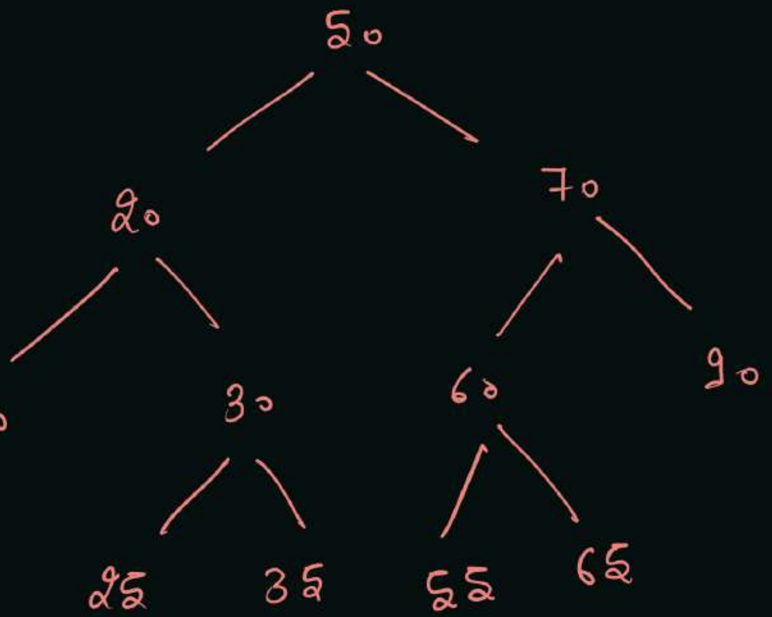


Method 2.

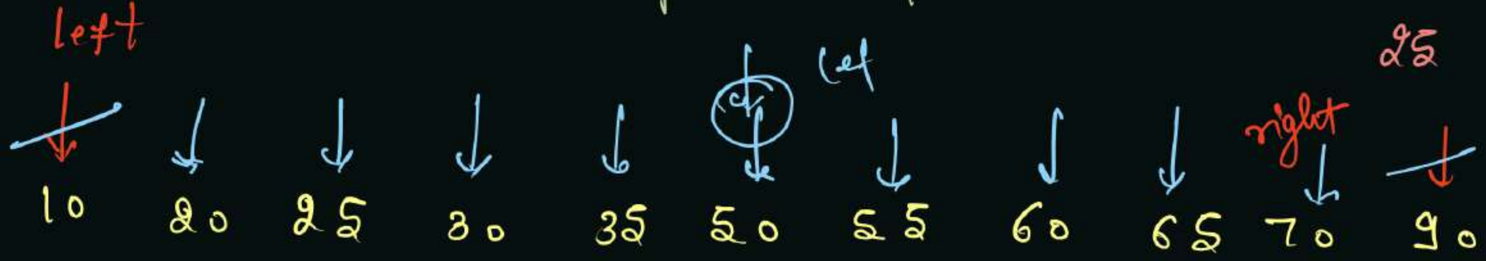
Make an array and fill it in InOrder,

↳ target sum pair.

10	90
30	70
35	65



TnS → Target sum pair



condition for iteration → <u>left < right</u>

sum = arr[left] + arr[right]

```
if(sum == target)
print( arr[left)    arr[right]);
    left++;
    right--;

else if ( sum> target)
    right --;

else
    left++;
```

