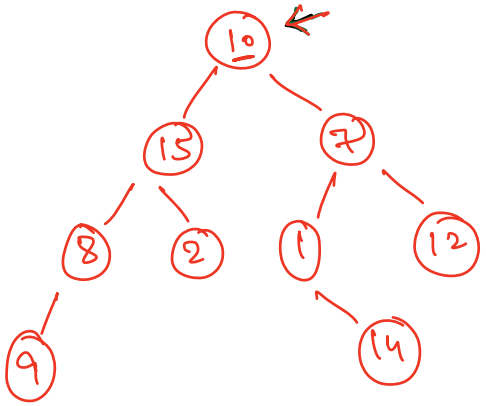
Q. Given a number K & a binary tree. Check if 'K' exists in the given B.T or not ?

⇒

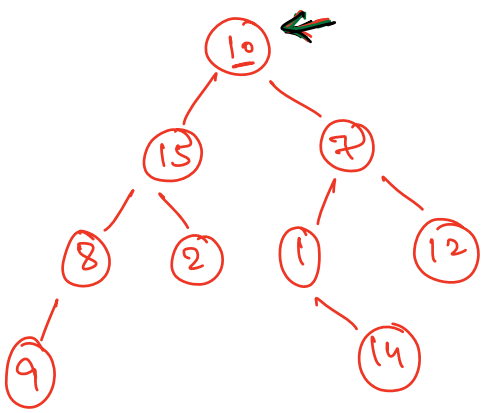


K = 2 ⇒ True

K = 20 ⇒ false.

⇒ Tree traversal.

- Pre order : O(N)
- Inorder : O(N)
- Post order : O(N)

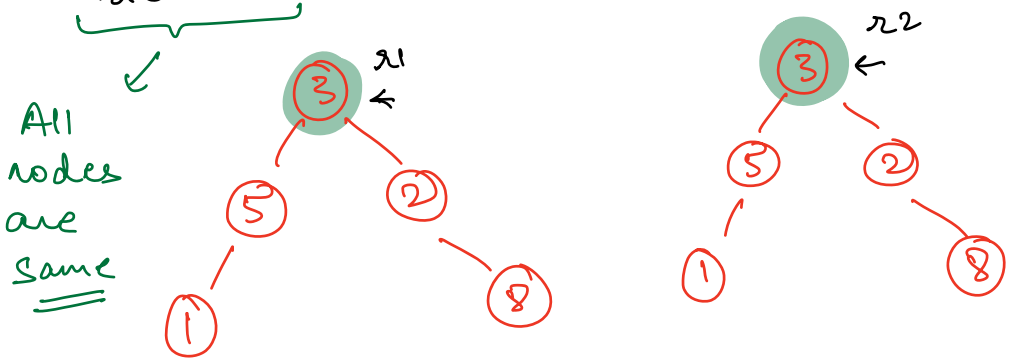


K = 10

- Pre Order ⇒ 1 Comparison.

⇒ fail fast Approach.

Q Given two Binary Tree's. Check if they are identical or not.




```
bool isIdentical(root1, root2){
    // If both roots are Null.
    if(root1 == null && root2 == null);
        return true;
    // If one of the root is Null.
    => if(root1 == null || root2 == null)
        return false;
    // If root's value are not same
    => if(root1.val != root2.val)
        return false;

    return [isIdentical(root1.left, root2.left)
            &&
            [isIdentical(root1.right, root2.right)
}
```

Base Cases.

