Trees 2 : BST

Good Morning

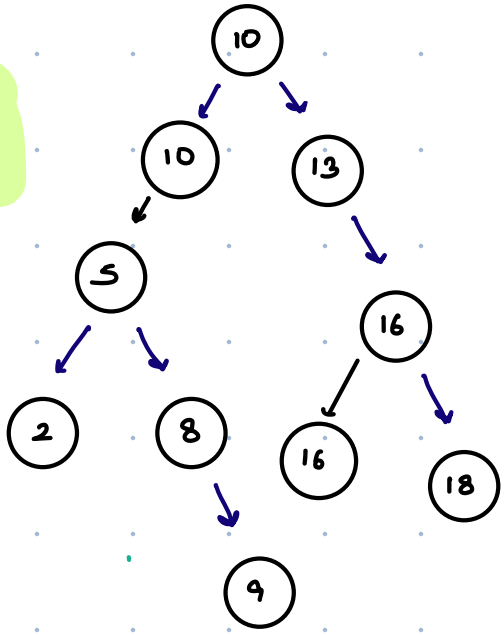
Today's content

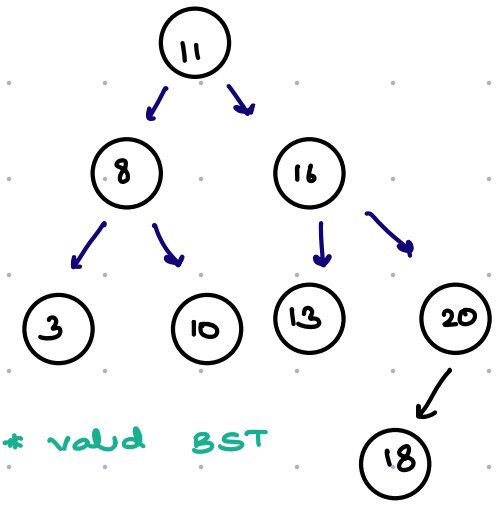
- → BST Basics + Searching
- → Insertion
- → Deletion in BST
- → Construct BST from sorted arr
- → Check if BT is a BST

A binary tree is BST

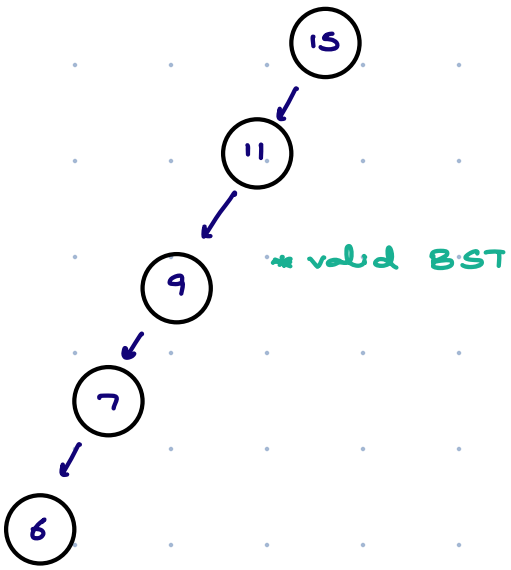
All elements in LST $\leq$ node $<$ All elements in RST

