Lowest Common Ancestor (LCA) Code Explanation

Problem Statement:

Given a binary tree, find the lowest common ancestor (LCA) of two given nodes p and q.

Definition:

The **Lowest Common Ancestor (LCA)** of two nodes p and q is defined as the lowest node in the tree that has both p and q as descendants (where we allow a node to be a descendant of itself).

Example:

Example 1:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1 **Output:** 3 **Explanation:** The LCA of nodes 5 and 1 is 3.

Example 2:

Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 4 **Output:** 5 **Explanation:** Since a node is also considered as a descendant of itself, the LCA of nodes 5 and 4 is 5.

Example 3:

Input: root = [1,2], p = 1, q = 2 **Output:** 1

Code Explanation

Function: exists(TreeNode root, TreeNode* t)*

This function checks if node t exists in the subtree rooted at root.

Logic:

1. If root is NULL, return false.
2. If root is equal to t, return true.
3. Recursively check in the left and right subtrees.

Code:

```
bool exists(TreeNode* root, TreeNode* t){
   if(root == NULL) return false;
   if(root == t) return true;
   return exists(root->left, t) || exists(root->right, t);
}
```

Function: lowestCommonAncestor(TreeNode root, TreeNode* p, TreeNode* q)*

This function finds the lowest common ancestor (LCA) of p and q.

Logic:

1. **Base Case:** If root is equal to p and q, return root. *(However, this condition never occurs because p != q is guaranteed.)*
2. **Check Different Subtrees:** If p and q exist in different subtrees (left & right), return root as the LCA.
3. **Check If Root is One of the Nodes:** If root is equal to p or q, and the other node exists in the subtree, return root.
4. **Recursive Calls:**
    - If both p and q exist in the right subtree, recursively call lowestCommonAncestor(root->right, p, q).
    - If both exist in the left subtree, call lowestCommonAncestor(root->left, p, q).

Code:

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
   if(root == p && root == q) return root;
   else if(exists(root->right, p) && exists(root->left, q)) return root;
   else if(exists(root->left, p) && exists(root->right, q)) return root;
   else if(root == p && (exists(root->right, q) || exists(root->left, q))) return root;
   else if(root == q && (exists(root->right, p) || exists(root->left, p))) return root;
   else if(exists(root->right, p) && exists(root->right, q))
      return lowestCommonAncestor(root->right, p, q);
   else
      return lowestCommonAncestor(root->left, p, q);
}
```

Complexity Analysis

- **Time Complexity:**
    - exists() function runs in **O(N)** worst case.
    - lowestCommonAncestor() calls exists() multiple times, leading to **O(N^2)** complexity.
- **Space Complexity:**
    - **O(H)** due to recursive calls, where H is the height of the tree.

Key Takeaways:

- **exists() function:** Checks if a node exists in a subtree.
- **lowestCommonAncestor() function:** Recursively determines the lowest common ancestor based on subtree existence.
- **Efficiency:** The solution can be improved to **O(N)** using a different approach.
- **Alternative Approach:** Instead of using exists(), a **single traversal method** can be used to find the LCA in **O(N)** time.

Summary Table:

Case	Condition	Return
root == p && root == q	Impossible case (p != q always)	root
p in left, q in right	Different subtrees	root
q in left, p in right	Different subtrees	root
root == p, q exists in subtree	p is LCA	p
root == q, p exists in subtree	q is LCA	q
Both p and q in right subtree	Recursive call	lowestCommonAncestor(root->right, p, q)
Both p and q in left subtree	Recursive call	lowestCommonAncestor(root->left, p, q)

This explanation should help in understanding the problem statement, code logic, and implementation details better. 🚀