Root

Which Traversal ? ⇒ Pre Order Traversal.

fail fast approach :- ✓

if cond1 && cond2

↓

false

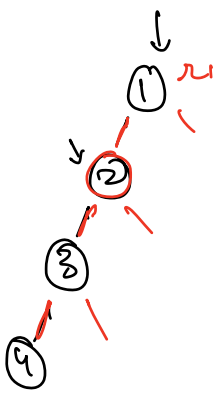
(F) && T/F → (F)

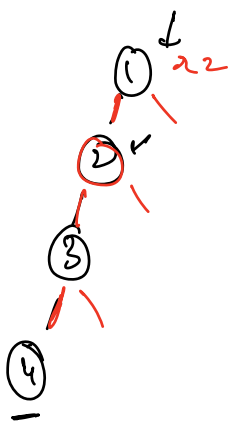
If cond1 is false then cond2 will not be evaluated.

TC : O(N)

SC : O(N)

H = 4





Skewed Tree's

f(1,1)

↓

f(2,2)

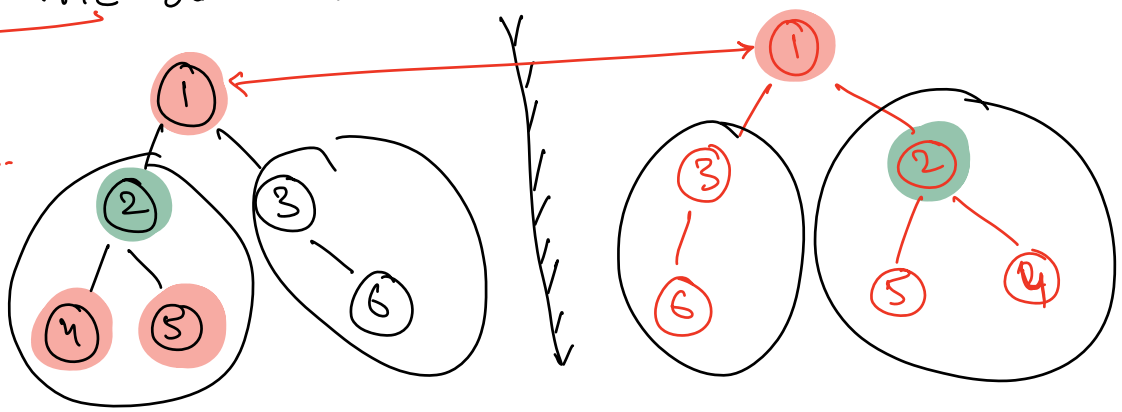
f(3,3)

f(4,4)

O(N)

Q Given two binary tree's. check if they are symmetric or not.

Mirror images.



Observations:

1) Root's value must be same.

2) Root1's left becomes Root2's right & vice-versa.

```
bool isSymmetric ( root1 , root2 ) {
    // If both roots are Null.
    if ( root1 == null && root2 == null )
        return true;
    // If one of the root is Null.
    if ( root1 == null || root2 == null )
        return false;
    // If root's value are not same
    if ( root1.val != root2.val )
        return false;
    return isSymmetric ( root1.left , root2.right )
        &&
        isSymmetric ( root1.right , root2.left );
}
```

Base Cases.

