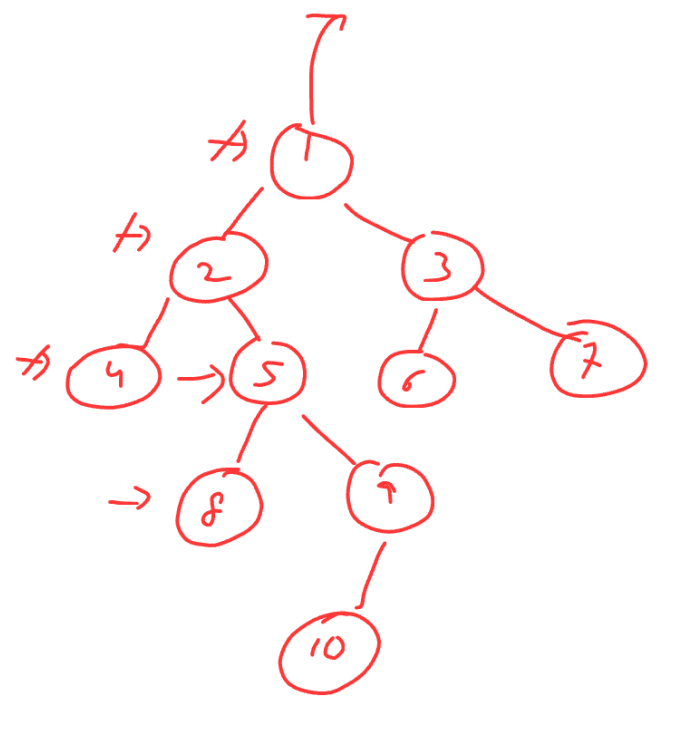


currPath: [1, 2, ~~4~~ 5, 8]

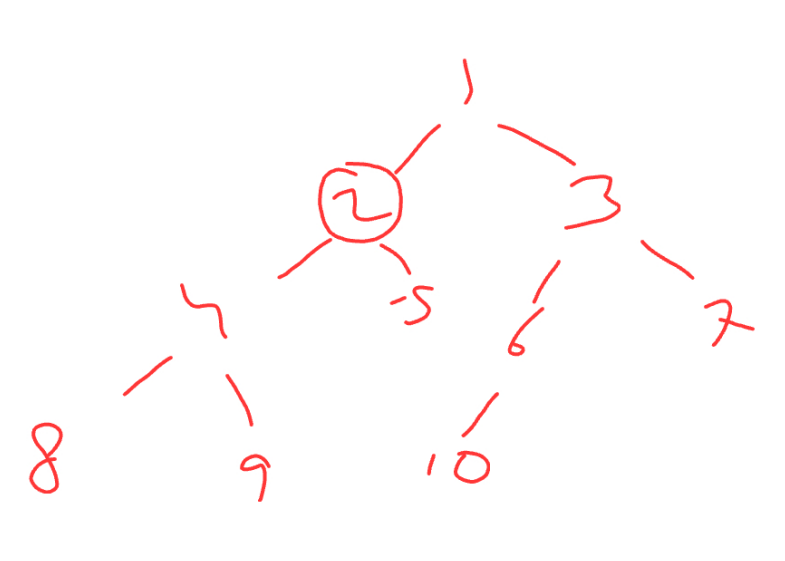
n1: 8 : 5, 2, 1

n2: 10 : 9, 5, 2, 1

path n1 = [1, 2, 5, 8]

path n2 = [1, 2, 5, 9, 10]

