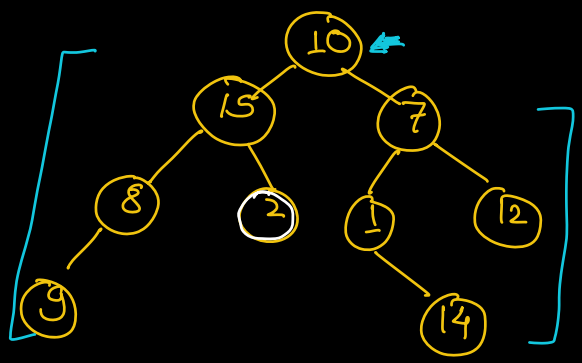
Q Given a binary tree & a no. K.
Check if K is present in the tree.

K = 2 → True

K = 20 → False

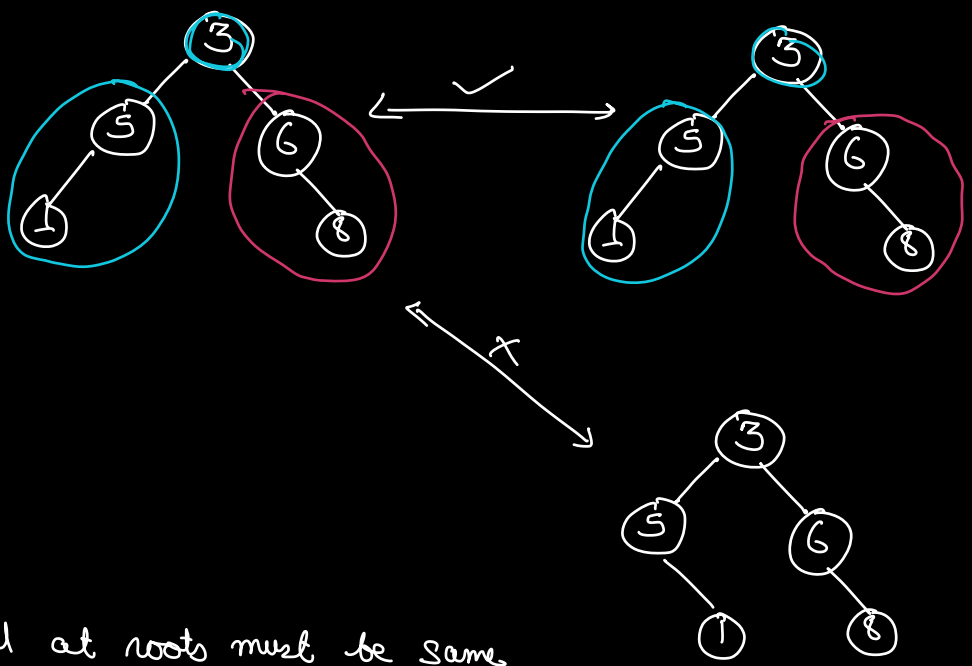
K = 10





Pre
Post
In
} O(N)

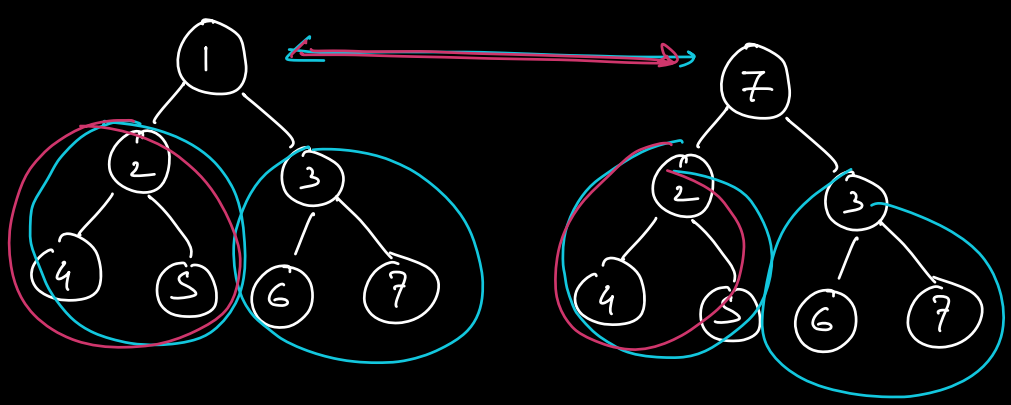
Q Given two binary trees. Return true if they are identical.




