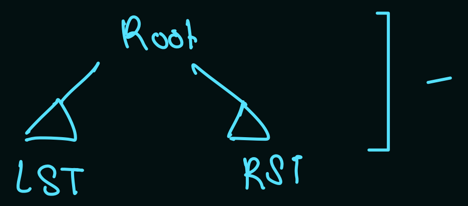
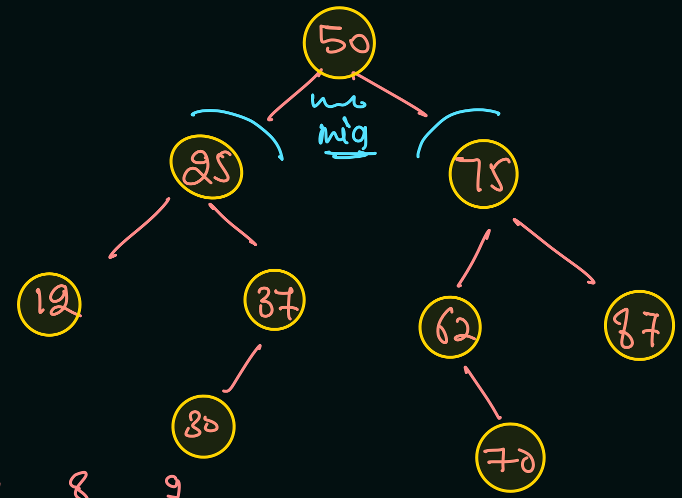
Binary Search tree (BST):



max(LST) < Root·data < min(RST)



Input → (sorted array)

0	1	2	3	4	5	6	7	8	9
10	20	30	40	50	60	70	80	90	100

Construction-

lo = 0, hi = 9

mid = $\frac{0+9}{2}$ = 4

50 Root

lo - (mid-1) ← → (mid+1) to hi

lo = 0, hi = 3
mid = $\frac{0+3}{2}$ = 1
20

lo = 0, hi = 0
mid = 0
10

lo = 0, hi = -1 | lo = 1, hi = 0

lo = 2, hi = 3
mid = $\frac{2+3}{2}$ = 2
30

lo = 2, hi = 1 | lo = 3, hi = 3

lo = 5, hi = 9
mid = $\frac{5+9}{2}$ = 7
80

lo = 5, hi = 6 | lo = 8, hi = 9

lo > hi invalid con.

BST vs. B.T.

- # value based problem is different.
- # structure based problem is same in both tree