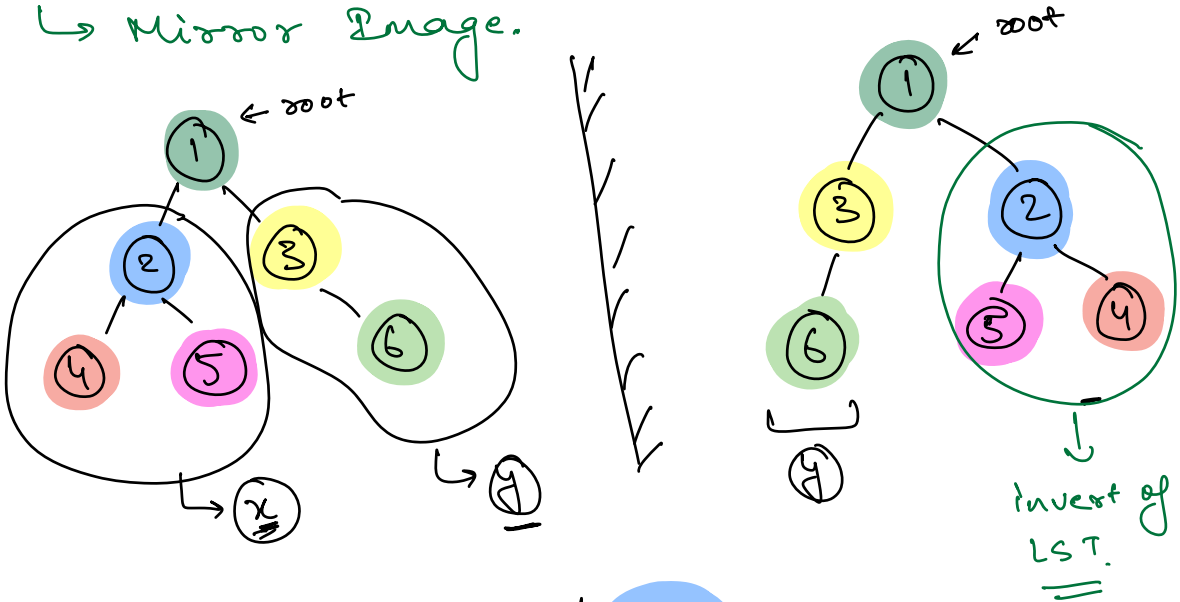
Root

TC : O(N)

SC : O(N)

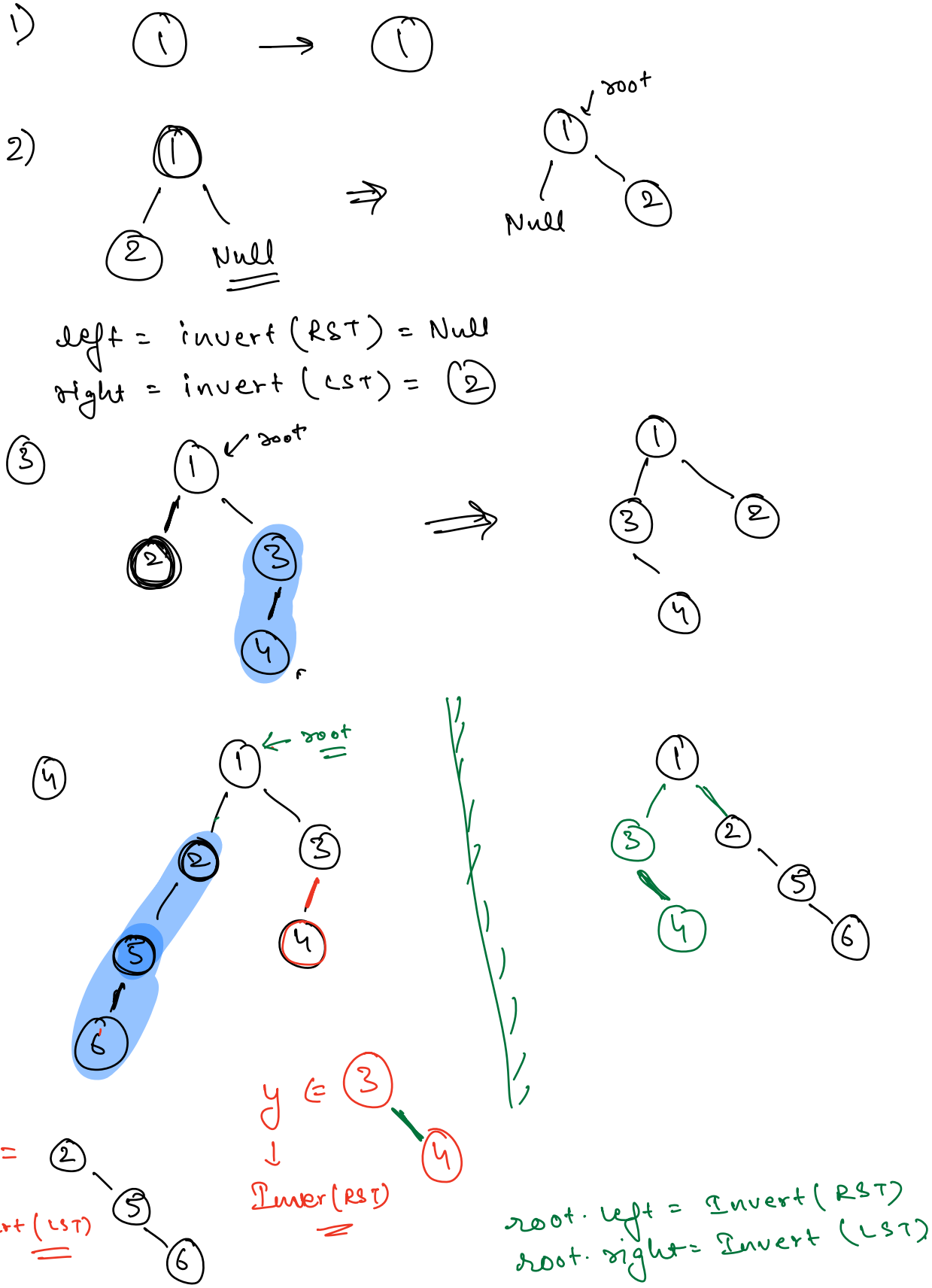
Q Invert a Binary Tree.