→ val at roots must be same.

→ T1.left & T2.left must be identical

→ T1.right & T2.right must be identical



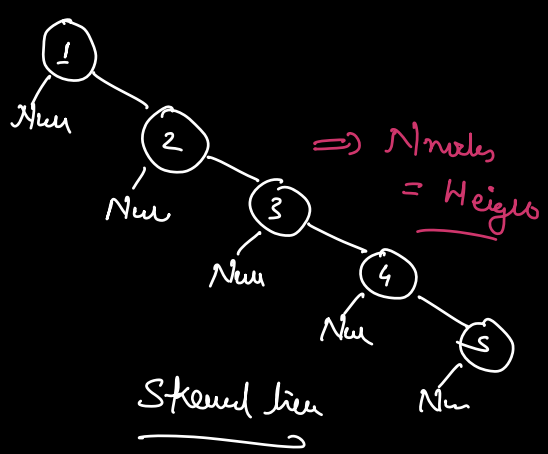
```
boolean isIdentical(root1, root2) {
    // Base Condition
    if (root1 == Null && root2 == Null)
        ret true;
    if (root1 == Null || root2 == Null)
        ret false;

    // Check root
    if (root1.val != root2.val)
        ret false;

    // Check LST & RST
    ret isIdentical(root1.left, root2.left)
            &&
        isIdentical(root1.right, root2.right);
}
```

Preorder Traversal

Fail fast

TC: O(N)

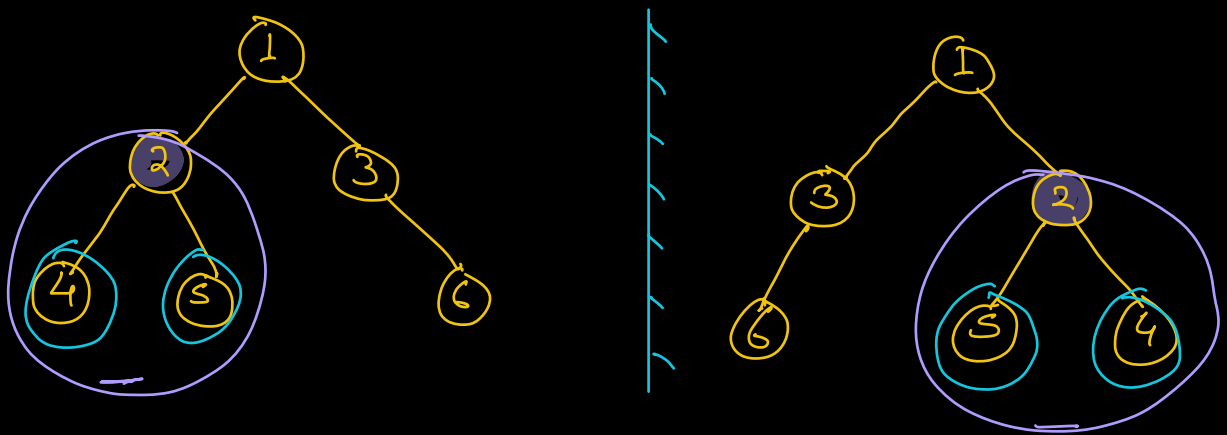
SC: O(N)



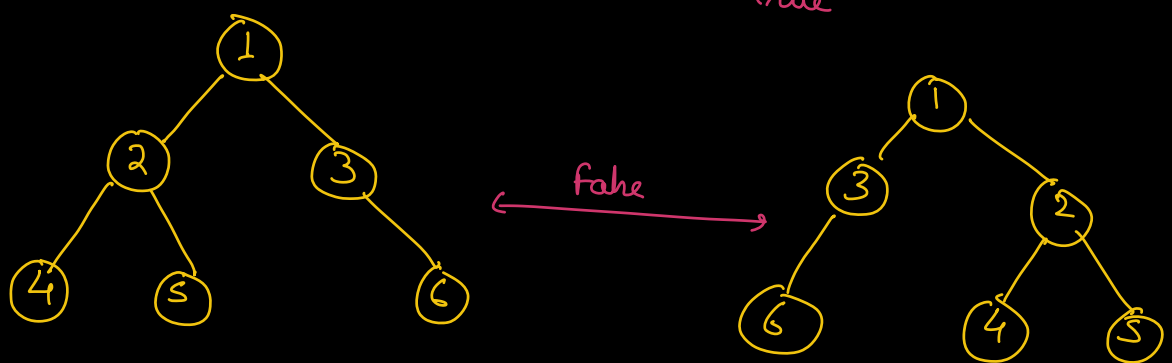
Q Given two binary trees.

