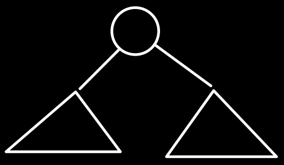
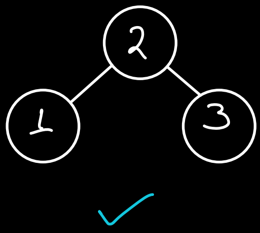
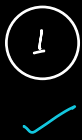
Binary Search Tree

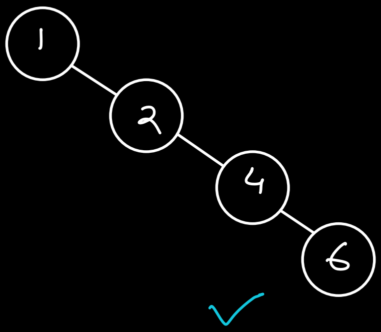
∀ nodes in the tree

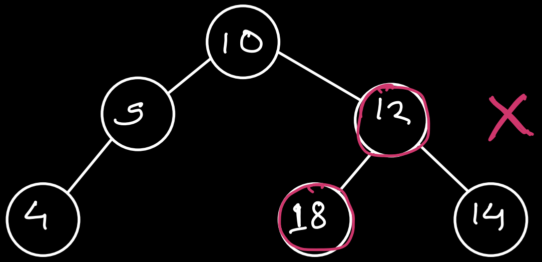
all values of LTS < Root.val < all values of RST









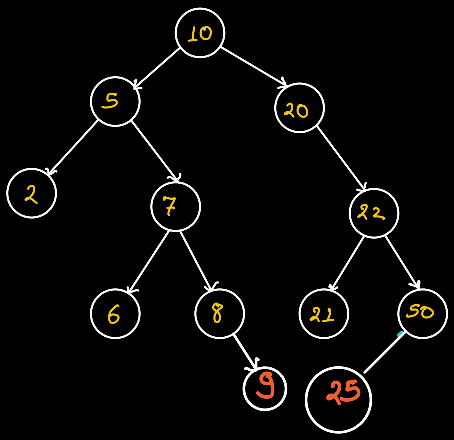


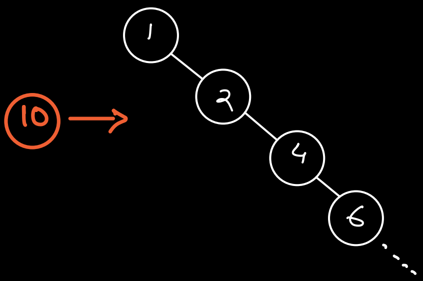
Null

✓

Q Given a BST. Insert a val maintaining the BST property.
(assume no duplicate)

```
TreeNode insert (root, K) {

// Assumption : insert (node, K) => 
insert K at appropriate position in
tree rooted at 'node'
& returns the updated root node.

    if (root == null)
            ret new TreeNode(K);

    if( root.val > K){
          root.left = insert (root.left, K);
    }
    else
          root.right = insert (root.right, K);

    ret root;
}
```





TC : O(N)
SC : O(N)
Skewed Tree

Q Given a BST. Check if a given target is present in it.

```
boolean search (root, K) {

        if (root == null) {
                ret false;
        }

        if( root.val == K) {
                ret true;
        }

        if (root. val > K) {
                ret search (root.left, K);
        }

        else
                ret search (root.right, K);
}
```

TC : O(N)

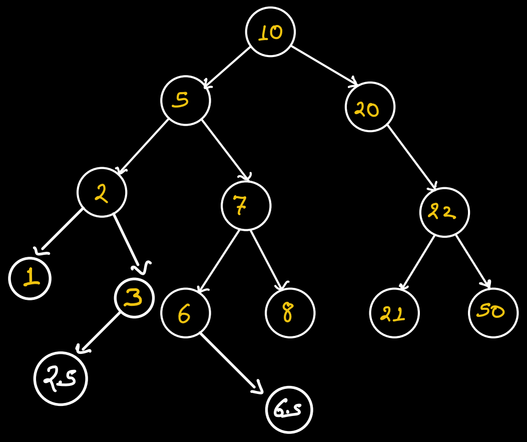
SC : O(N)

Q Given a BST. Delete a value K from it.

Amazon

[No duplicates]

del(5)



Case I

K is present at a leaf node

⇒ make it null

Case II

Node to be deleted has 1 child

⇒ Return the non-null child.

Case III

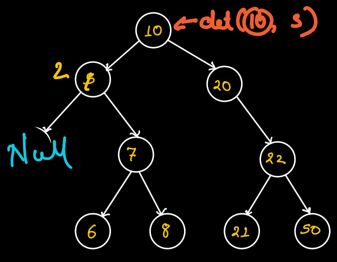
Node to be deleted has both childn.

