TREES 3 : BST

Good Evening

Today's content

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

Friday

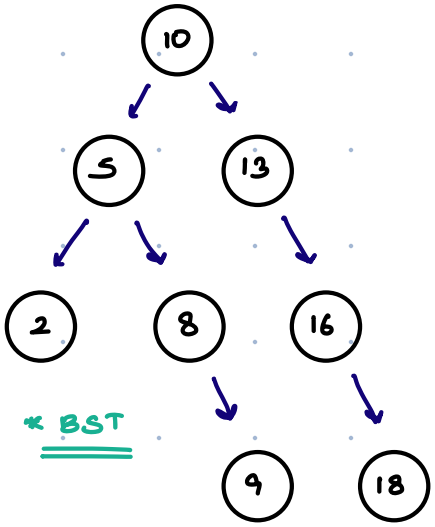
Doubt session

9 to 11:30 pm

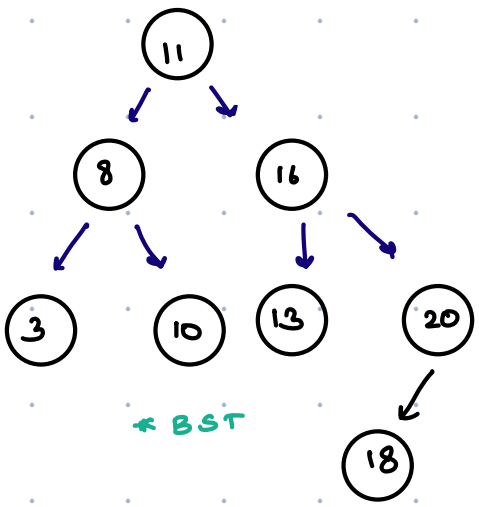
A binary tree is BST

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

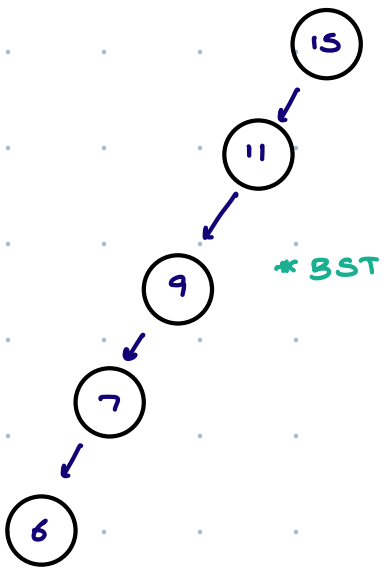
for all nodes



* BST

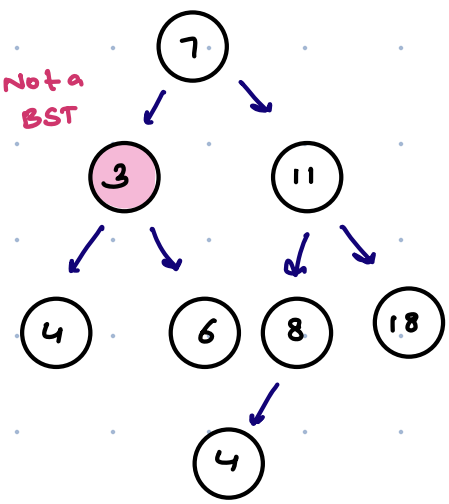


* BST



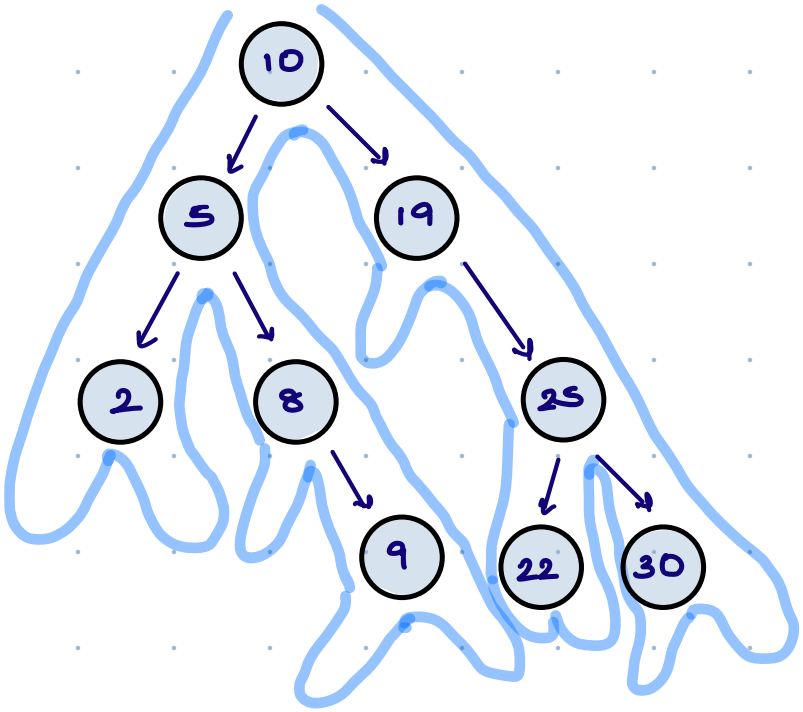
* BST

* Not a BST



* valid BST

* Special property of BST



Inorder ⇒ 2 5 8 9 10 19 22 25 30

* For a BST, inorder traversal will give sorted data in ascending order.

Scenerio