Check if they are symmetric to each oth

(Mirror image)




True




Observations

① Roots' values must be the same.

② Root's right becomes roots left & vice-versa

```
boolean isMirror (root1, root2) {
    // Base Condition
    if (root1 == Null && root2 == Null)
        ret true;
    if (root1 == Null || root2 == Null)
        ret false;

    // Check root
    if (root1.val != root2.val)
        ret false;

    // Check LST & RST
    ret isMirror (root1.left, root2.right)
            &&
        isMirror (root1.right, root2.left);
}
```

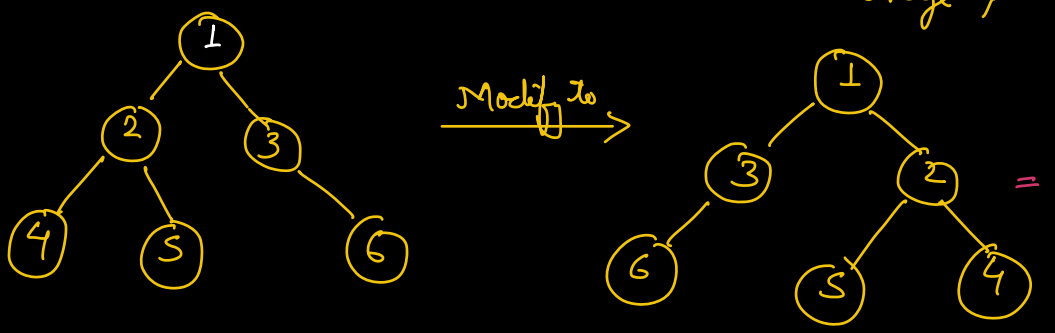
Pre Order

TC : O(N)

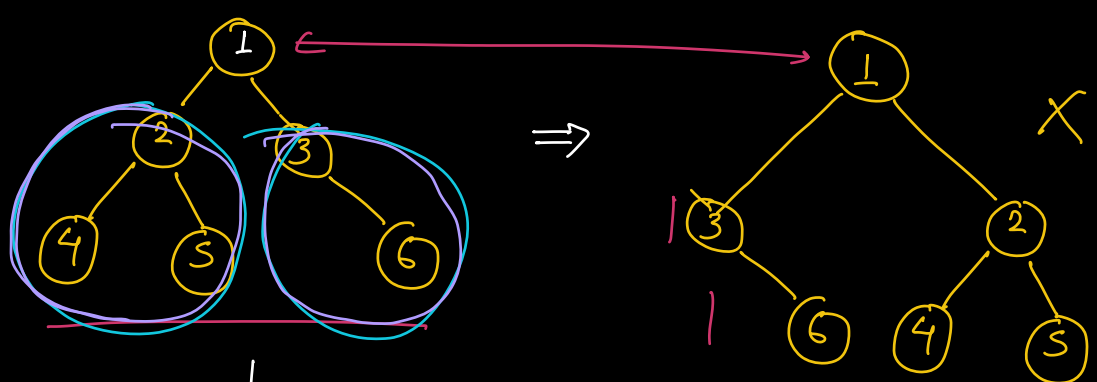
SC : O(N)