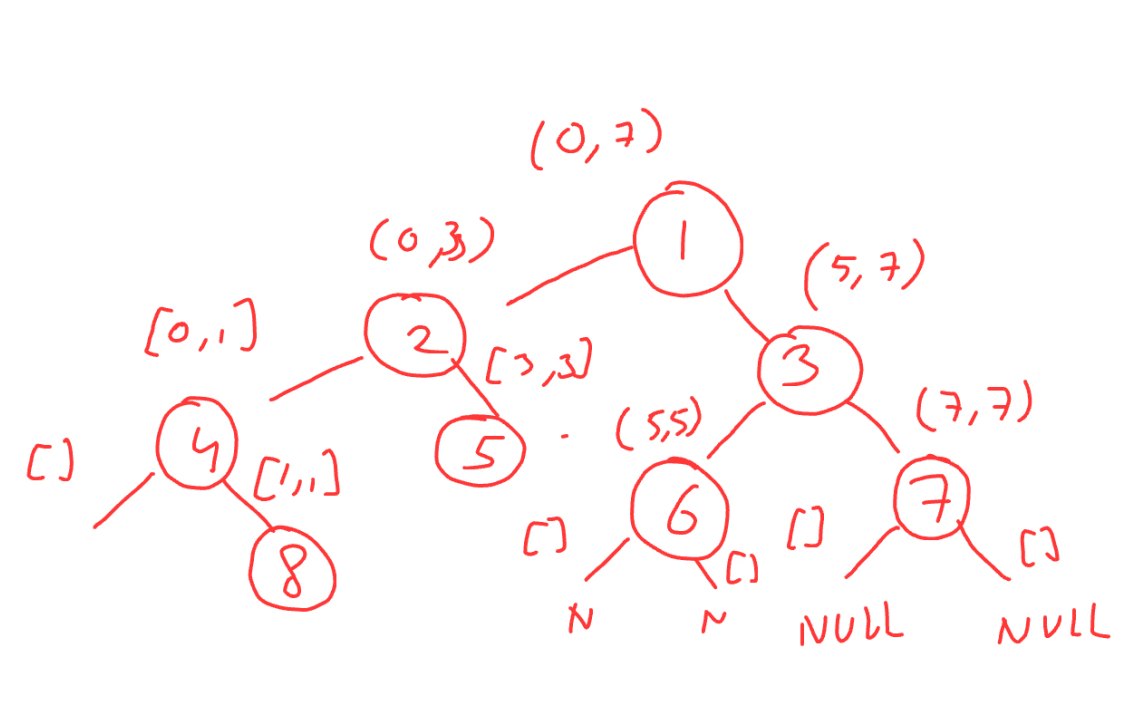
res = ~~x~~ ~~8~~ 5

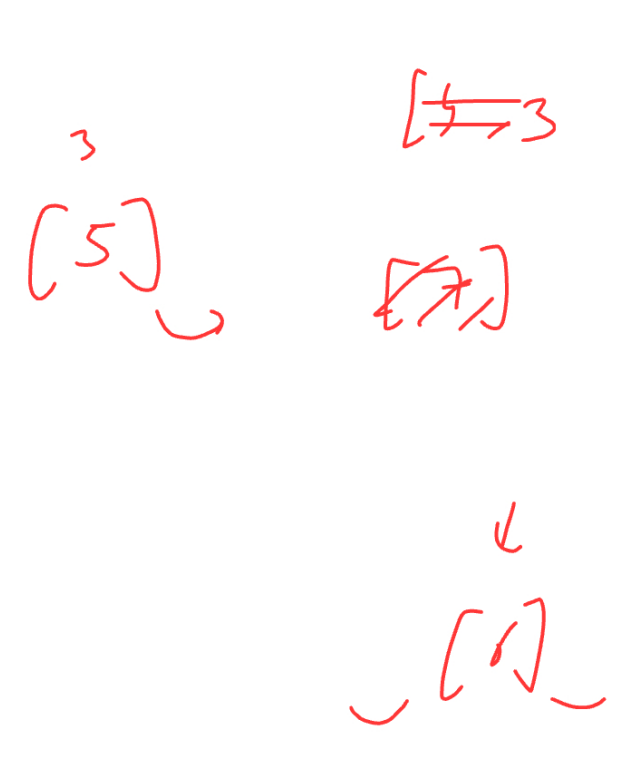


Only a unique B. T. can be constructed using a pair of inorder & preorder / inorder & postorder