Ex. ht = structure base
↓
BST = BT

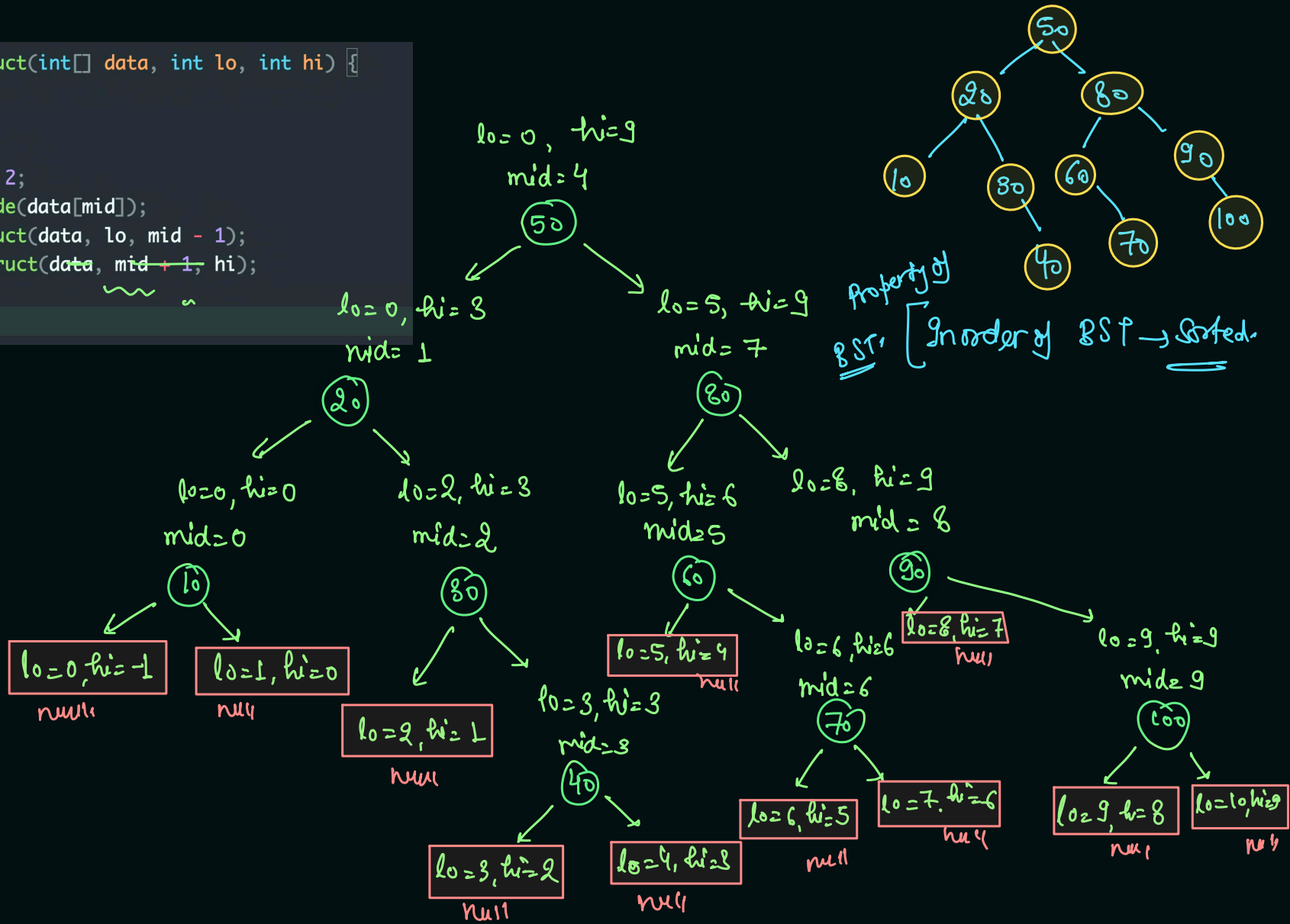
B.T(max) ≠ BST(max)

```
int[] data = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};
```

0 1 2 3 4 5 6 7 8 9

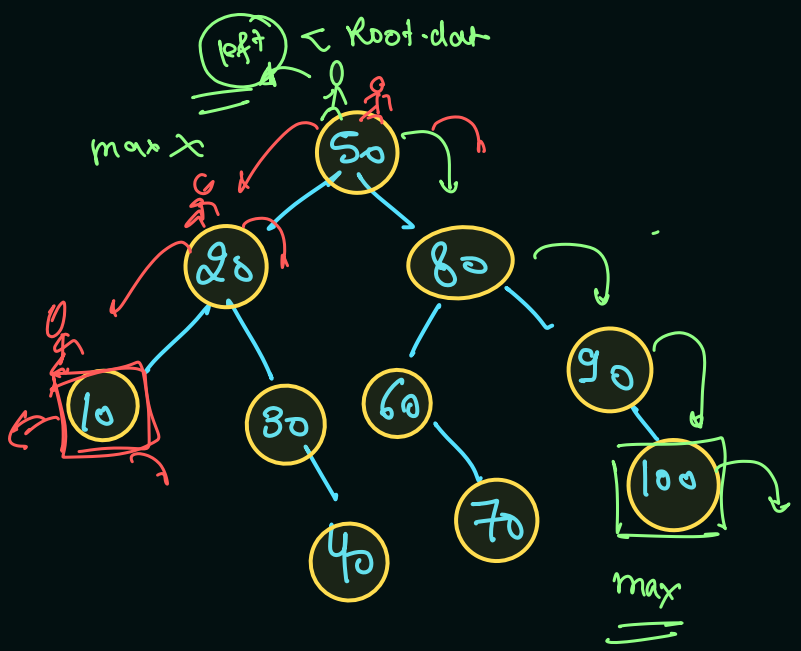
```
public static Node construct(int[] data, int lo, int hi) {
    if(lo > hi) {
        return null;
    }
    int mid = (lo + hi) / 2;
    Node midNode = new Node(data[mid]);
    midNode.left = construct(data, lo, mid - 1);
    midNode.right = construct(data, mid + 1, hi);
    return midNode;
}
```