↳ Mirror Image.




```
TreeNode Invert (root) {
    if (root == null) return null;

    TreeNode invertLST = invert(root.left)
    TreeNode invertRST = invert(root.right)

    root.left = invertRST;
    root.right = invertLST;
    return root;
}
```

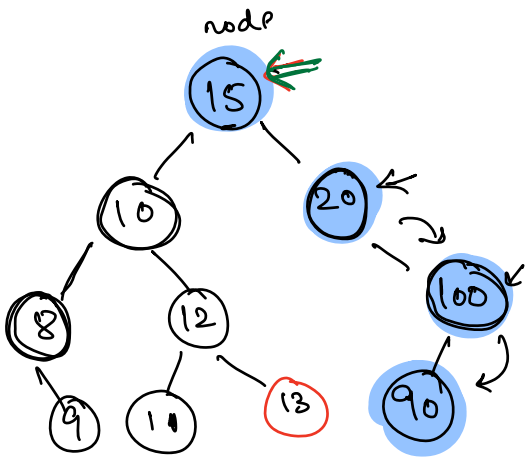
TC: O(N)


1)
2)
root
Null
left = invert (RST) = Null
right = invert (LST) = 2
3)
4)
root
x =
Invert (LST)
y =
Invert (RST)
root. left = Invert (RST)
root. right = Invert (LST)

⇒ TC of searching an element :

1) Array : O(N)
2) L.L : O(N)
3) Binary Tree : O(N)

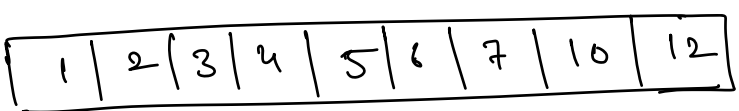
⇒ Specific order of data in Binary Tree will lead to better TC.



search K = 90

Binary Search Tree's (BST)