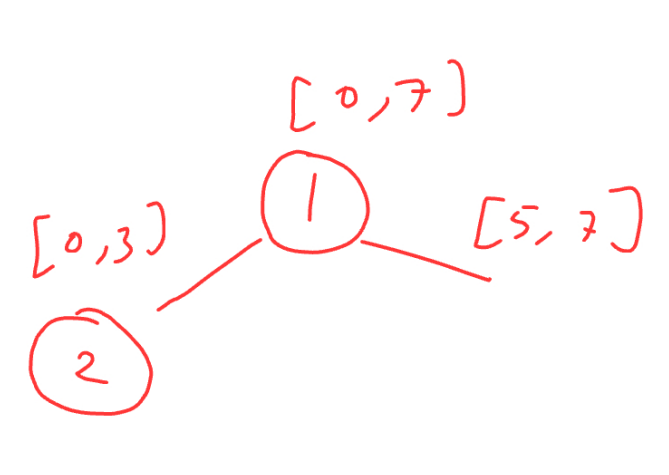
---

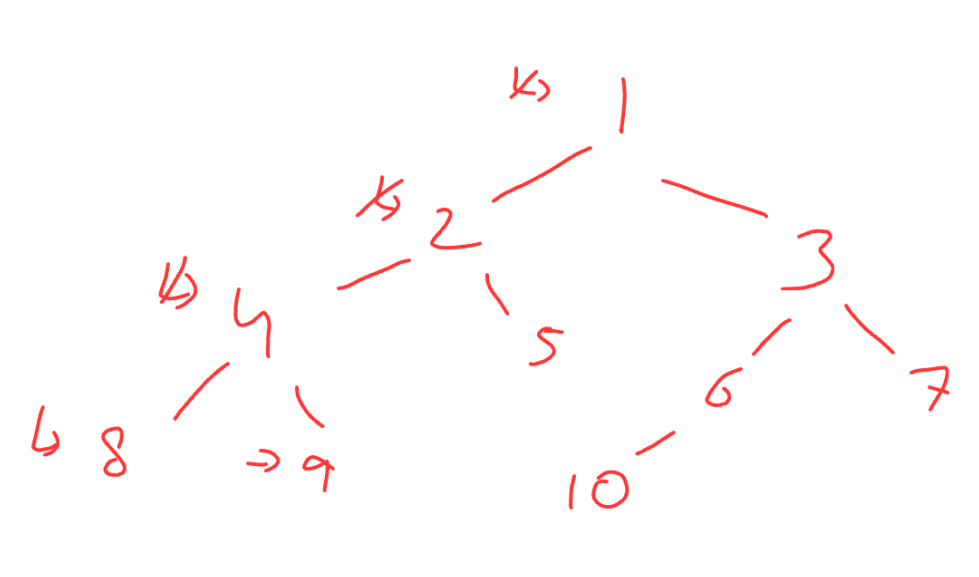
	0	1	2	3	4	5	6	7	
In [] :	4	8	2	5	1	6	3	7	→ LSR
Post [] :	8	4	5	2	6	7	3	1	→ LRS