for all nodes

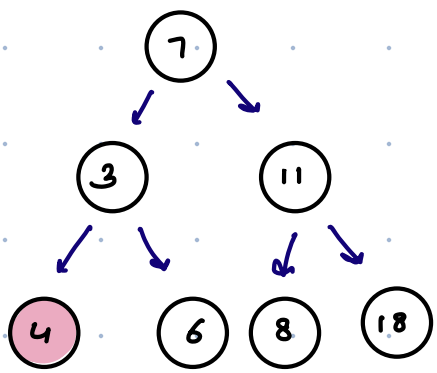




* valid BST



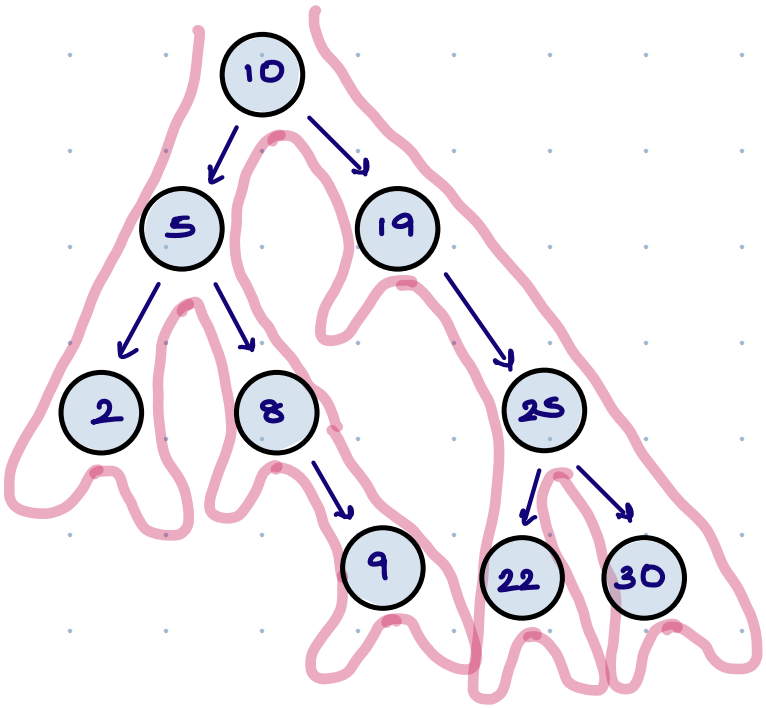
* valid BST



* not a valid BST

* valid BST

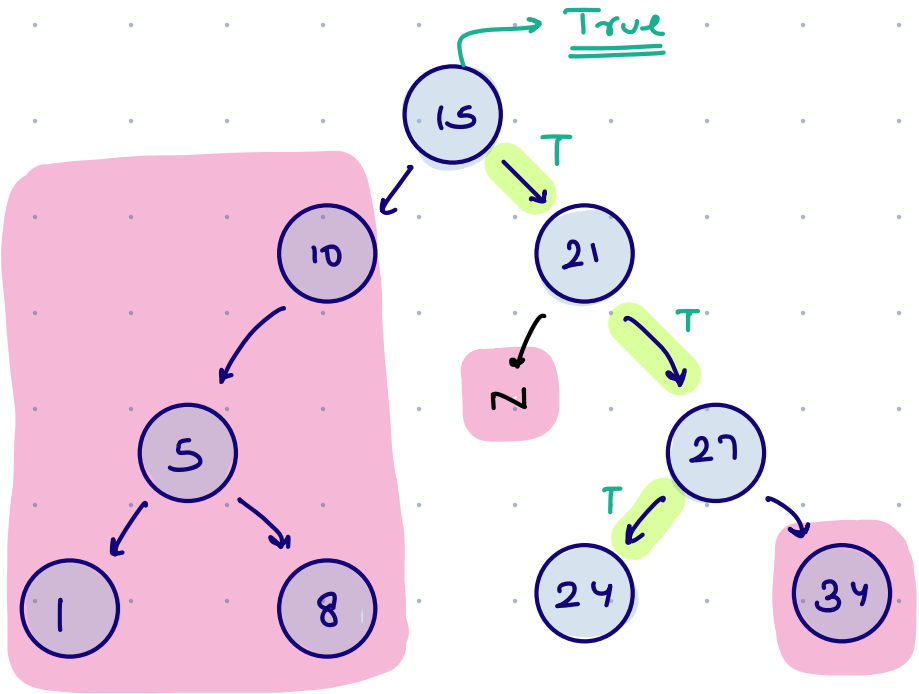
* Inorder of BST



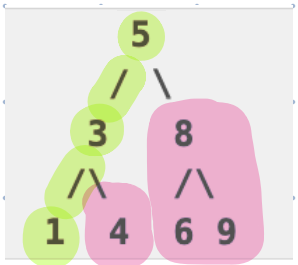
Inorder = 2 5 8 9 10 19 22 25 30

> Note ⇒ Inorder of BST is always going to be in increasing sorted order

Search in BST



K = 24

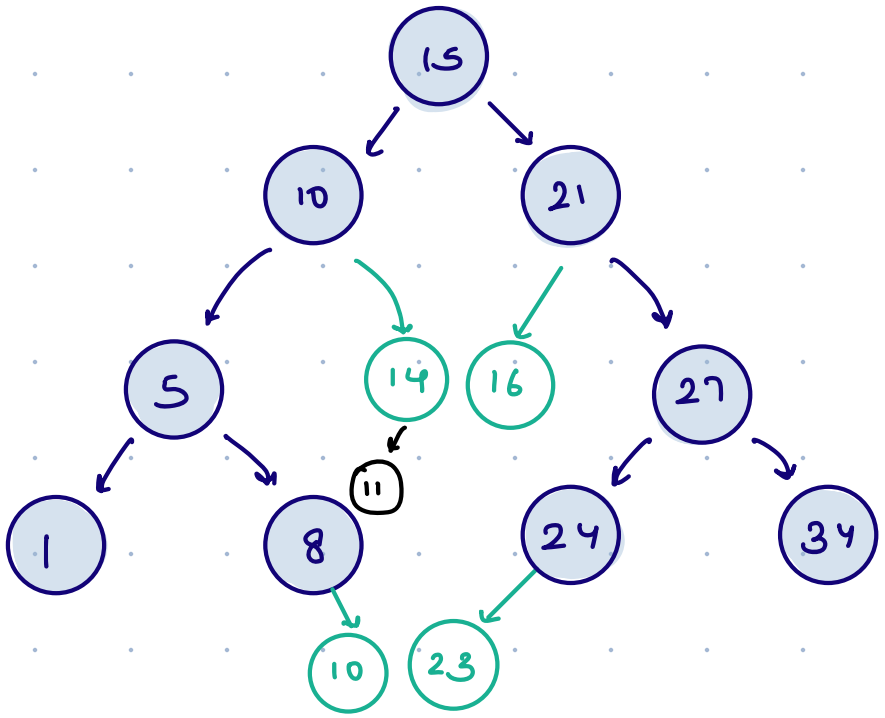


No. of nodes we need to visit to find 1 → Ans = 3

```
boolean search ( Node root , int k)
    if (root == Null) return false;
    if (root.val == k) return true;
    else if (root.val > k) return search(root.left, k);