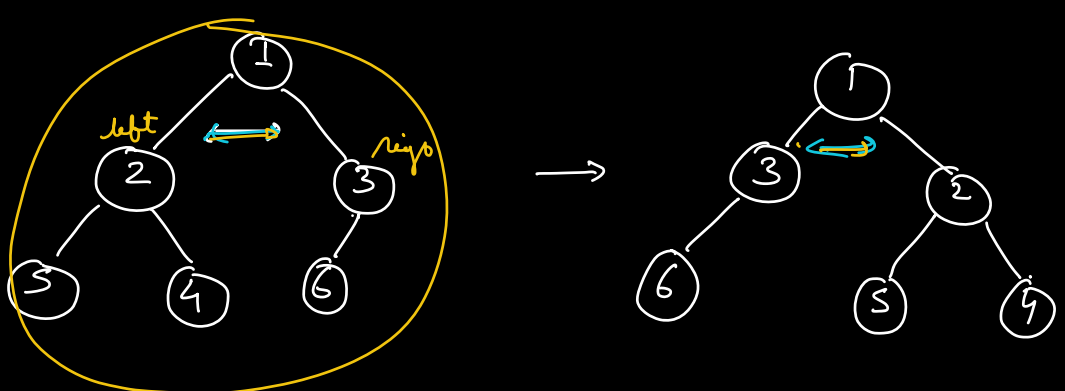
Max Howell ⇒ Homebrew (Mac)

Q Given a binary tree. Invert it.
(Convert it to its mirror image)

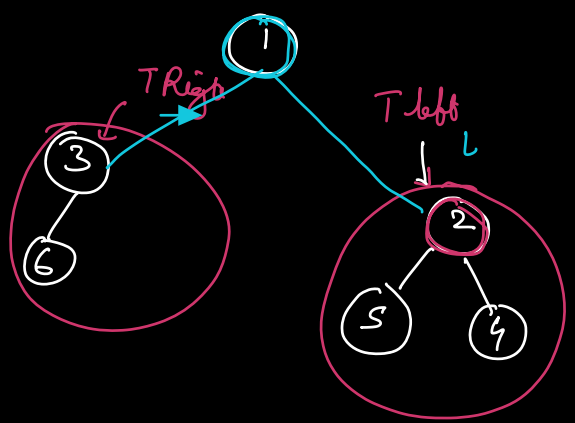



TreeNode invert (root) {



```
TreeNode invert(root) {
    if(root == null) ret null;

    TreeNode TRight = invert(root.right);
    TreeNode TLeft  = invert(root.left);

    root.left  = TRight;
    root.right = TLeft;
    ret root;
}
```

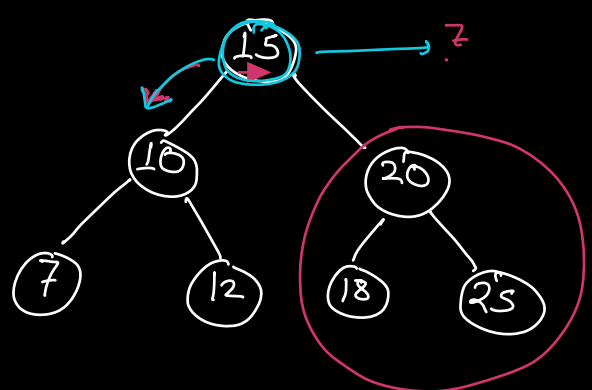



## Binary Search Tree

For every node

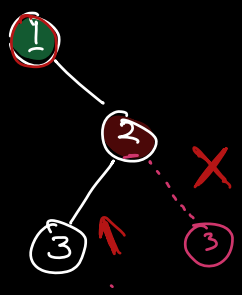
all values on LST $\leq$ Node's val

all values on RST $>$ Node's val



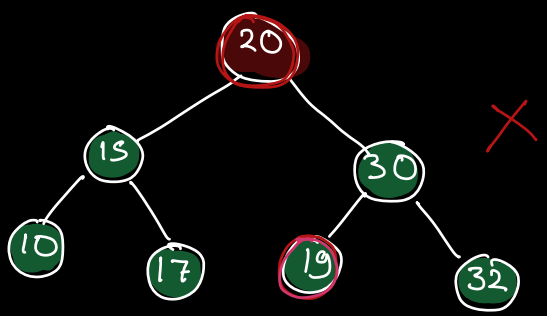

7, 10, 12, 15, 18, 20, 25

a)