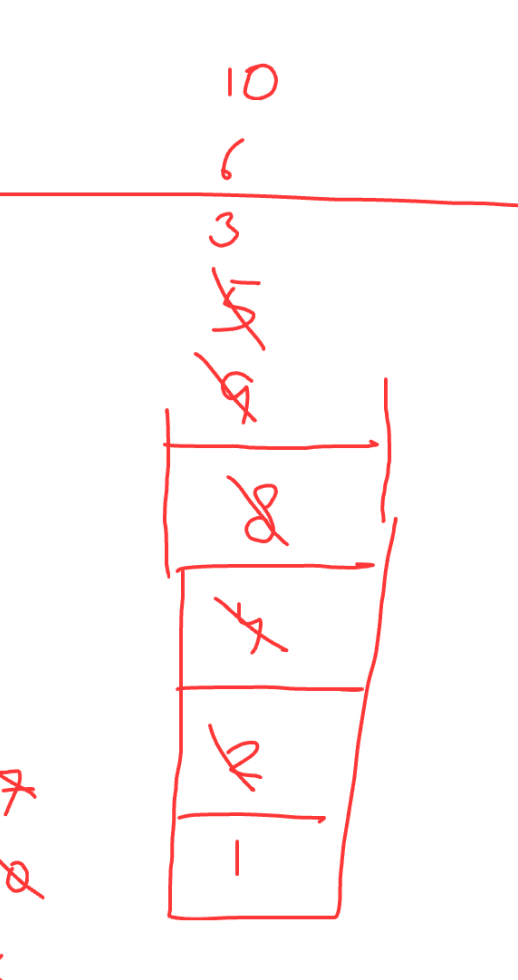

(0,7)
(0,3)
(5,7)
[0,1]
[3,3]
(5,5)
(7,7)
[]
[1,1]
[]
[]
[]
[]
N
N
NULL
NULL



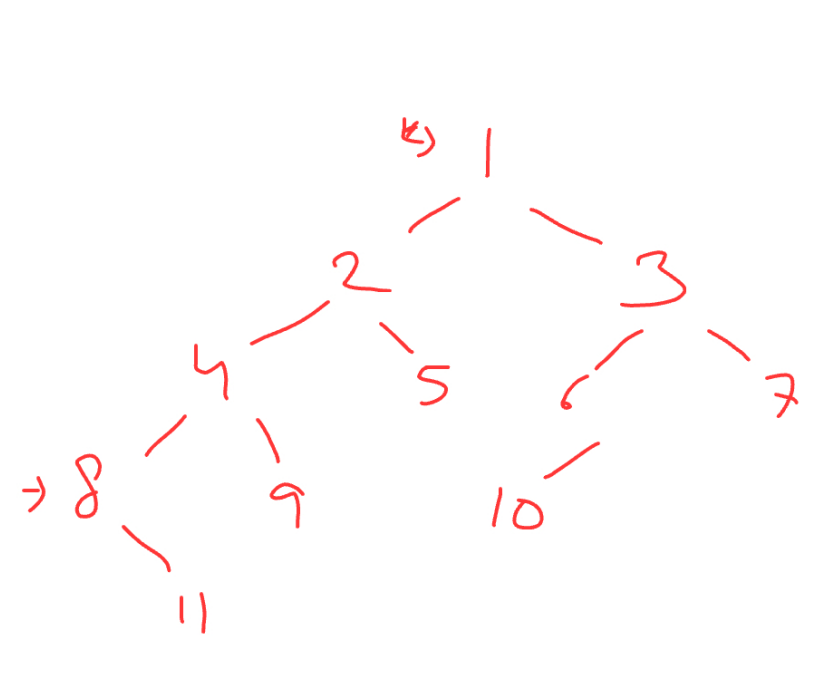
	0	1	2	3	4	5	6	7	
in [] :	4	8	2	5	1	6	3	7	→ LSR
pre [] :	1	2	4	8	5	3	6	7	→ SLR