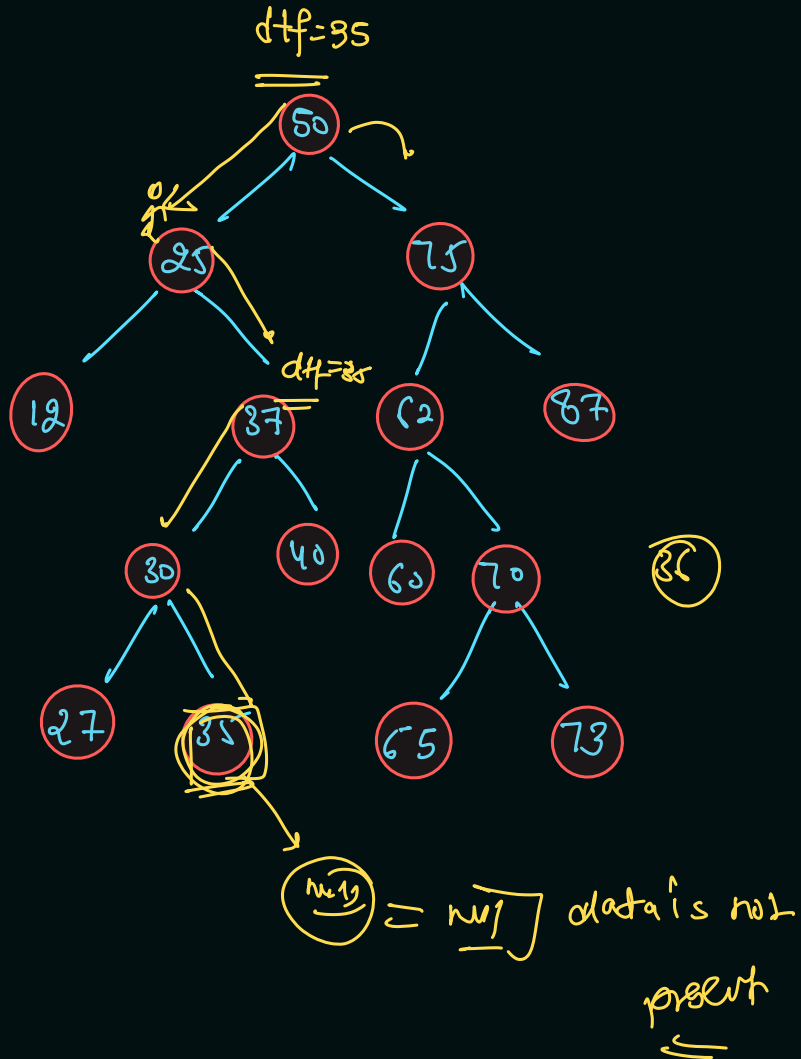
```
20 <- 50 -> 80
10 <- 20 -> 30
 . <- 10 -> .
 . <- 30 -> 40
 . <- 40 -> .
60 <- 80 -> 90
 . <- 60 -> 70
 . <- 70 -> .
 . <- 90 -> 100
 . <- 100 -> .
```



Property of BST: [Inorder of BST → Sorted.]