    else return search(root.right, k);
}
```

TC : O(ht)
SC : O(ht)

* Insertion in BST



k = 14

k = 16

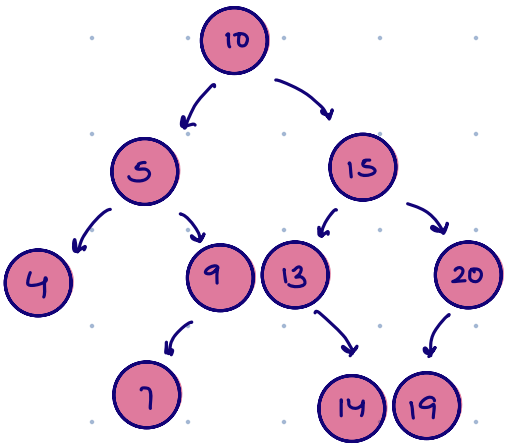
k = 23

k = 10

k = 11

```
Node insert ( Node root, int k)
{
    if (root == Null)
    {
        Node nn = new Node(k);
        return nn;
    }
    if (root.val < k)
    {
        root.right = insert(root.right, k);
    }
    else
    {
        root.left = insert(root.left, k);
    }
    return root;
}
```

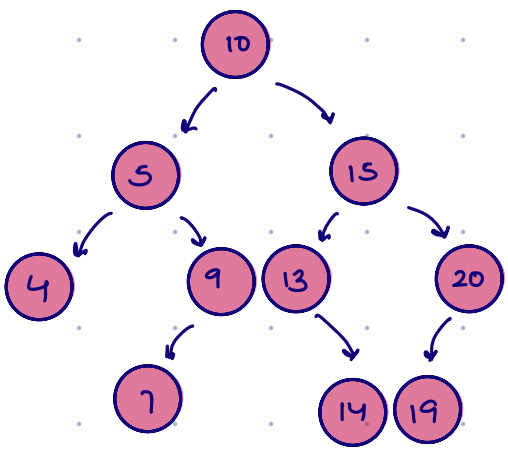
Min in BST




```
int mini (Node root)
{
    Node curr = root;
    while (curr.left != Null)
    {
        curr = curr.left;
    }
    return curr.val;
}
```