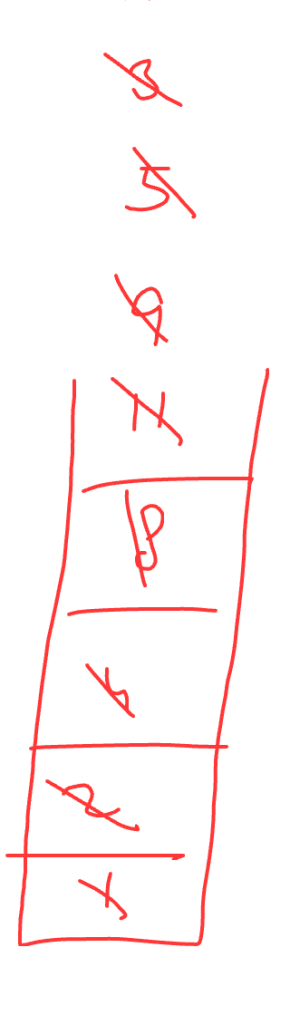





10
6
3
5
9
8
4
2
1

[8, ~~4, 9, 2, 5~~, 1




7
10
4
3
5
9
4
8
4
2
1

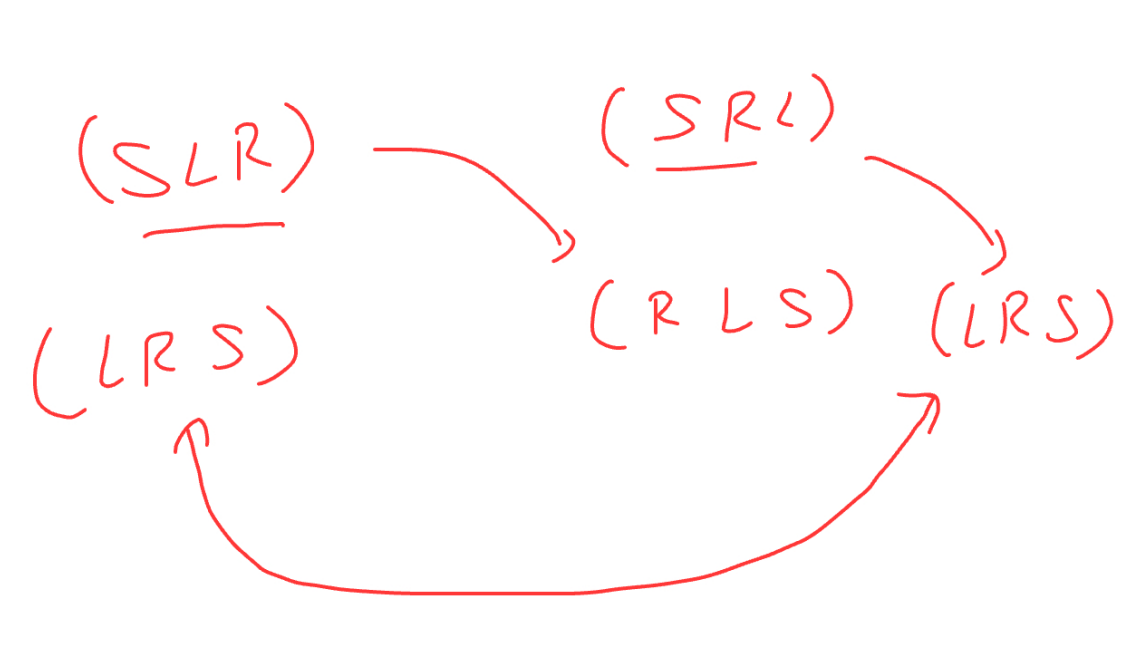
[8, 11, 4, 9, 2, 5, 1, 10, 6, 3, 7]

TC: O(n)