Flipkart stores and manages the history of all the orders that were processed and stores them efficiently such that whenever some information is required regarding any order, it can be fetched easily.

Problem

Develop a system for Flipkart that efficiently stores and manages the history of all processed orders. The system should allow easy access to information regarding any specific order when needed.

Requirements

There are two **types** of requirements :

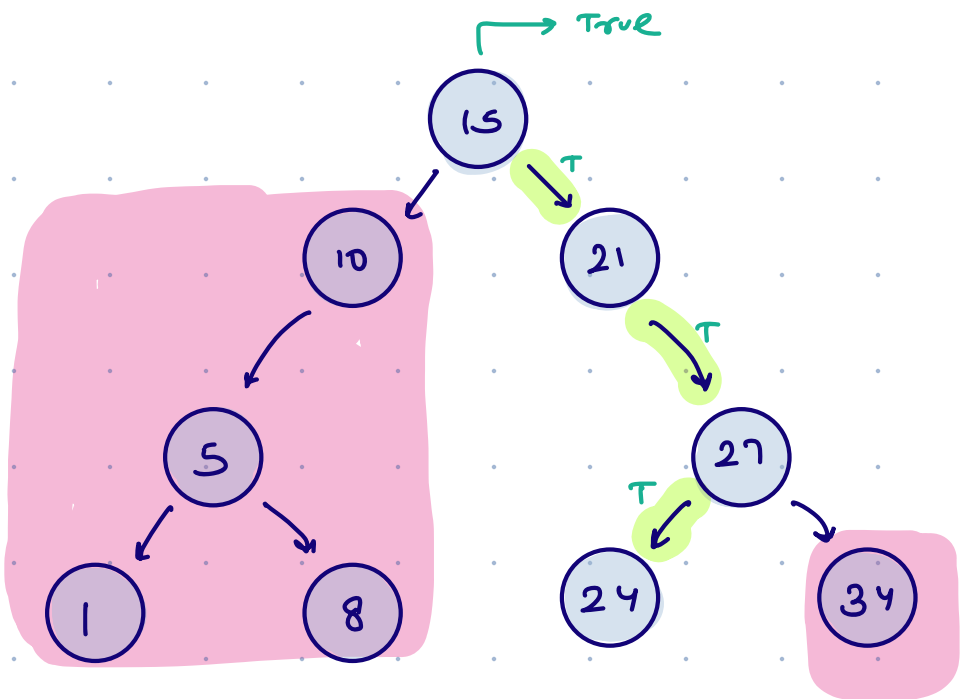
1. **Add Order:** The system should be able to record a new order by storing its **unique order ID** and the time it was placed.
2. **Find Order Time:** The system should be able to quickly retrieve the time of placing the order, given the **unique order ID**.