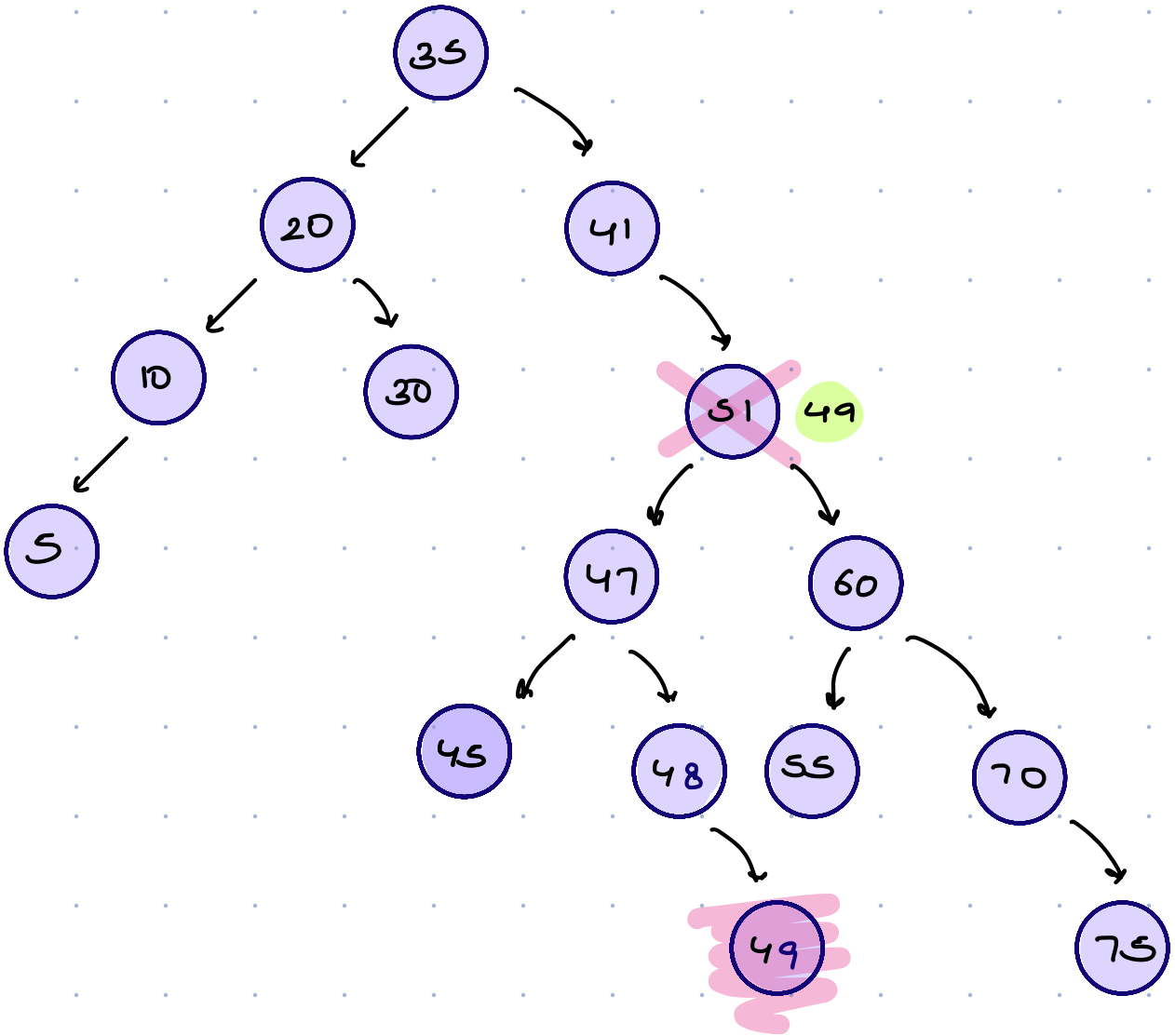
TC : O(ht)
SC : O(1)

Max in BST




```
int maxi (Node root)
{
    Node curr = root;
    while (curr.right != Null)
    {
        curr = curr.right;
    }
    return curr.val;
}
```

