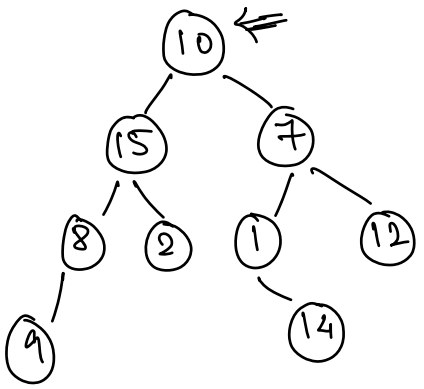
Q. Given a Binary Tree and a no. of (k). Search if (k) exists in Binary Tree or not.



K = 12 ⇒ True

K = 20 ⇒ false.

*1) PreOrder
2) InOrder
3) PostOrder

O(N)

Preference ⇒ PreOrder

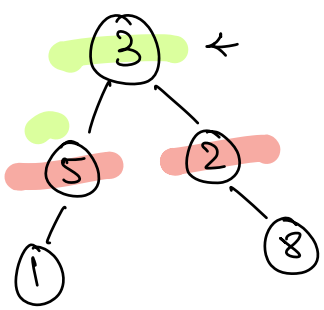
Fail Fast Approach ⇒

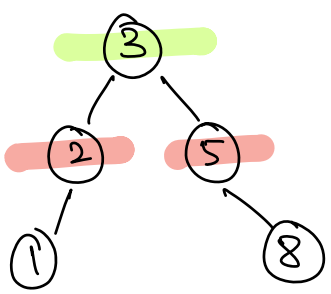
↳ If function is failing then it should be as fast as possible.

Q. Given two Binary Trees. Check if they identical

↳ root1

↳ root2





NOT identical

```
bool isIdentical(root1, root2) {
    // Base cases
    if (root1 == Null && root2 == Null)
        return true;
    if (root1 == Null || root2 == Null)
        return false;
    if (root1.data != root2.data)
        return false;

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

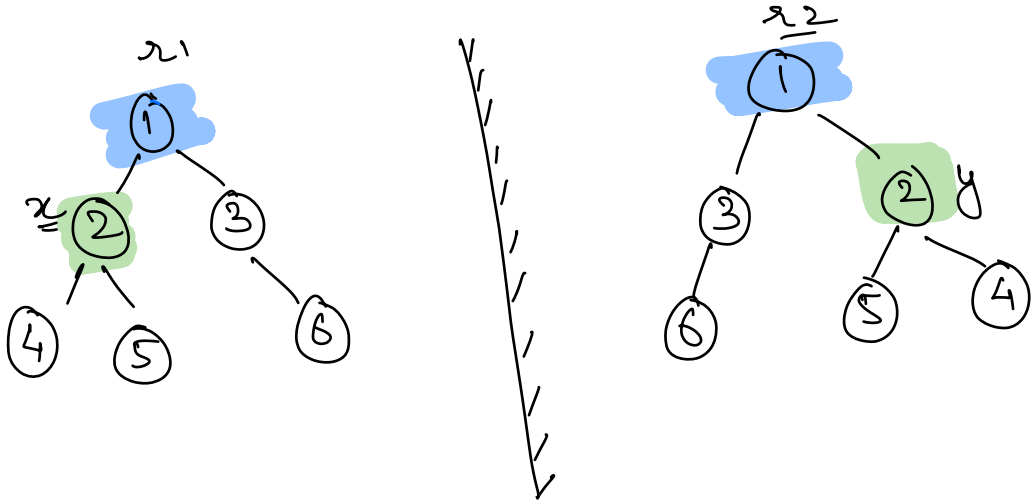
TC : O(N)

SC : O(N)

PreOrder Traversal

Q Given two Binary Trees, check if they are Symmetric or Not.

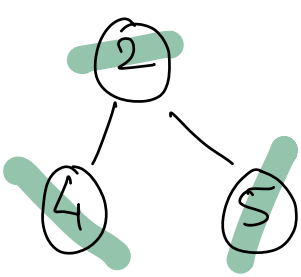
↳ Mirror image

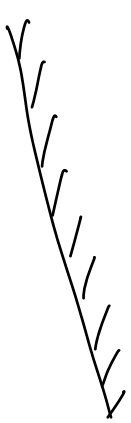


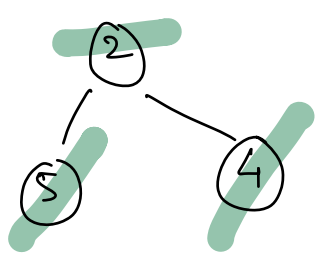
Observations

1) Root's value must be same.
2) Root's left becomes Root's right & vice-versa
   ↳ for all the subtrees.