* My requirements

01. Insertion
02. Searching
03. Deletion

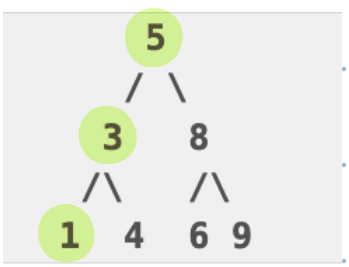
} BST → O(logn)

+ Search in BST



K = 24

Traverse on this tree 3 times.



Shantanu Kumar To: Me 9:46 PM

what if the requirement is if not found return the closedt number

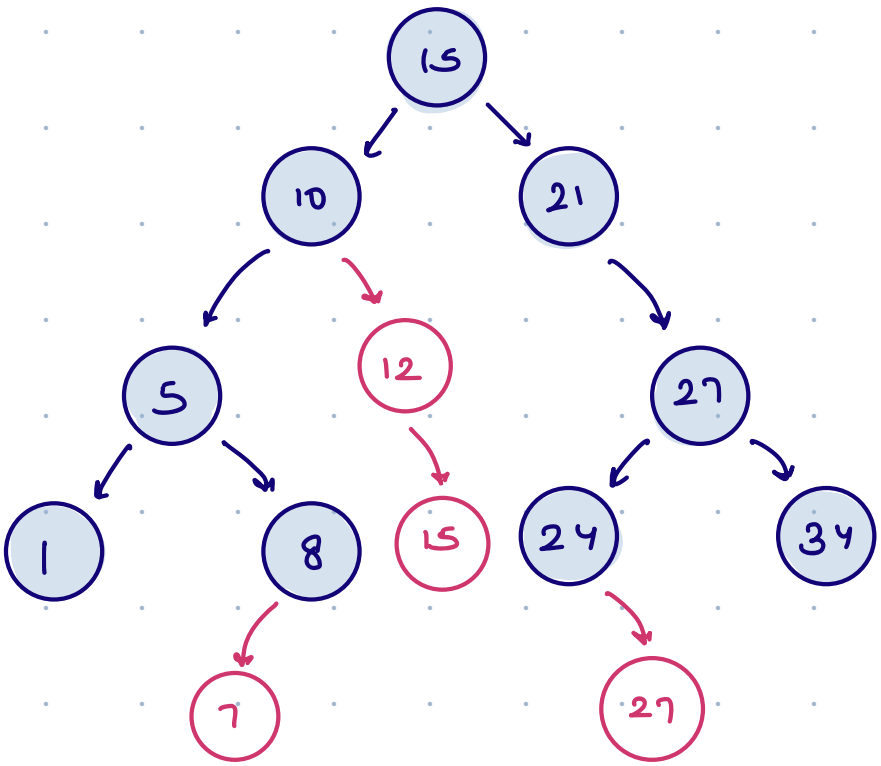
→ Friday's Doubt session.