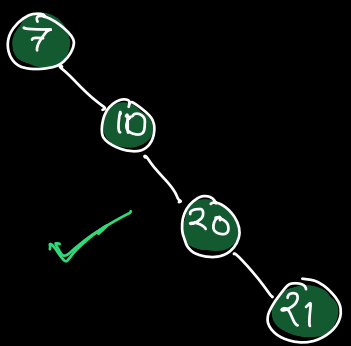



b)


c)



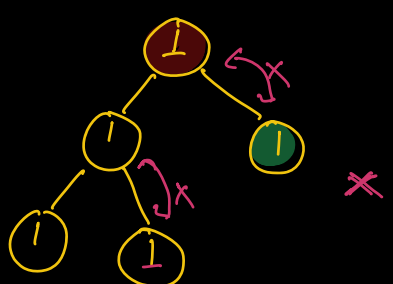

d)



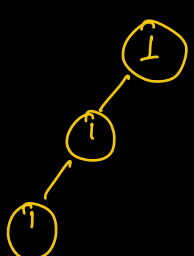

e)

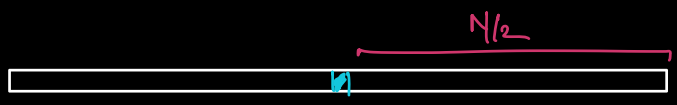
Null

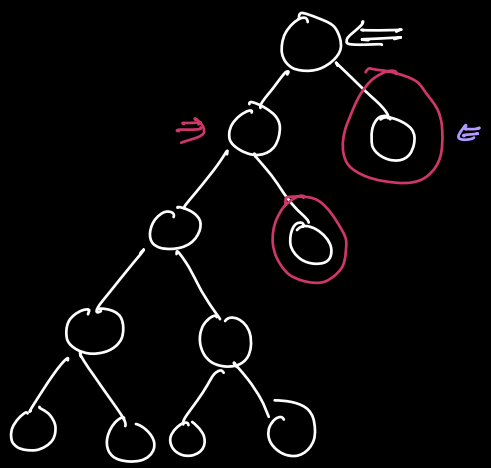
f)

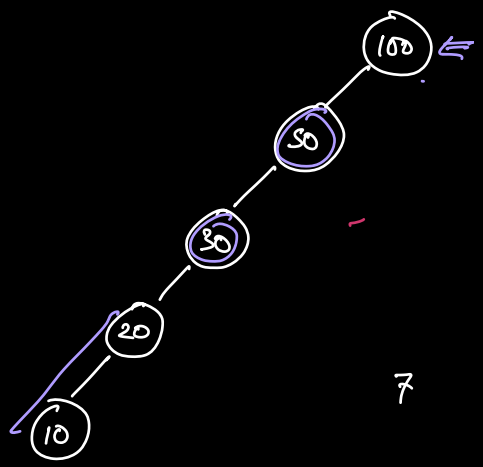



g)

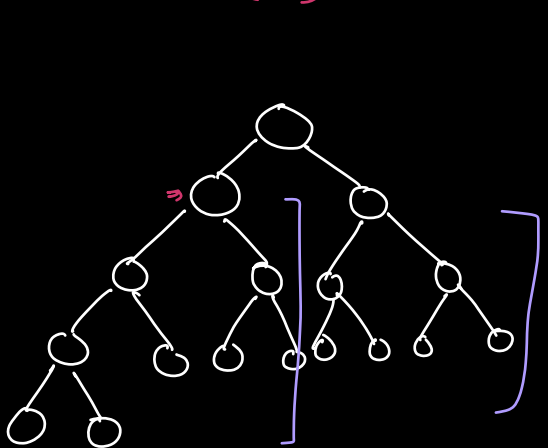


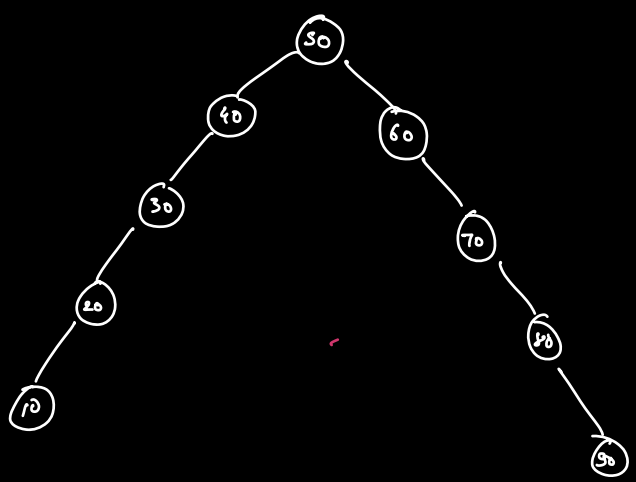





TC : O(N)