Delete a node in BST



K = 5
K = 45
} Delete a leaf Node

K = 10
K = 41
} Delete a node with single child

K = 20
K = 51
} Delete a node with two children

```
Node deleteBST ( Node root, int k )
{
    if (root == Null) return root;

    if ( k < root.val)
    {
        root.left = deleteBST (root.left, k);
    }
    else if ( k > root.val)
    {
        root.right = deleteBST (root.right, k);
    }
    else
    {
        if (root.left == Null && root.right == Null)
        {
            return Null;                    // delete a leaf node
        }
        else if (root.left != Null && root.right == Null)
        {
            return root.left;               // delete a single child node
        }
        else if (root.left == Null && root.right != Null)
        {
            return root.right;              // delete a node with single child
        }
        else {
            int max = maxi (root.left);
            root.val = max;
            root.left = deleteBST (root.left, max)
        }
    }
    return root;
}
```

Balanced Binary Search Tree

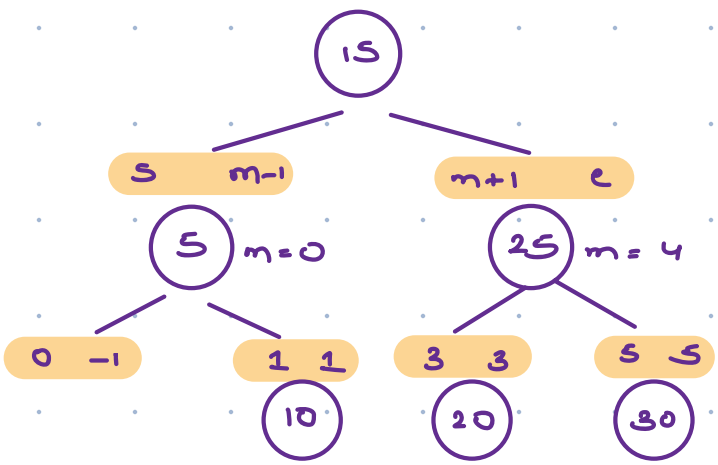
↳ $|\text{left subtree height} - \text{right subtree height}| \leq 1$

purpose for balancing → searching, insertion, deletion

TC : O(log N)

* Construct a balanced BST using a sorted integer array of distinct elements.

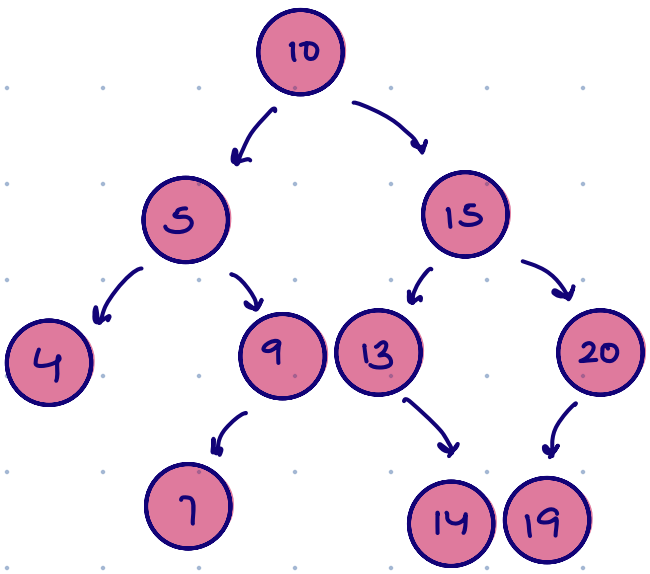
```
        s              m              e
A[] = { 5   10   15   20   25   30 }
        0    1    2    3    4    5
```



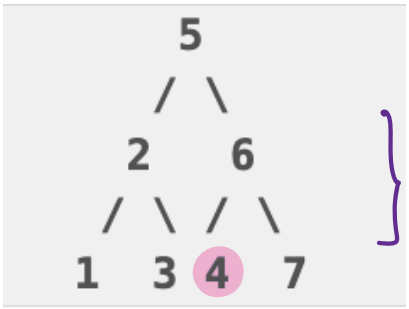
$\frac{3+5}{2} = 4$

```
Node constructBST(int[] A, int s, int e){
    if(s > e) return Null;
    int m = (s+e)/2;
    Node root = new Node(A[m]);
    root.left = constructBST(A, s, m-1);
    root.right = constructBST(A, m+1, e);
    return root;
}
```

* Given a Binary tree, check if it is BST or not?