```
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(H)

SC : O(H)

* Insertion in BST



K = 12

K = 7

K = 27

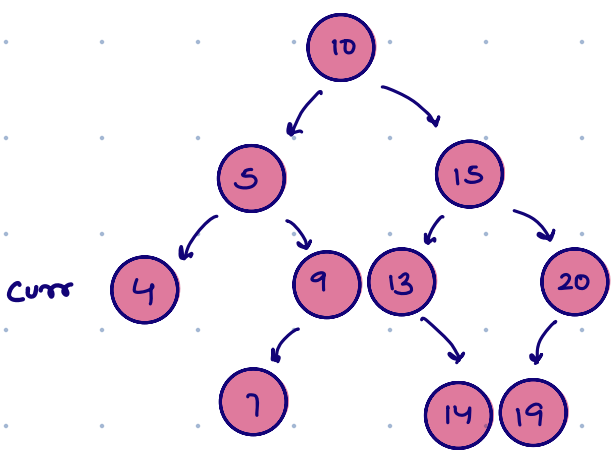
K = 15

K = 23

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

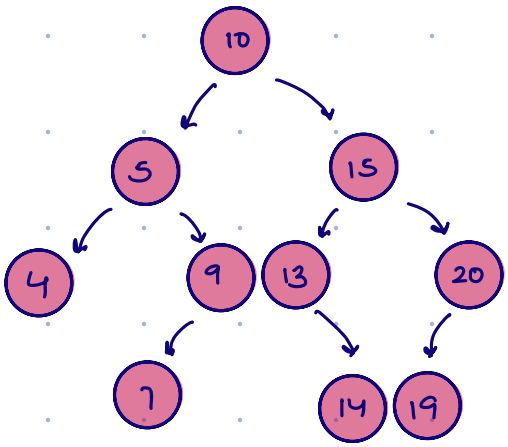
TC : O(H)
SC : O(H)

Min in BST



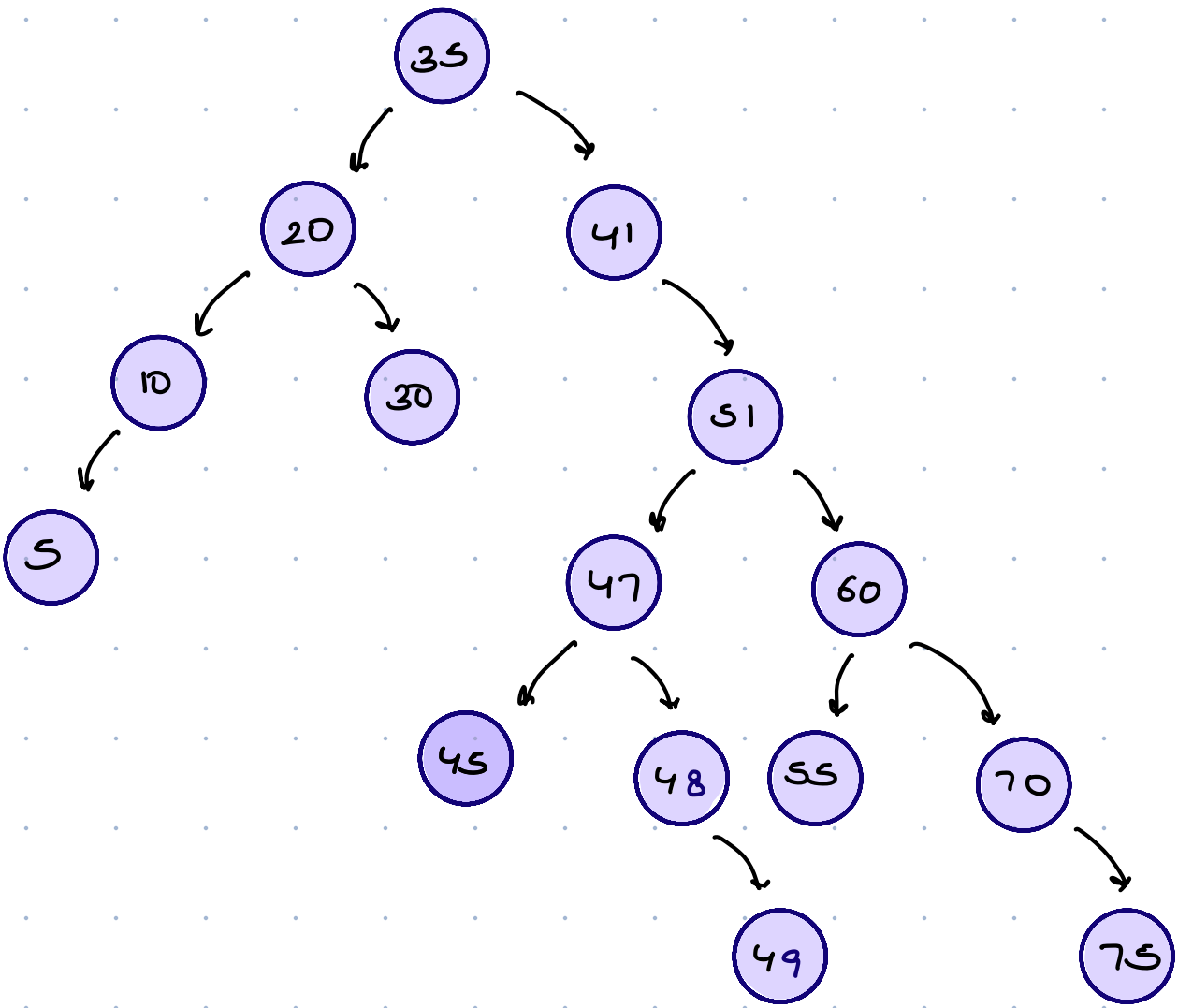