```
bool isSymmetric(root1, root2) {
    // Base cases
    if (root1 == Null && root2 == Null)
        return true;
    if (root1 == Null || root2 == Null)
        return false;
    if (root1.data != root2.data)
        return false;
    return isSymmetric(root1.left, root2.right)
           &&
           isSymmetric(root1.right, root2.left);
}
```

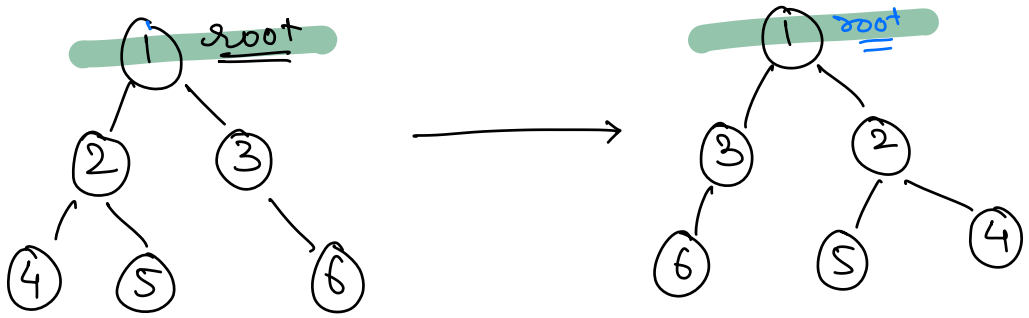






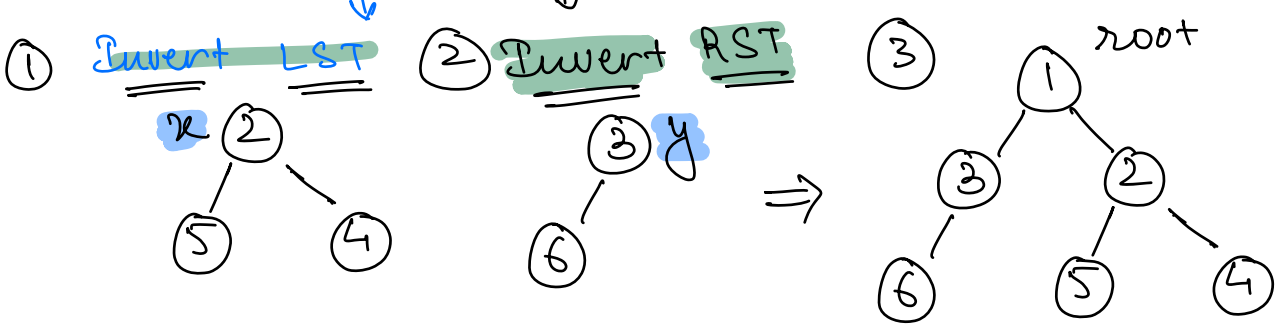
Q. Invert a Binary Tree.

↳ Convert it into its mirror image.



Input

Output