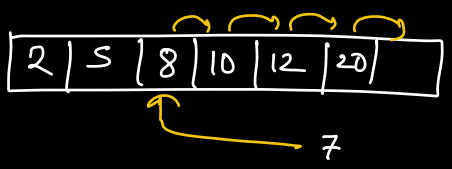



$|Ht(LST) - Ht(RST)| \leq 1$

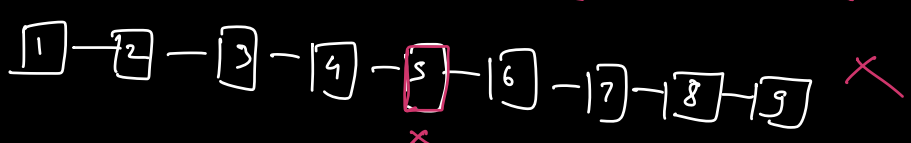
The absolute diff b/w ht of LST & RST $\leq 1$

Height Balanced Binary Search Trees : O(log N)

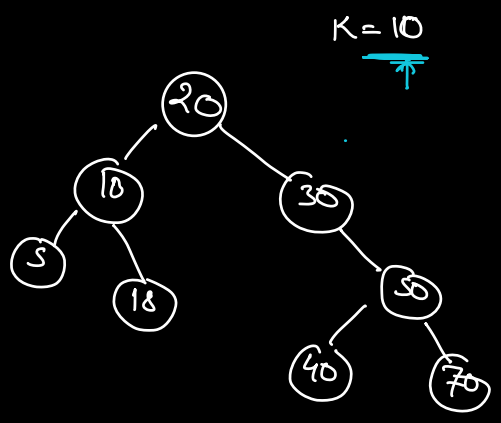
- Self Balancing BST $\Longrightarrow$ O(log N)
  (AVL / Red-black tree)

| | Sorted Array | Balanced BST |
|---|---|---|
| Search : | $O(\log N)$ | $O(\log N)$ |
| Insertion : | $O(N)$   | $O(\log N)$ |
| Delete : | $O(N)$ | $O(\log N)$ |
| Random Access : | $O(1)$ | × |


$\Leftarrow \log N$



Q Given a BST. Find a Key.




```
boolean searchBst(root, K){
    if(root == null)
        ret false;
    if(root.val == K)
        ret true;
    if(root.val > K)
        ret searchBst(root.left, K);
    else
        ret searchBST(root.right, K);
}
```

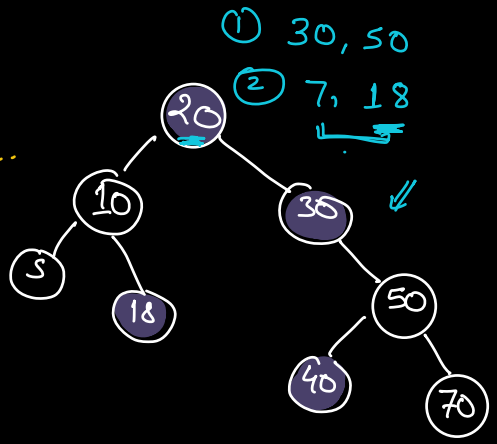
TC : O(N)

SC : O(N)
(Recursion Stack)

Q Given a BST & a range [S, E]

Count no of nodes inside the range.