```
curr = root;
while (curr.left != null) {
    curr = curr.left;
}
return curr;
```

* Max in BST




```
int maxi (root){
    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 1 child

K = 51

K = 20

} Delete a node with two children

```
Node DeleteBST ( root, K){

    if (root == null) return null;

    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){

            // 0 child
```

```
        return null;
    }
    else if (root.left != null && root.right == null){
        // single left child
        return root.left;
    }
    else if (root.left == null && root.right != null){
        // single right child
        return root.right;
    }
    else {
        int max = maxi(root.left);
        root.val = max;
        root.left = DeleteBST(root.left, max);
    }
}
return root;
}
```

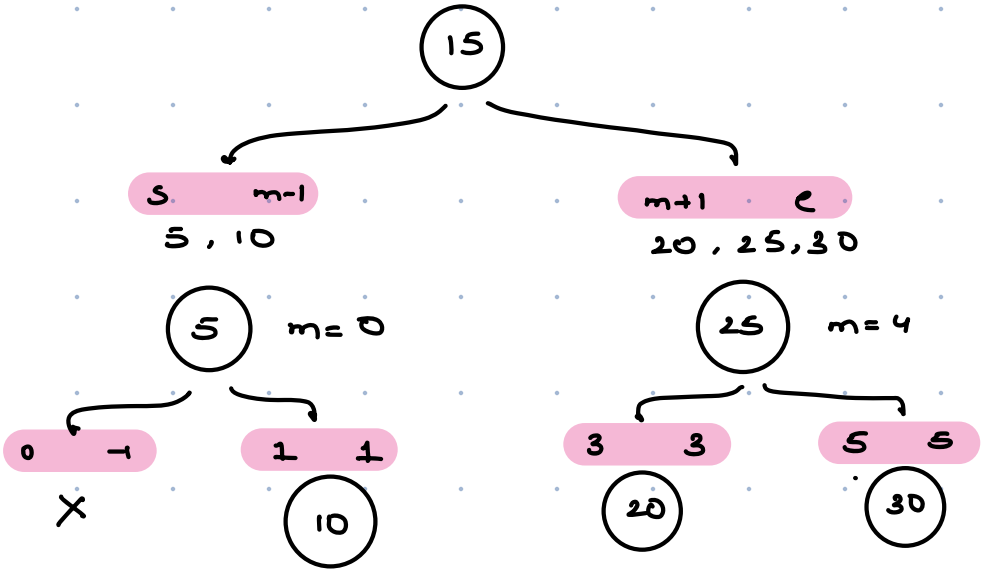
10:47 PM → 10:57 PM

+2

* Construct a **balanced BST** using an sorted array of distinct elements. $|LSTht - RSTht| \leq 1$

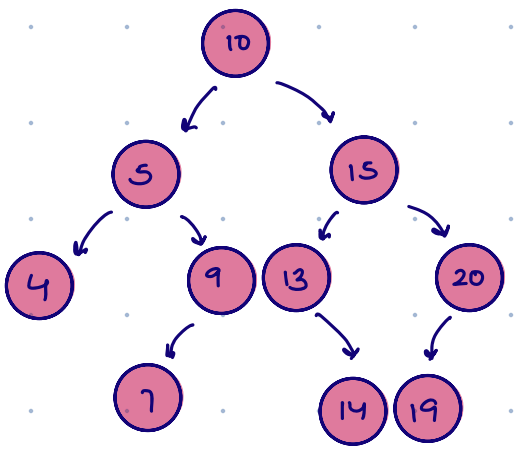
$$A[] = \{ \underset{s}{5} \quad 10 \quad \underset{m}{15} \quad 20 \quad 25 \quad \underset{e}{30} \}$$

$$\quad\quad 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5$$

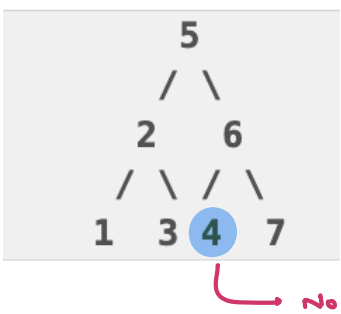


$m = \frac{s+e}{2}$

* Given a BT, check if it is BST or not.