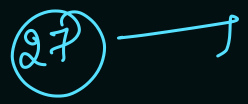
Max, Min in BST:



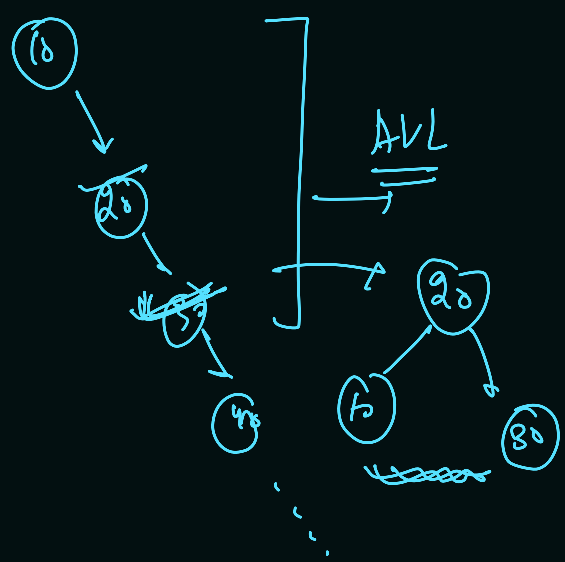


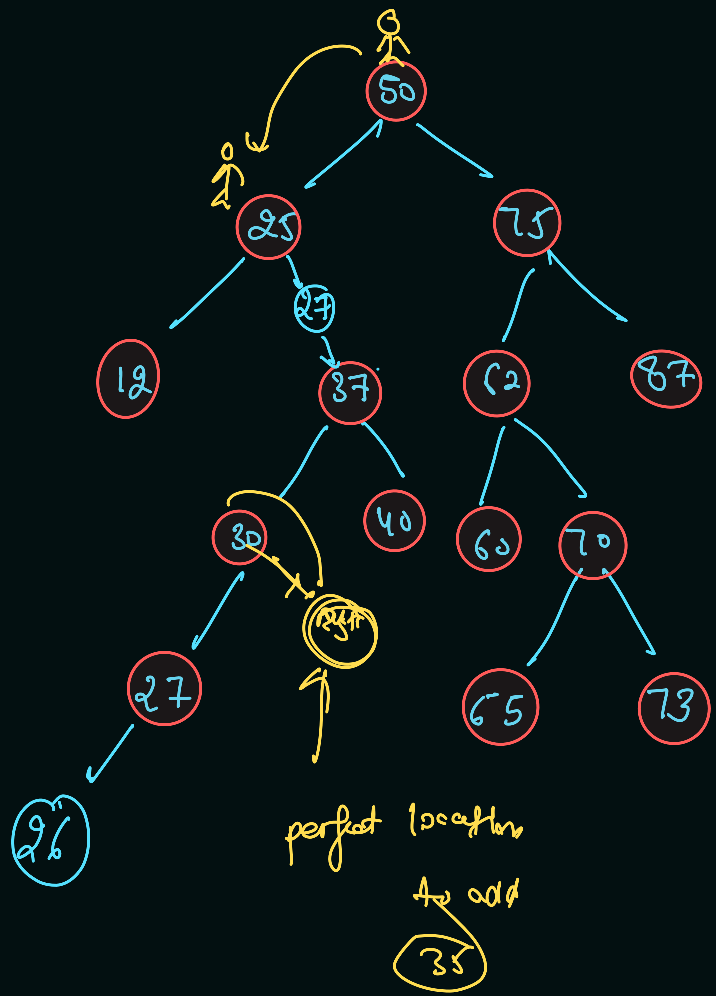
Add node in BST :

Add (35) in BST



add - 10 , ad , ad





Ideal Node.

```
int data,
Node left
Node right
int freq
```

If same data encounter then increase freq of that Node

Remove from BST :