∀nodes, { all data in LST ≤ x ; all data in RST > x }



Not a BST

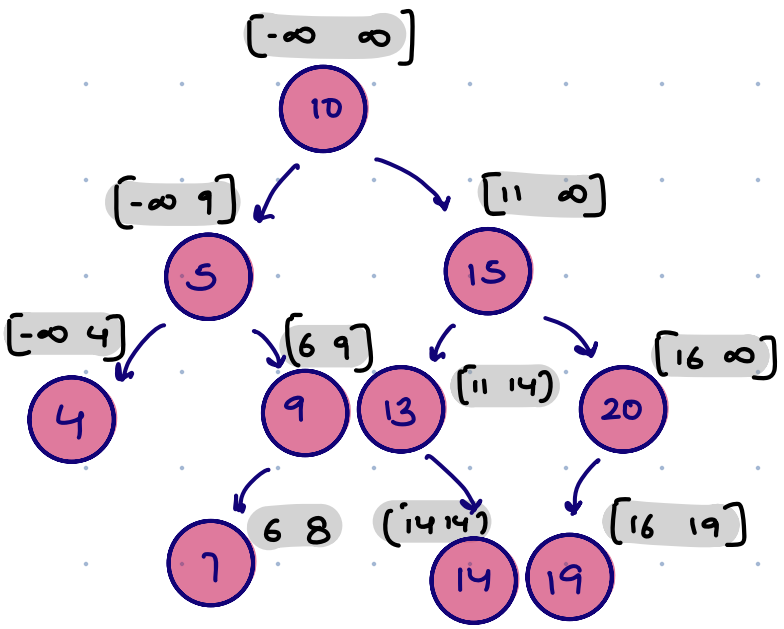
Inorder = 1 2 3 5 4 6 7

Brute force → Store inorder of tree inside AL & then check if it is in increasing order.

TC : O(N)

SC : O(N)