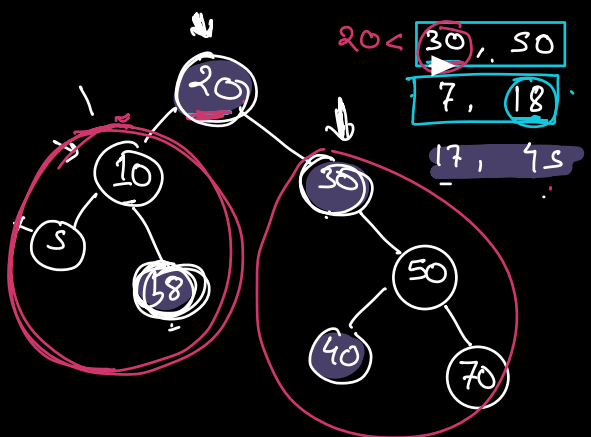
$S \le Node.val \le E$




[17, 45] ⇒ 4

S, E

- Root < S ≤ E ⇒ Count on RST
- S ≤ E < Root ⇒ Count on LST
- S ≤ Root ≤ E ⇒ 1 + Count on RST + Count on LST




20 < [30, 50]

[7, 18]

17, 45

[20, 50]

10 < [20, 50]

18 < [20, 50]

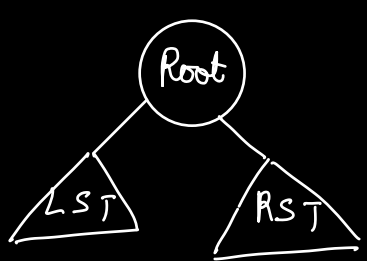
1 + C(RST) + C(LST)

```
int count (root, S, E) {
    if (root.val == Null) { ret 0; }
    // Root < S ≤ E
    if (root.val < S)
        ret count(root.right, S, E);
    // S ≤ E < Root
    if (root.val > E)
        ret count(root.left, S, E);
    // S ≤ Root ≤ E
    return (1) + count(root.right, S, E)
               + count(root.left, S, E);
}
```


Inorder Traversal

LST → Root → RST

BST




LST ≤ Root.val < RST

↳ Sorted order

- Binary Trees
- BST
- Traversal (Recursion)

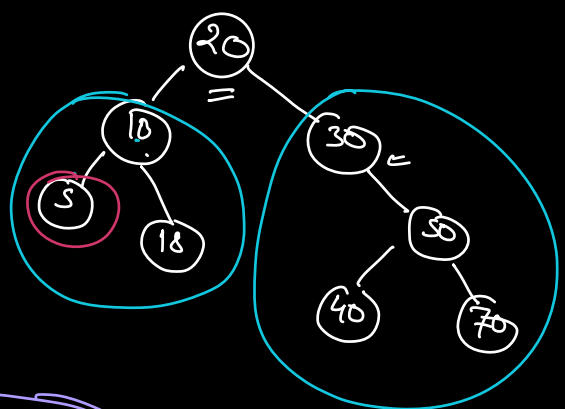
HashMap
HashSet
TreeMap
TreeSet

Heaps & Priority Queue

Seg Tree

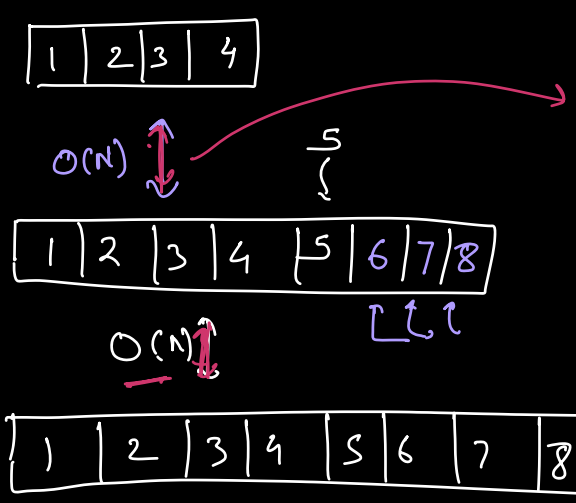
Trie (Trie)

5-6

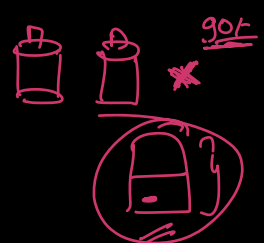



In Ord (10) $\leq$ 20 $<$ In Ord (30)

$5 \leq 10, < 18, < 20 < 30 < 40 < 50 < 70$




Amortized



- → Watch the lecture
- → Solve all problems from class
- → Remaining assignments
- → HW