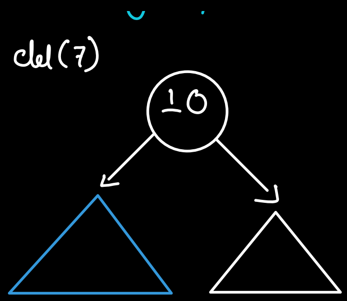
Replace the node by max of LST & delete max of LST

or

by min of RST & delete min of RST

Tree Node





TC : O(N)

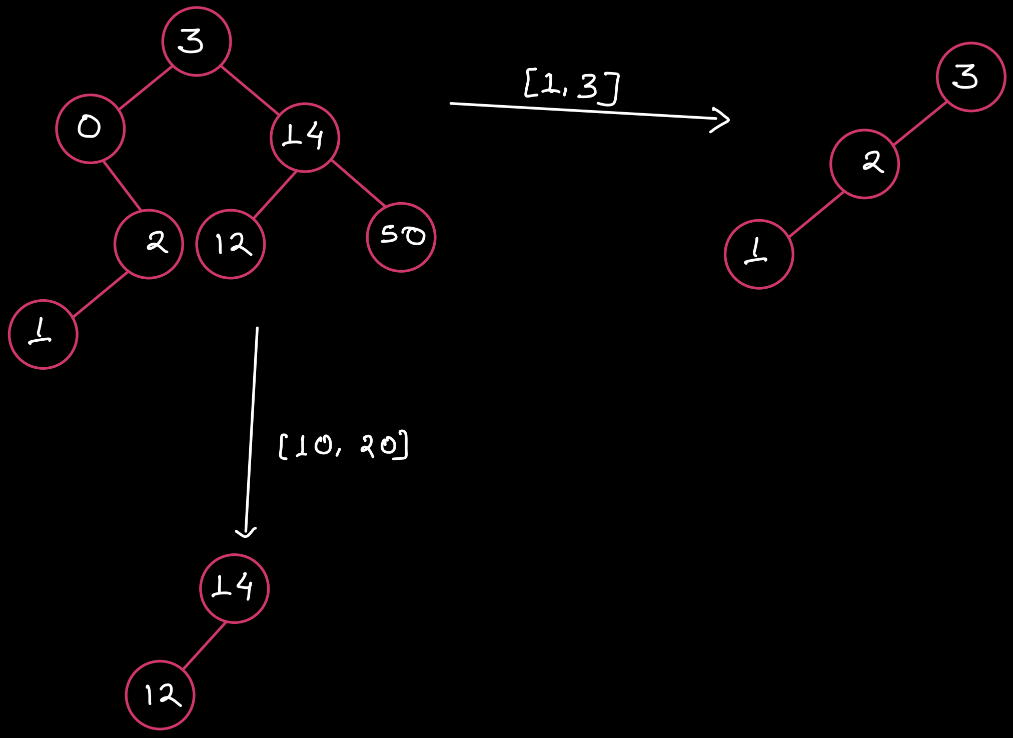
SC : O(N)

HW: Delete without Swapping value

```
deleteNode (root, K) {
  if(root == null) {
        ret null;
  if (root.val > K) {
      root.left = deleteNode (root.left, K);
  }
  else if (root.val < K) {
      root.right = deleteNode (root.right, K);
  }
  else {        // root.val == K

     // Case I : Leaf node
     if ( isLeaf(root) ) {
           ret null;
     }

   // Case II : Root has one child

     if(root.left == null)
            ret root.right;
      if(root.right == null)
            ret root.left ;
  // Case III : Root has both children
    else {
      max = getMax (root.left);
      root.val = max.val;
      root.left = deleteNode (root.left, max.val);
  } }
      ret root;
}
```

Amazon
MS

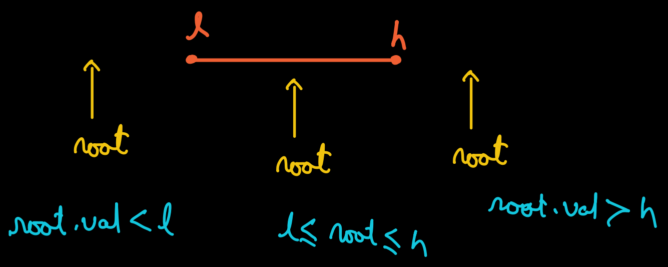
Q Given a BST. Given a range l to h.
Delete every node which has a val outside the range [l , h]

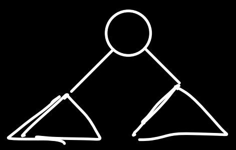


Approach 1

Call deleteNode(root, k) on all node's val outside the range.

$O(N^2)$





3

```
trimBST ( root, l, h ) {
        if (root == null) ret null;
      if ( root.val < l) {
              ret trimBST( root.right, l, h);
      }
    if( root.val > h){
              ret trimBST (root.left, l, h);
    }
    else
    {
      root.left = trimBST (root.left, l, h);
      root.right = trimBST( root.right, l, h);
      ret root;
  }

}
```

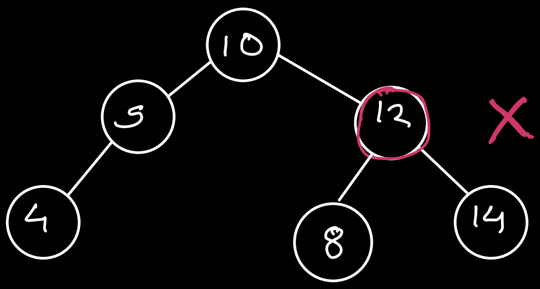
TC : O(N)

SC : O(N)

Q Given a BT. Return true if it is a BST.
[No duplicates]

```
boolean checkBST(root){
    :
    .
}
```

inorder
Preorder
Post order.


10
5
12
4
8
14
X