Sorted Array



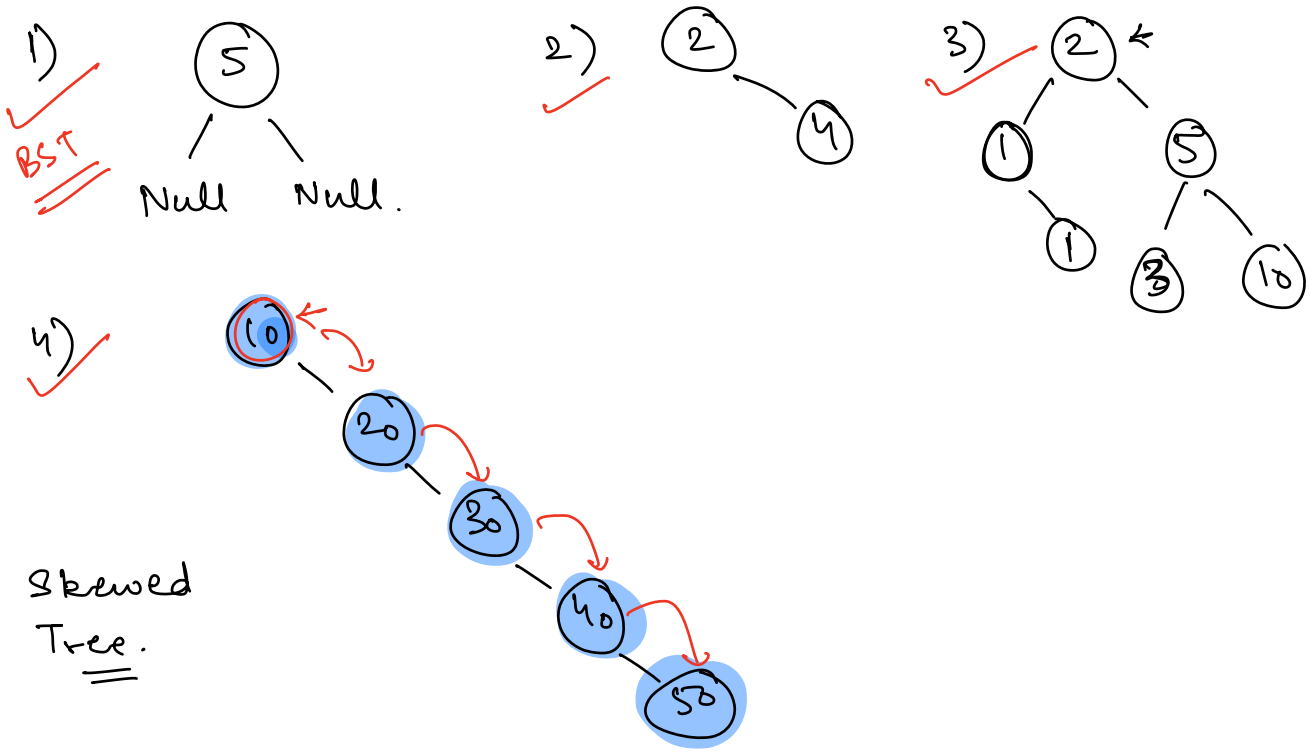
search (K)

Binary Search ⇒ O(log N)

