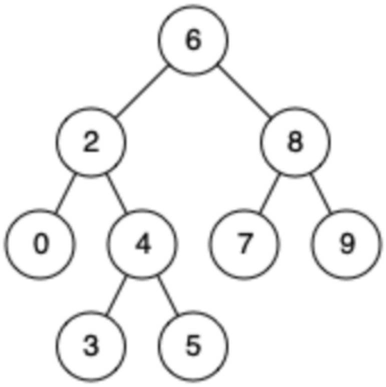
# Lowest Common Ancestor of a Binary Search Tree

| | |
|---|---|
| Difficulty | Medium |
| Category | Tree |
| Question | https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-search-tree/ |
| Solution | https://www.youtube.com/watch?v=gs2LMfuOR9k |
| Status | Done |

## Question

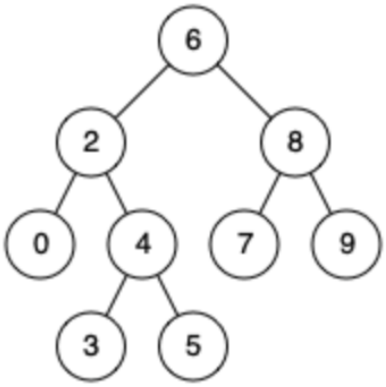
> Given a binary search tree (BST), find the lowest common ancestor (LCA) node of two given nodes in the BST.
>
> According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes `p` and `q` as the lowest node in `T` that has both `p` and `q` as descendants (where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

**Example 1:**



> **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.

**Example 2:**



> **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

## Solution

```
class Solution:
    def lowestCommonAncestor(self, root: 'TreeNode', p: 'TreeNode', q: 'TreeNode') -> 'TreeNode':
        cur = root  # Start the traversal from the root node.

        while cur:
            # If both nodes, p and q, are greater than the current node's value,
            # it means the LCA must be on the right subtree.
            if p.val > cur.val and q.val > cur.val:
                cur = cur.right  # Move to the right subtree.

            # If both nodes, p and q, are smaller than the current node's value,
            # it means the LCA must be on the left subtree.
            elif p.val < cur.val and q.val < cur.val:
                cur = cur.left  # Move to the left subtree.

            # If neither of the above conditions is met, it means one node is on the left
            # and the other is on the right, so the current node is the lowest common ancestor.
            else:
                return cur  # Return the LCA when found.
```

💡 Time Complexity: O(h), where 'h' is the height of the BST. In the worst case, 'h' can be equal to the number of nodes in the tree, making it O(n) for an unbalanced tree.

Space Complexity: O(1) as the algorithm uses a single pointer to traverse the tree without any additional data structures.