Target Sum Pair 2

Saturday, 26 June 2021 2:35 PM

target = 100

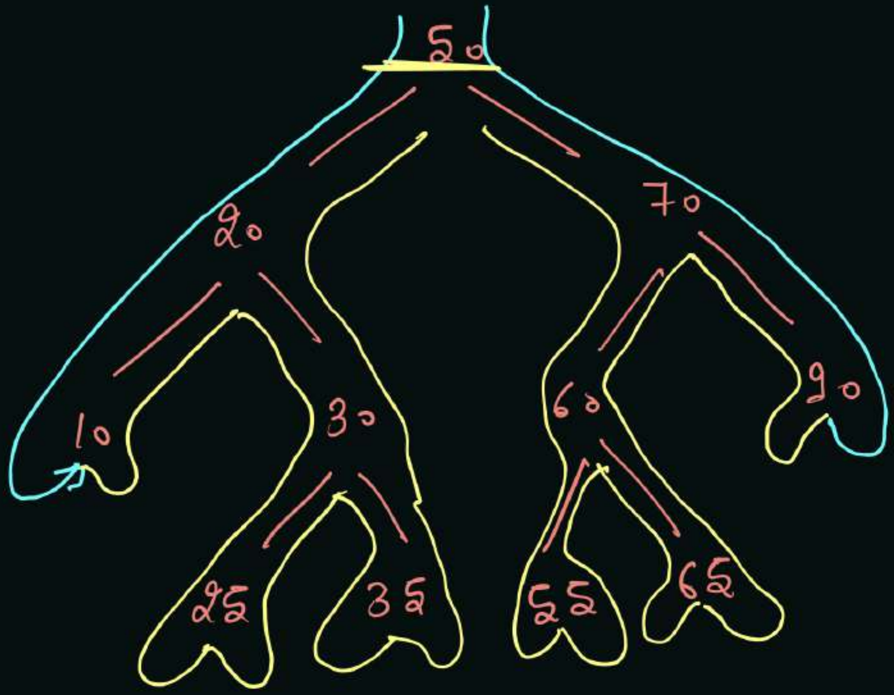
left = ~~10~~ ~~20~~ ~~25~~ ~~30~~ ~~35~~ 50

right = ~~90~~ ~~70~~ ~~65~~ ~~60~~ 55

```
Sum = left + right

if( sum == target) {
    print pair ⟶
    left++; // right--;
} elseif( Sum > target) {
    right--
} els {
    left ++; //
}
```

How we can handle left and right in recursive Tree ?