(Return type) → Return new root of Subtree After Removal

Remove

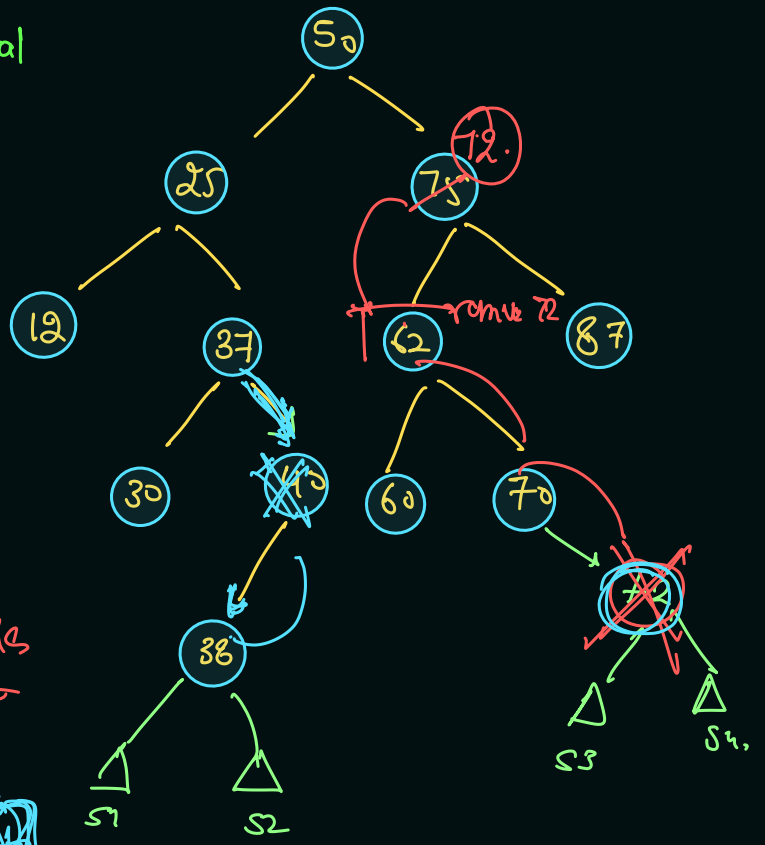
- leaf node → return → null → Ex 38
- have both child
- have only left child
- have only Right child

Done by Recursion

37.right = 38 → remove 40 → return 40.left

62.right = 72 → remove 70 → return 70.right

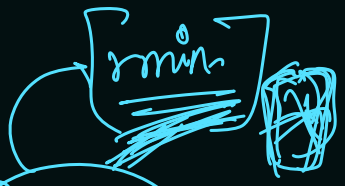
my works



have Both child-

↳ Remove · 75

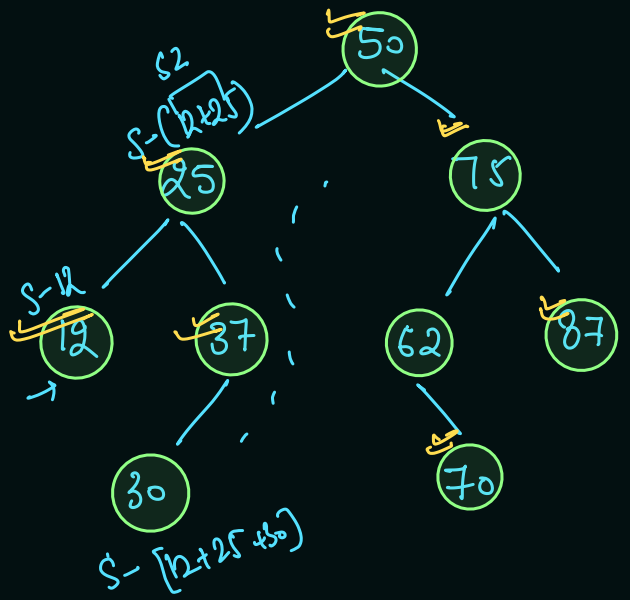
↳ lmax from 75 = lmax = 72



Set node.data = 72 leaf node OR Single child.

remove (node.left) data remove = 72

Replace with Sum of Larger value:



① Sum of nodes.