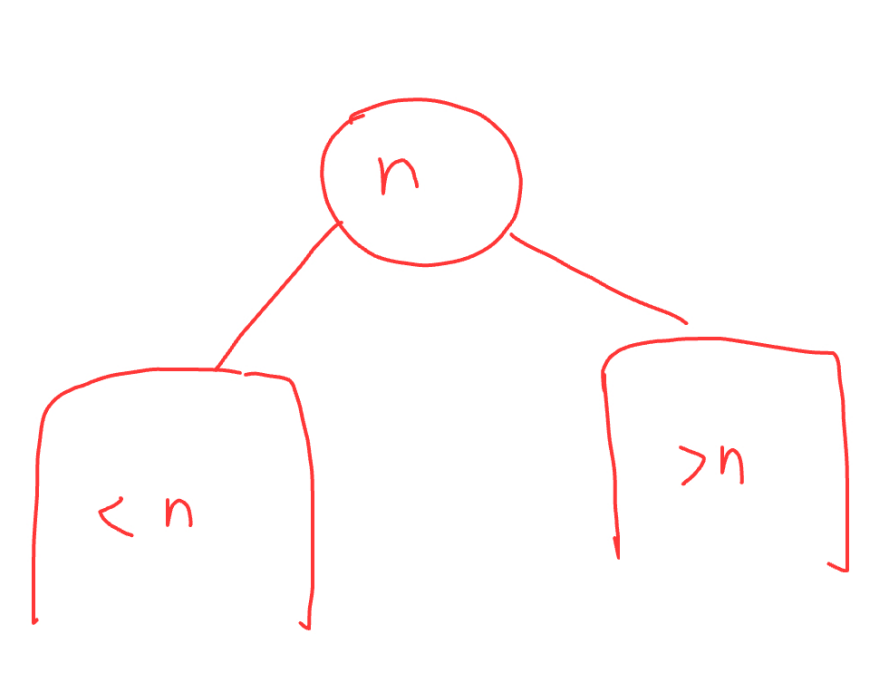
AS: O(H)

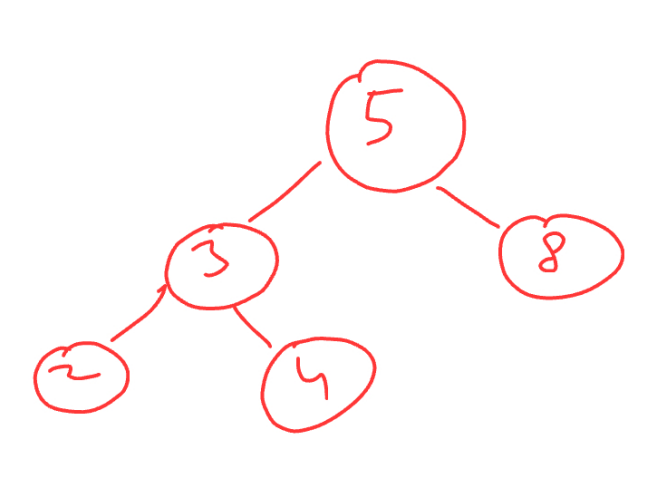
TODO: Iterative Preorder.

Postorder

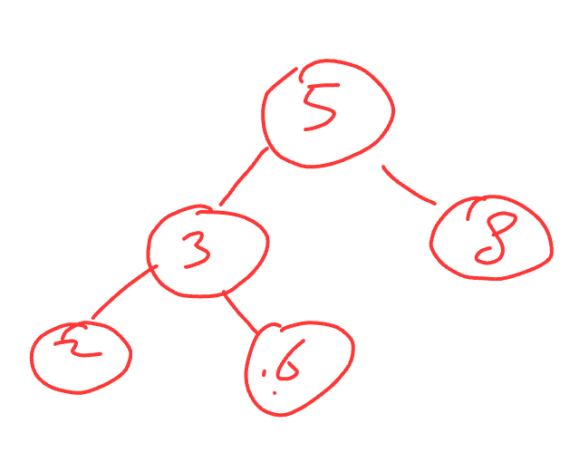

(SLR)
(SRL)
(RLS)
(LRS)
(LRS)