```
TreeNode invert(root) {
    if(!root){
        return root;
    }
LST [TreeNode x = invert(root.left);
RST [TreeNode y = invert(root.right);
root [root.right = x;
      root.left = y;
      return root;
}
```

TC : O(N)

Post Order Traversal

x = fun(root.left)

x = Null

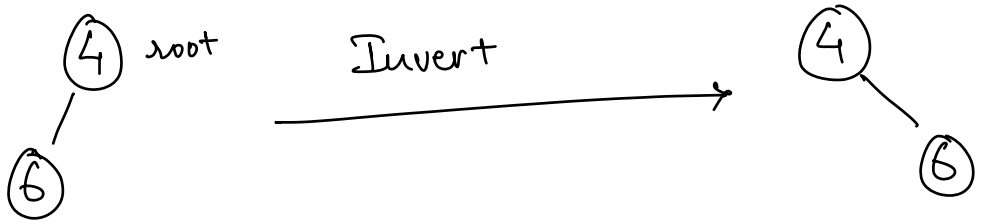
y = fun(root.right)

y = Null

root.right = x

root.left = y

⇒



x = fun(LST) (6)

x = 6

y = fun(RST)

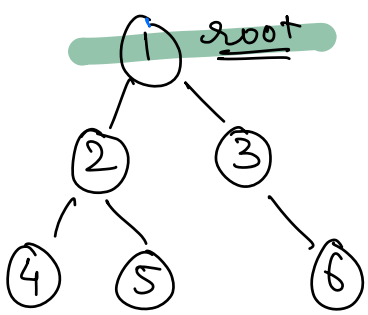
y = fun(Null)

y = Null

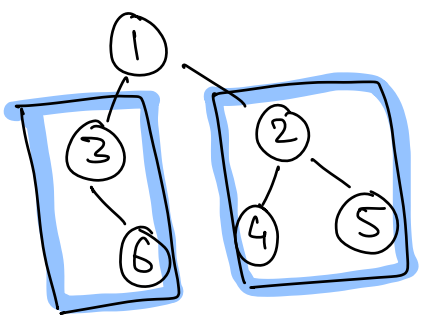
root.right = x;

root.left = y.

#



① Swap LST & RST



Pre Order Traversal

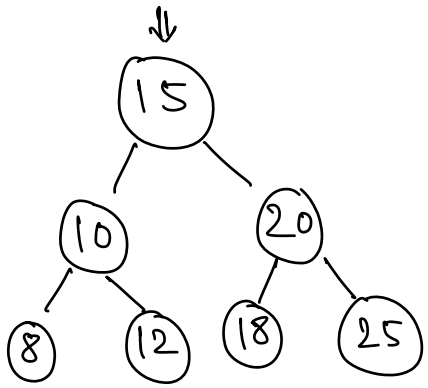
② invert(LST)

③ invert(RST)

Search an element in a Binary Tree

↳ TC : O(N)

TC of searching in Array / L.L :- O(N)



k = 25

Binary Search Tree (BST)

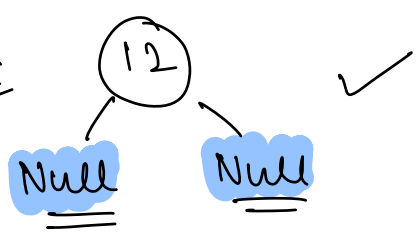
⇒ for every node :-

1) values on LST ≤ node's value

2) values on RST > node's value

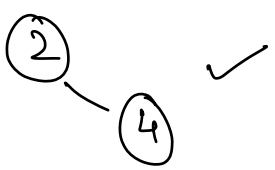