* **Optimal Solution**



left

s → remains same

e → root.val −1

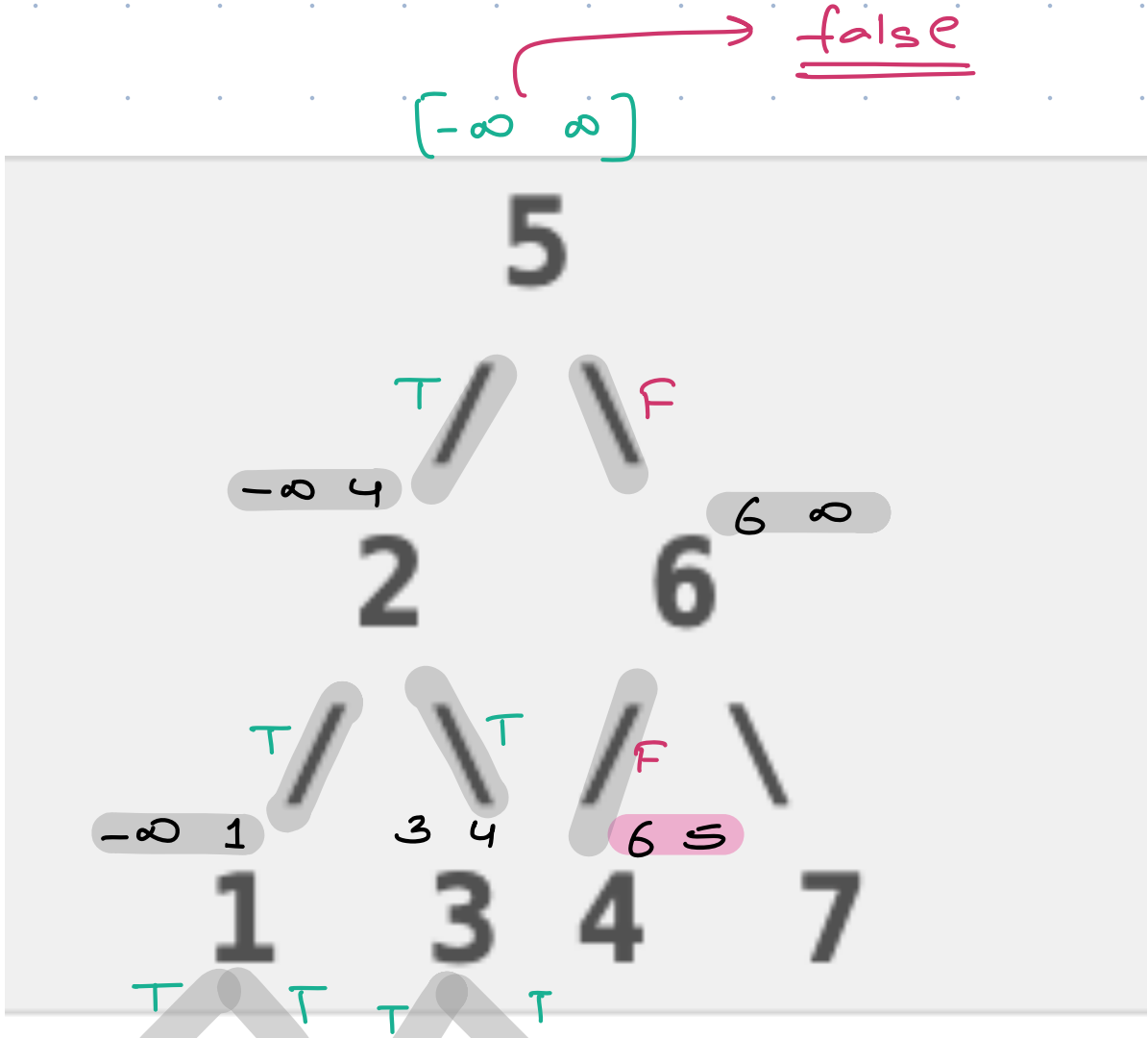
right

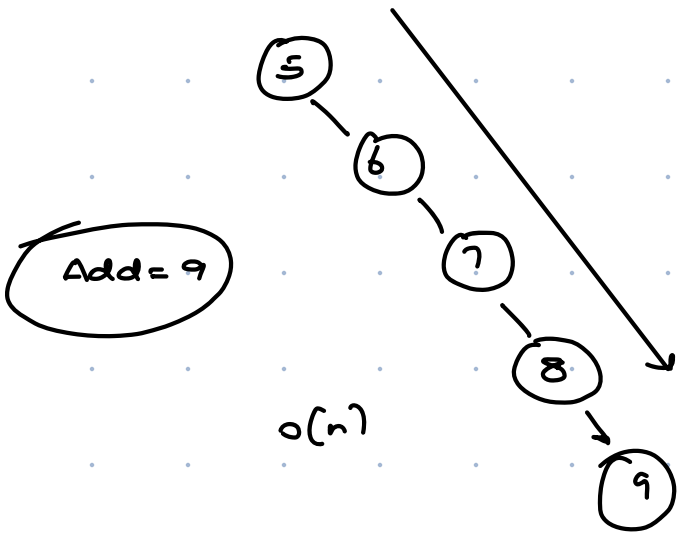
s → root.val +1

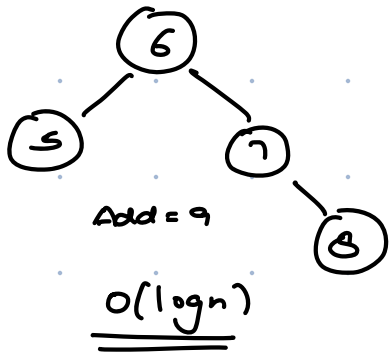
e → remains same

```
boolean isBST ( Node root , int s , int e)
    if (root == Null) return true;
    if ( s ≤ root.val && root.val ≤ e )
        boolean l = isBST (root.left , s , root.val-1);
        if ( l == false ) return false;
        boolean r = isBST (root.right , root.val+1 , e);
        if (r == false) return false;
        return true;
    else return false;
```

Tc : O(n)
Sc : O(ht)




5
6
7
8
9
Add = 9
O(n)


6
5
7
Add = 9
8
O(logn)