② move in Inorder maintain sum2 & subtract sum2 from sum and set the value.

Inorder of BST → sorted (increasing order)

Rev. Inorder of BST → sorted (Decreasing order).

10	20	30	40	50	60

(Sum = 210

Sum2 = 10 → 30 → 60 → 100 → 150 → 210 → 2 - Iterations Required

Sum - Sum2 = 200 → 180 → 150 → 110 → 60 → 0

10	20	30	40	50	60

reverse traversal

200 180 150 110 60 0 = sum

210 200 180 150 110 60 ← sum + arr[i]

Set data

Add

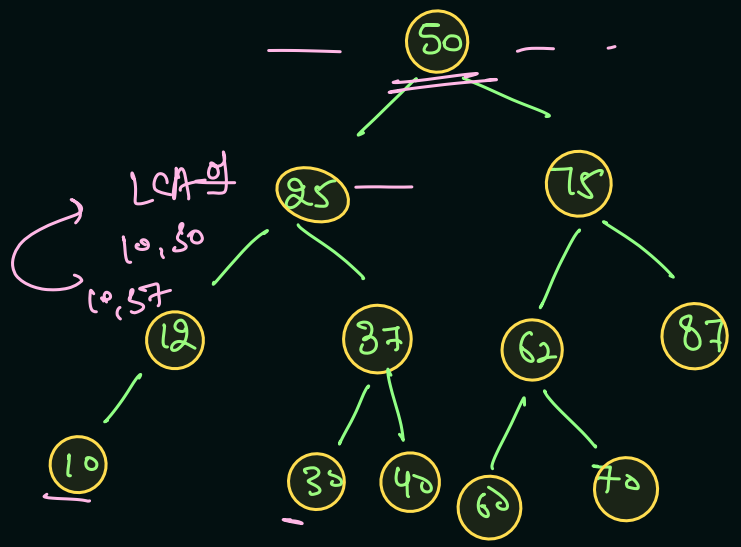
⇒ reverse Inorder traversal

right call

left call

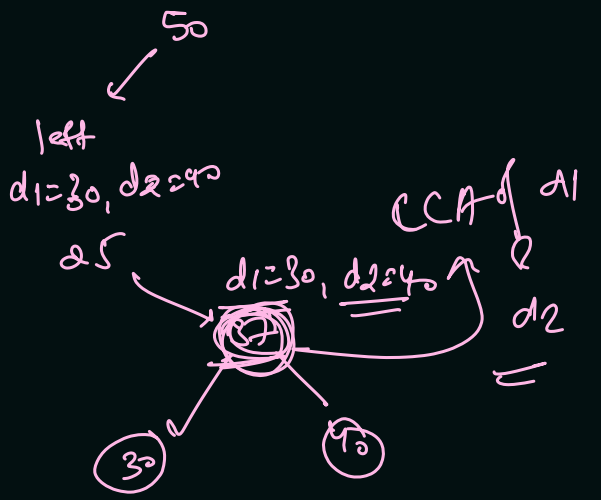
LCA of a BST :→

LCA → d1 = 10, d2 = 30



BST =?

