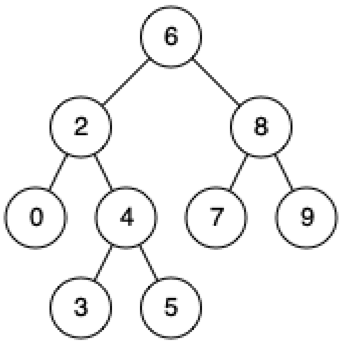
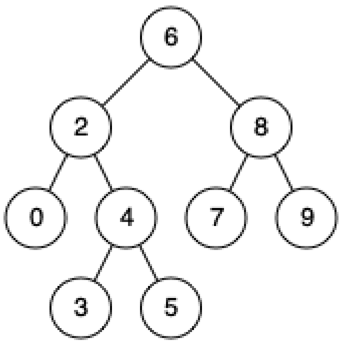
235. Lowest Common Ancestor of a Binary Search Tree

Given a binary search tree (BST), find the lowest common ancestor (LCA) of two given nodes in the BST.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where we allow **a node to be a descendant of itself**).”



Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 8
Output: 6
Explanation: The LCA of nodes 2 and 8 is 6.



Input: root = [6,2,8,0,4,7,9,null,null,3,5], p = 2, q = 4
Output: 2
Explanation: The LCA of nodes 2 and 4 is 2, since a node can be a descendant of itself according to the LCA definition.

```
def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q:
'TreeNode') -> 'TreeNode':
        res   = self.helper(root,p,q)
        return res
```

```python
def helper(self,root,p,q):
    if root is None:
        return
    if root.val>p.val and root.val>q.val:
        return self.helper(root.left,p,q)
    elif root.val<p.val and root.val<q.val:
        return self.helper(root.right,p,q)
    else:
        return root
```

Now, here the property is BST is used. If p and q both are less than root, go to left as we can only find the LCA there. If p and q both are greater than root, go to right as we can only find the LCA there. If the val of root is in between the p<=root.val<=q, then this root is our answer as either going to left or right we wont get both the values.