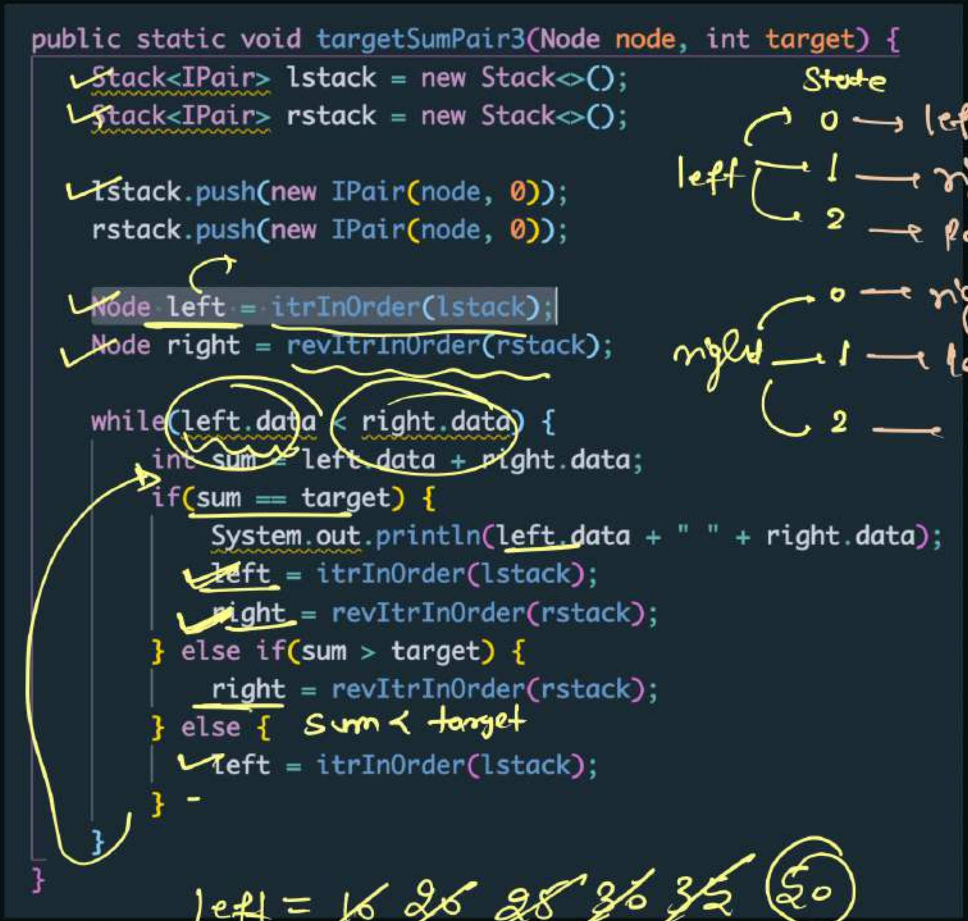


10 — 90

30 — 70

35 — 65

Iterative Traversal : ->



```
public static void targetSumPair3(Node node, int target) {
    Stack<IPair> lstack = new Stack<>();
    Stack<IPair> rstack = new Stack<>();

    lstack.push(new IPair(node, 0));
    rstack.push(new IPair(node, 0));

    Node left = itrInOrder(lstack);
    Node right = revItrInOrder(rstack);

    while(left.data < right.data) {
        int sum = left.data + right.data;
        if(sum == target) {
            System.out.println(left.data + " " + right.data);
            left = itrInOrder(lstack);
            right = revItrInOrder(rstack);
        } else if(sum > target) {
            right = revItrInOrder(rstack);
        } else {  sum < target
            left = itrInOrder(lstack);
        }
    }
}
```

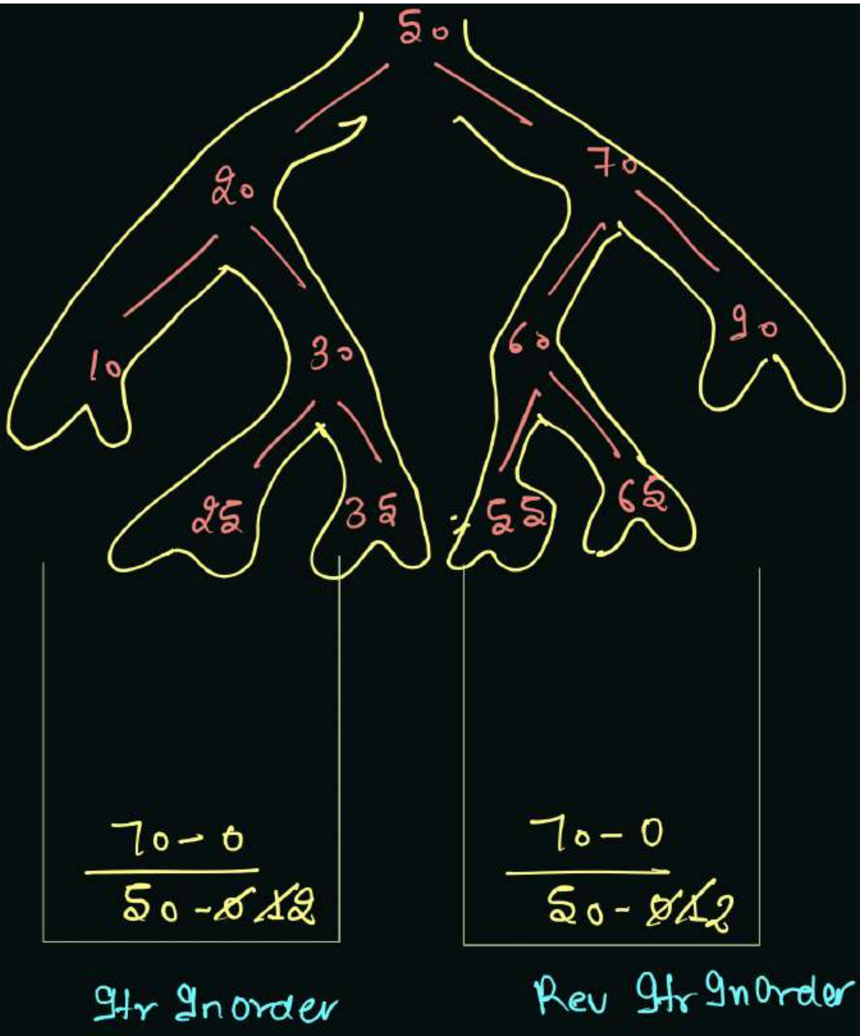
State

left
- 0 → left child
- 1 → right child
- 2 → pop.

right
- 0 → right child
- 1 → left child
- 2 → pop

left = ~~10~~ ~~20~~ ~~25~~ ~~30~~ ~~35~~ $\boxed{50}$

right = ~~90~~ ~~70~~ ~~65~~ ~~60~~ 55 $\boxed{50}$



10 — 90
30 — 70
35 — 65