For every node in BST :-

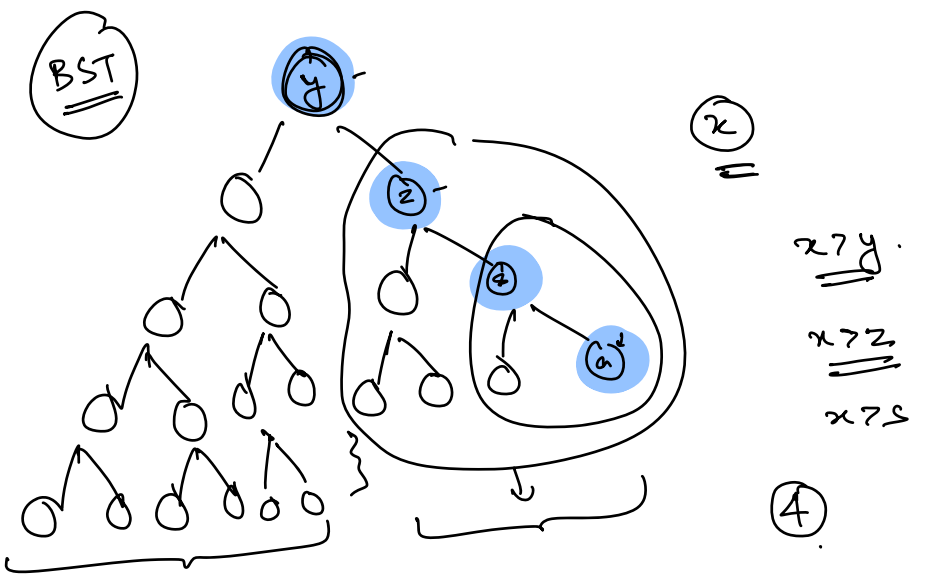
Value in LST ≤ node's value

Value in RST > node's value.



Search TC in BST : O(N)

Worst case



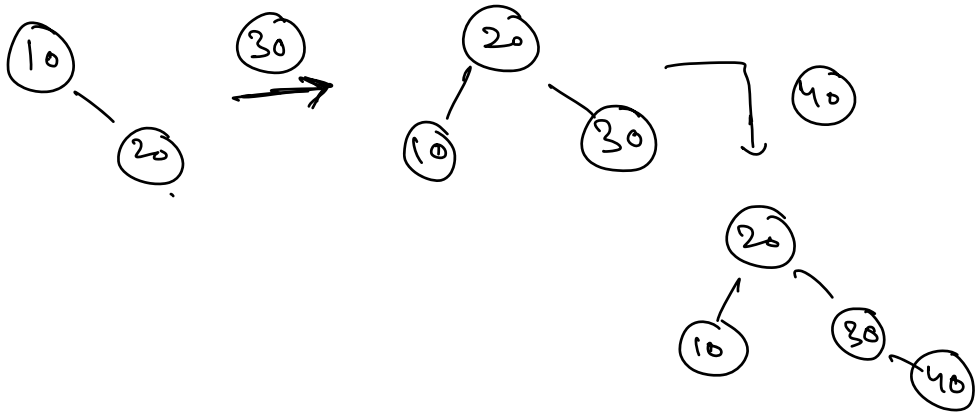
Balanced Binary Search Tree

→ Balanced BST

↳ AVL ←

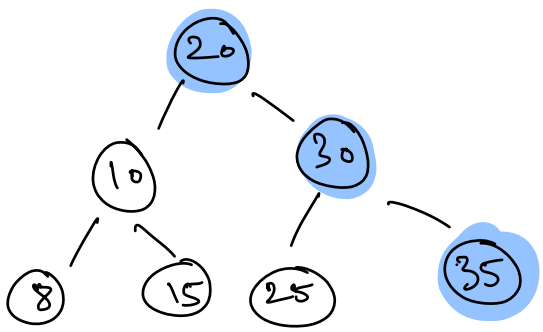
→ Red | Black Tree

} → self balancing BST.



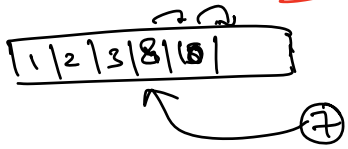
TC of search in Balanced BST : O(log N)

Balanced BST



Sorted Array VS Balanced BST

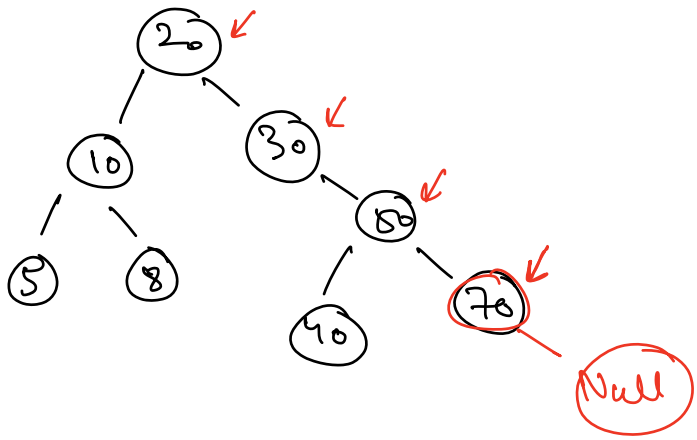
- Insertion. O(N)