BST

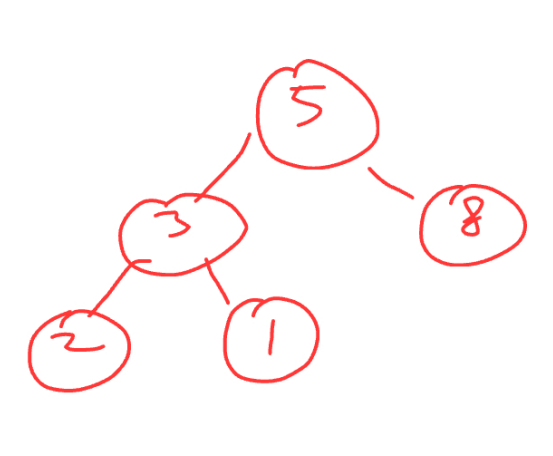




BST



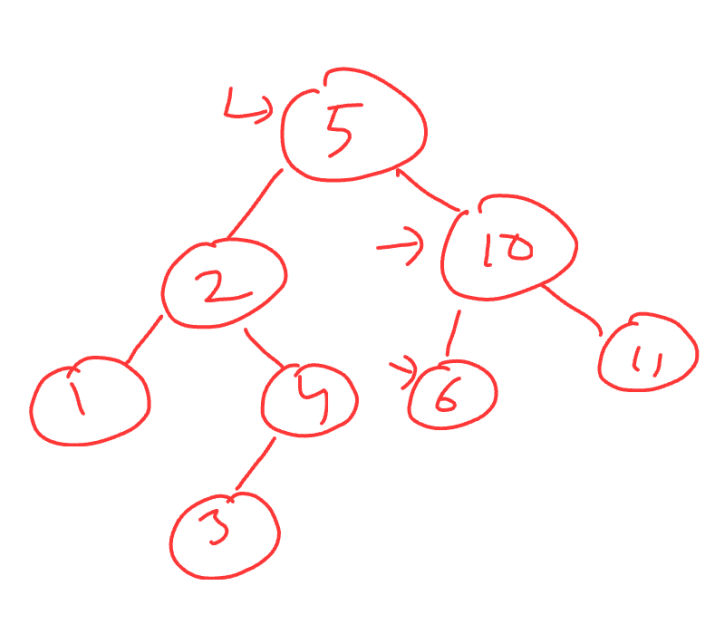
Not a BST.



Not a BST.

BST provides a storing mechanism which can be used for efficient sorting / searching operation.

Search.



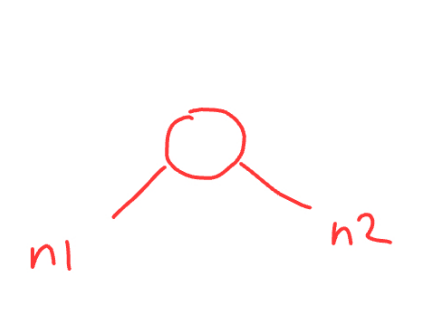
target = 6

TC: O(H)

AS: O(H) rec.

O(1) iterative.

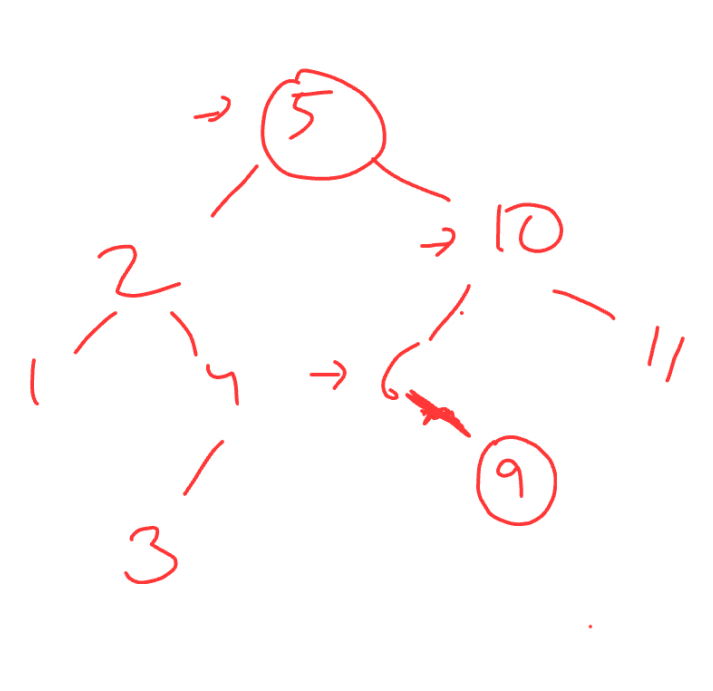
Ordered sets. → Self-Balancing BSTs.