d1 = 30, d2 = 40



10 → node to Root path

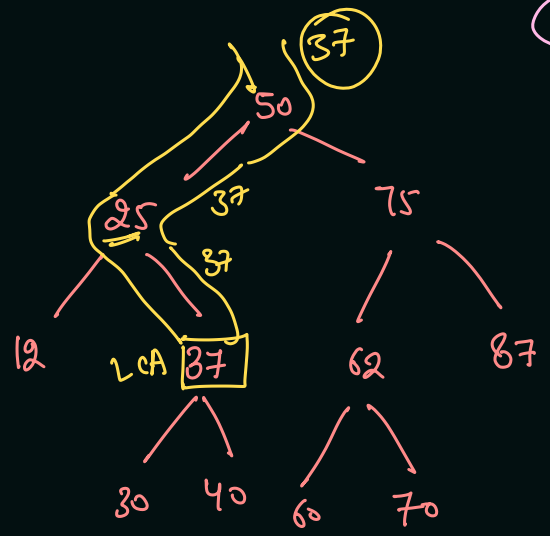
10 → 12 → 25 → 50

30 node to Root path

30 → 37 → 25 → 50

LCA

[d1=30]
[d2=40]




```java
public static int lca(Node node, int d1, int d2) {
    if(node == null) return -1;

    if(d1 < node.data && d2 < node.data) {
        // lca exist in left
        return lca(node.left, d1, d2);
    } else if(d1 > node.data && d2 > node.data) {
        // lca exist in right
        return lca(node.right, d1, d2);
    } else {
        // node is LCA
        return node.data;
    }
}
```

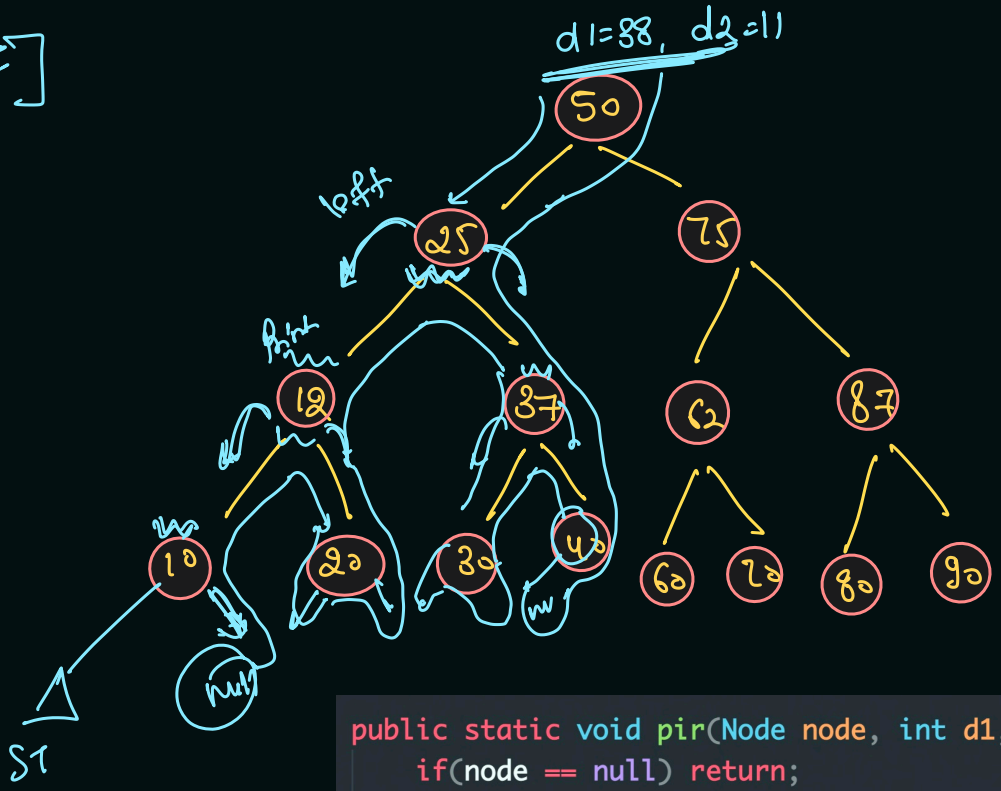
Print in Range:

12 20 25 30 37

① lo: 11, hi: 38

② lo: 61, hi: 85

③ lo: 15, hi: 85




```java
public static void pir(Node node, int d1, int d2) {
    if(node == null) return;

    if(d1 < node.data && d2 < node.data) {
        // print segment is appears in left
        pir(node.left, d1, d2);
    } else if(d1 > node.data && d2 > node.data) {
        // print segment is appears in right side
        pir(node.right, d1, d2);
    } else {
        // move toward left, print yourself then move toward right
        pir(node.left, d1, d2);
        System.out.println(node.data);
        pir(node.right, d1, d2);
    }
}
```

Target Sum pair in BST:-

helper function → find , target = 100 pair

visit of nodes → (n) travel

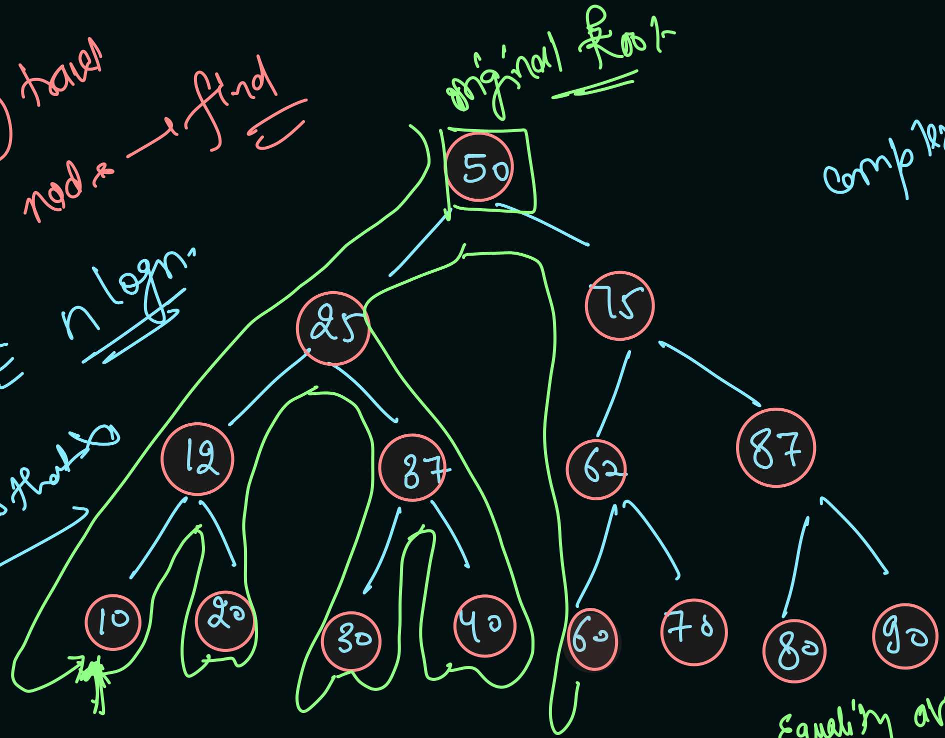
for Every single node → find

n x log n ≡ n logn

↳ O(n) we can do that

complexity of find } → height of tree in BST

↳ log(n)



10 90
20 80
25 75
30, 70
40, 60

[60, 40] ✗] Do not Repeat

Equality avoid

✓ val1 = 10 = 12, 20, 25, 30, 37, 40, 50, 60

✓ val2 = target - val1 = 100 - 10 = 90 → check if value is present in tree

original Root [val2 = 88 80, 75, 70, 63, 60, 50, 40

[val1 < val2] find print

Target Sum pair

	time complexity		space complexity	
Method1	Inorder + find	$n \log n$	time is not optimised	$\log n$	space optimised
Method2	fill inorder in ArrayList then solve with two pointers	n	time optimised	n	space is not optimised
Method3	Inorder + Reverse inorder using Iterative method of traversal	n	optimise time	$\log n$	optimised space ↳ stack space

Self try