∀ nodes, { all data in LST ≤ x (leftmax ≤ x)
all data in RST > x (rightmin > x)

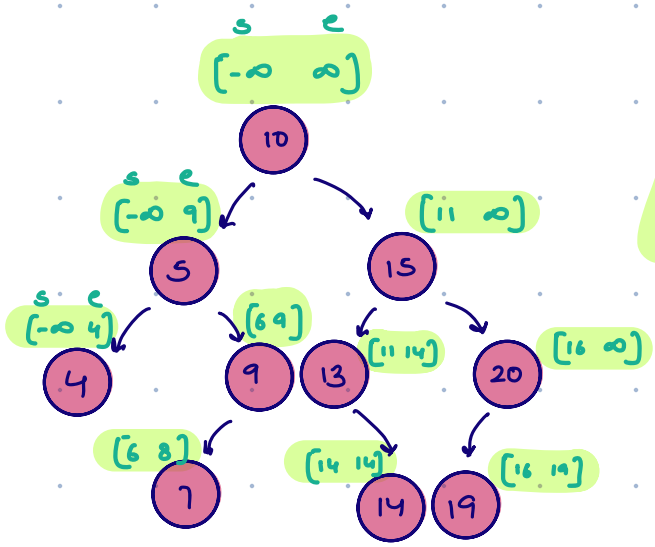


AL → 1 2 3 5 4 6 7

Brute force → store inorder of tree inside an Arraylist & check if data is in increasing order or not

TC: O(n)
SC: O(n)

* Approach 2



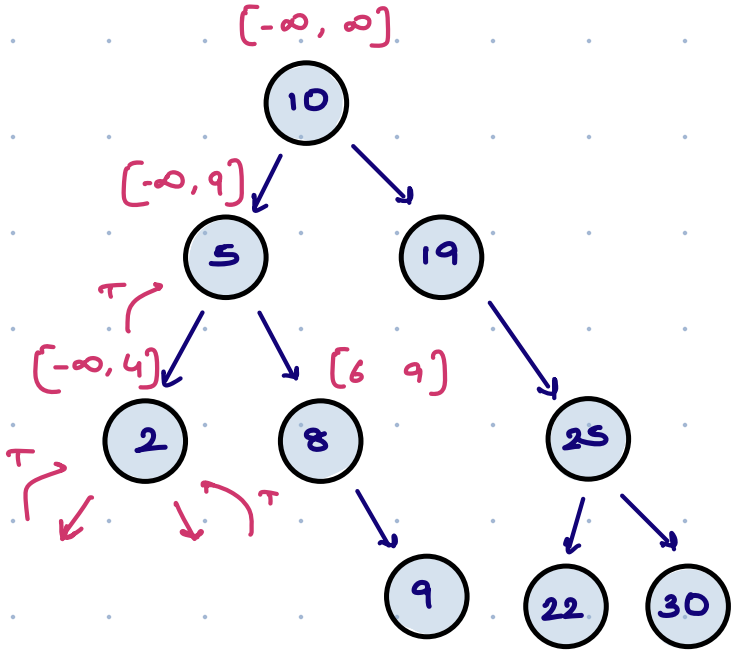
right
s → root.val + 1
e → remain same

left
s → remain same
e → root.val - 1

```
boolean isBST(root, -∞, ∞){
    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
        else return true;
    }
    else return false;
}
```

considering unique elements

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


[-∞, ∞]
10
[-∞, 9]
5
19
T
[-∞, 4]
[6 9]
2
8
25
T
T
9
22
30