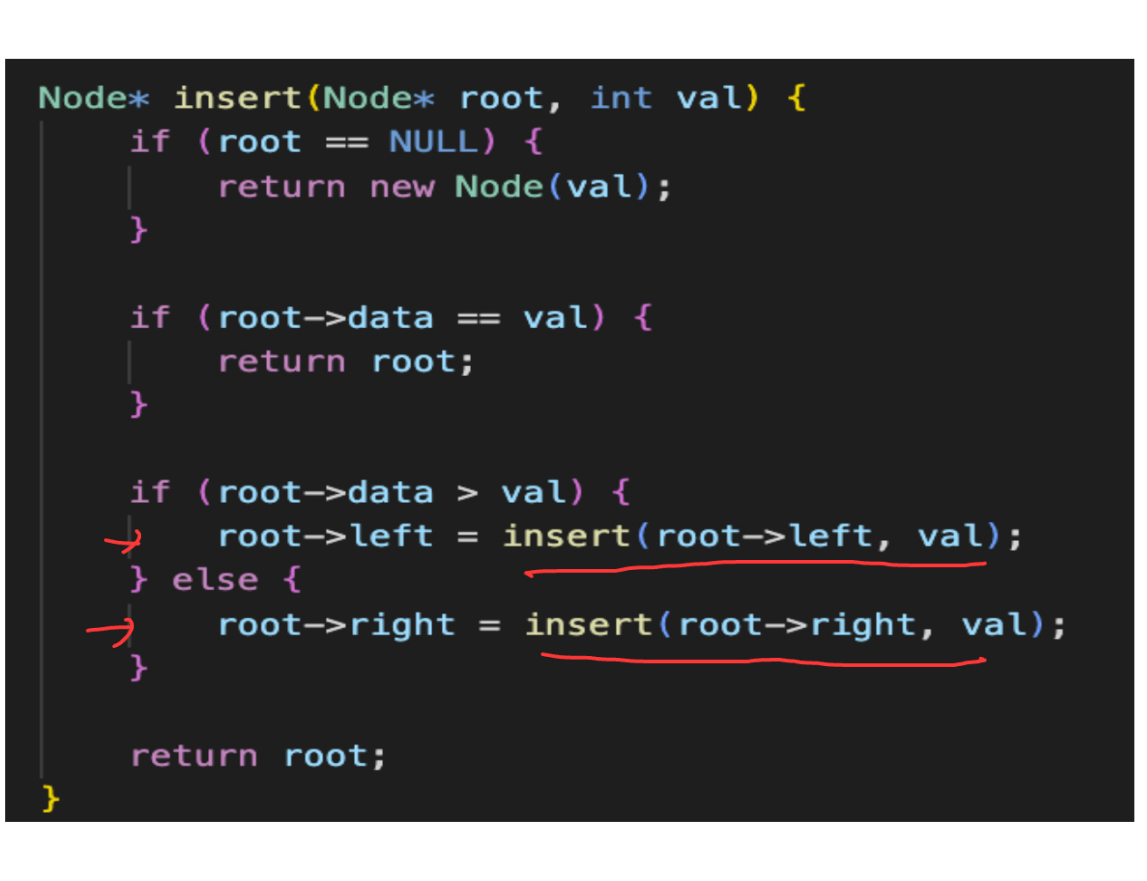


→ Balanced BT: abs (h1 - h2) ≤ 1

→ Search: O(log n)

val = 9





```
Node* insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }

    if (root->data == val) {
        return root;
    }

    if (root->data > val) {
        root->left = insert(root->left, val);
    } else {
        root->right = insert(root->right, val);
    }

    return root;
}
```

Inorder: [1, 2, 3, 4, 5, 6, 9, 10, 11]

(LSR)

The Inorder traversal of a BST is always sorted.