Eg :-

1.



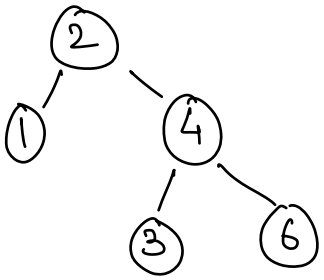
✓

2.



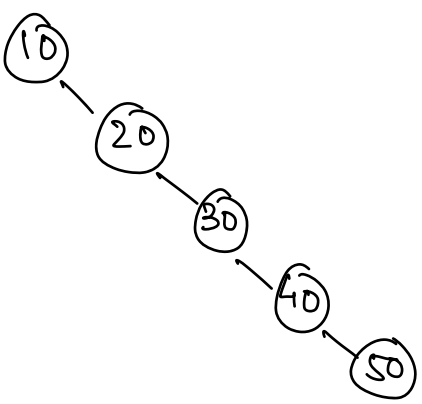
✓

3.



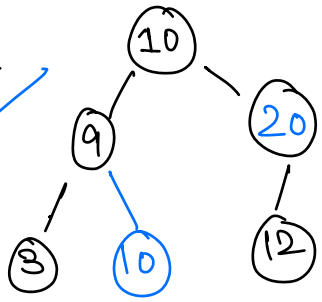
✓

4.

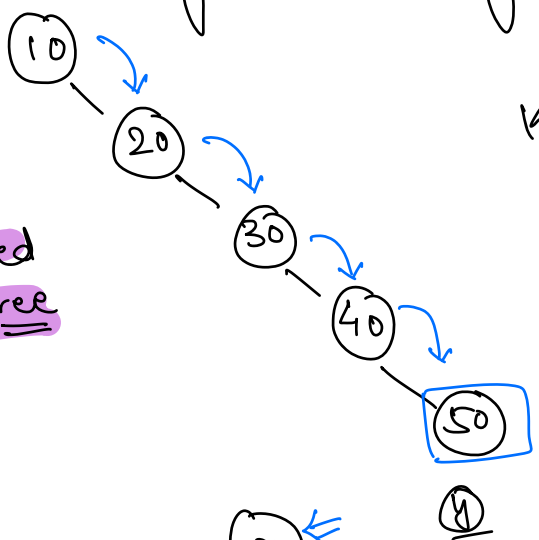


✓

5.



Quiz TC of searching in a BST

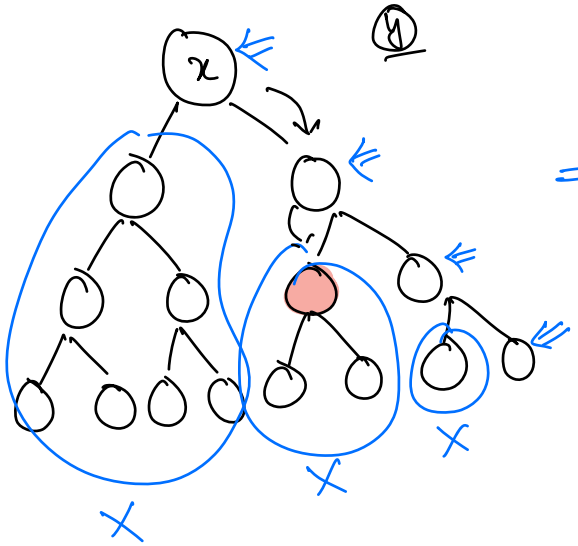


Skewed Tree

k = 50

⇒ TC: O(N) worst case

#



⇒ TC: O(log N)

No. of nodes in LST ≃ No. of nodes in RST ⇒ TC: O(log N)

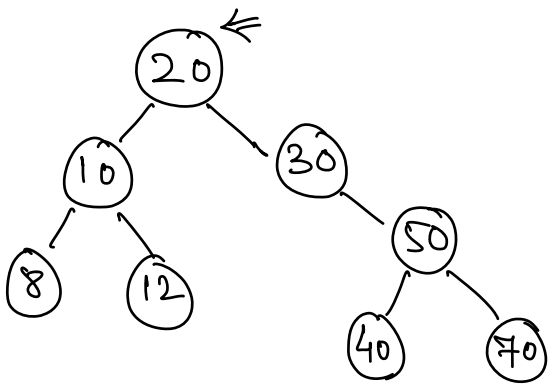
Balanced BST ⇒ O(log N)

↳ AVL Trees / Red Black Trees

Self Balancing BST

Sorted Array	Balanced BST
1) Insertion TC: O(N)	1) Insertion O(log N)
2) Deletion O(N)	2) Deletion O(log N)

Q Given a BST, search a key in BST



k = 10 ⇒ True

k = 100 ⇒ False

```
bool search(root, k){          fail
    if(root == Null)
        return false;
    if(root.data == k)
        return true;
    else if(root.data > k)
        return search(root.left, k);
    else
        return search(root.right, k);
}
```

TC: O(N)