Insertion :-

O(log N)

Q Given a BST, search a key in it. ⇒ T/F



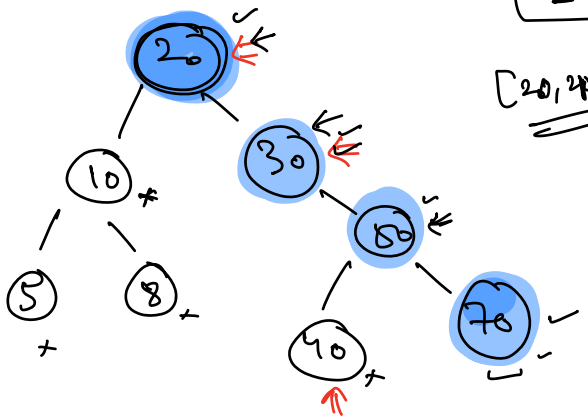
k = 10 → T

k = 100 → f.

```
bool search(root, key) {
    if(root == null)
        return false;
    if(root.val == key)
        return true;
    else if(root.val > key)
        return search(root.left, key)
    else
        return search(root.right, key)
}
```

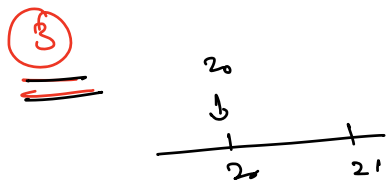
Q: Given a range [S, E] and a BST. Count the no. of nodes which have value inside the given range.

$S \le node.val \le E$

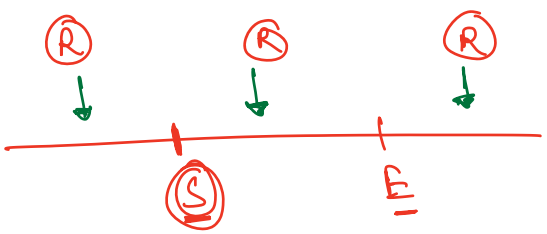


[20, 24]

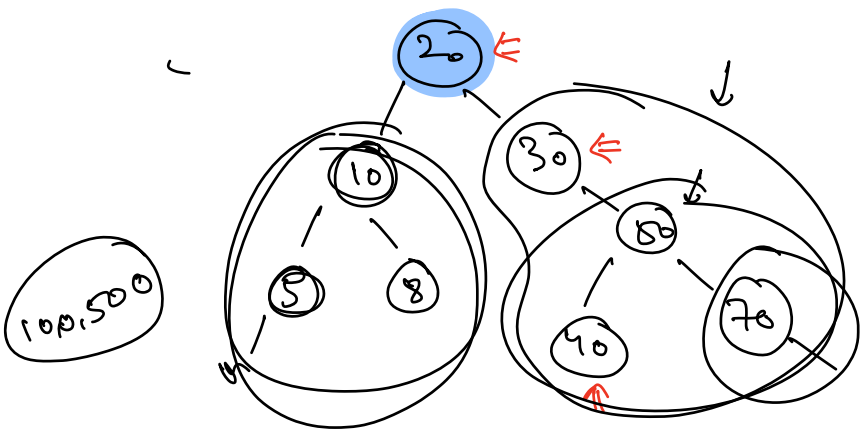
[17, 45]



- $R < S \le E \Rightarrow$ Count on the RST.
- $S \le R \le E \Rightarrow 1 +$ Count on RST + count on LST.
- $S \le E < R \Rightarrow$ Count on the LST.



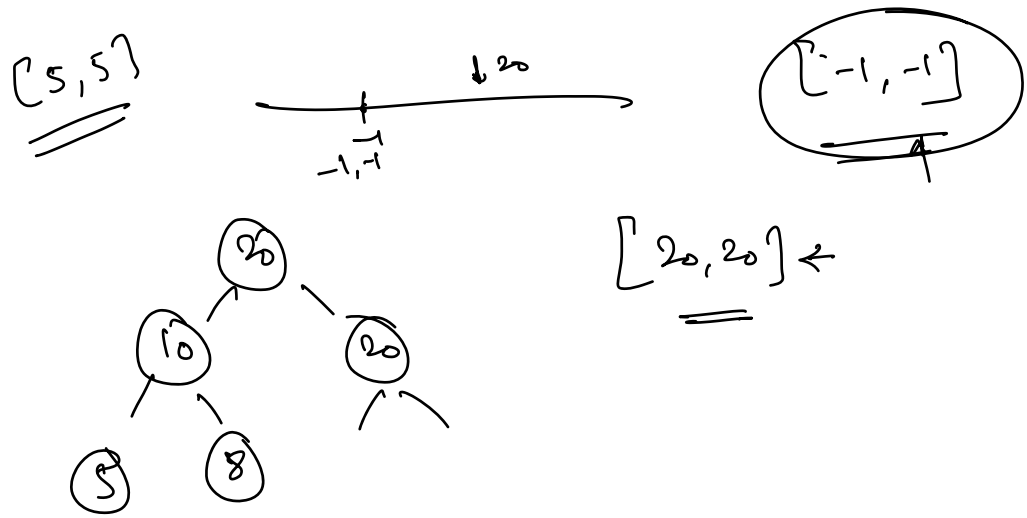
```
int countNodes(root, S, E){
    if (root == Null) return 0;
⇒   if (root.val >= S && root.val <= E){
return 1+ countNodes(root.left , S, E) +
                countNodes(roo.right, S, E);

    }
    if (root.val < S){
            return countNodes(root.right, S,
                                        E);

    }
    if (root.val > E)
return countNodes(root.left, S, E);

}
```

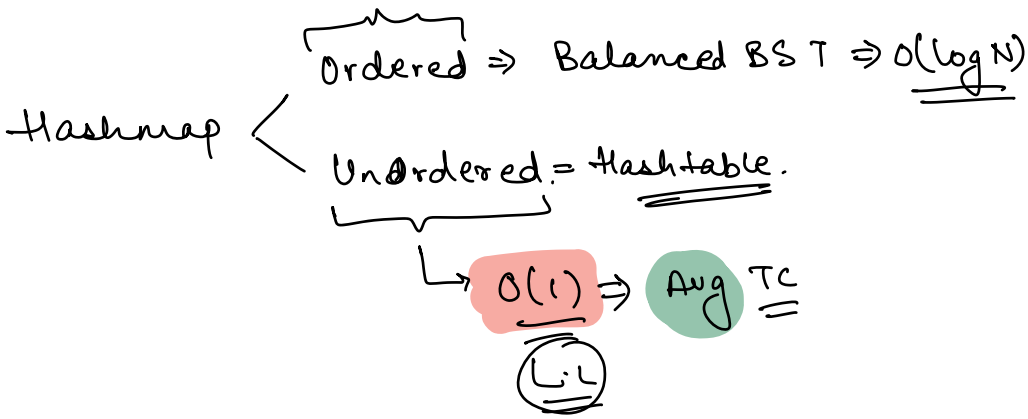


[20, 21]

20 >= 20 && 20 < 21

1+ 0 + 0


[5,5]
20
-1,-1
[-1,-1]
20
10
20
5
8
[20,20]


Ordered ⇒ Balanced BST ⇒ O(log N)
Hashmap
UnOrdered = Hashtable.
O(1) ⇒ Avg TC
L.L