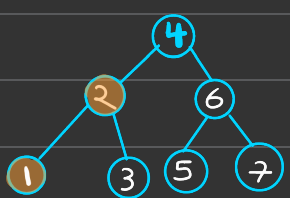
LEETCODE 235

By

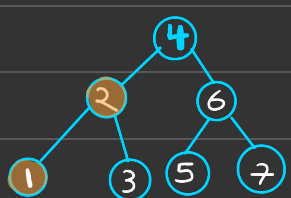
Priyansh

9.) The problem is Asking to find the nearmost or lowest common Ancestor of two nodes of a binary Search tree.

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val == p->val || root -> val == q->val) return root;
        else if(root->val > p->val && root->val < q->val)return root;
        else if(root->val > q->val && root->val< p->val)return root;
        else if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else return lowestCommonAncestor(root->right,p,q);
    }
};
```

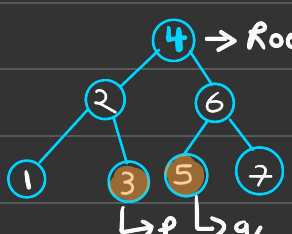


root= 2 P=2 q=1



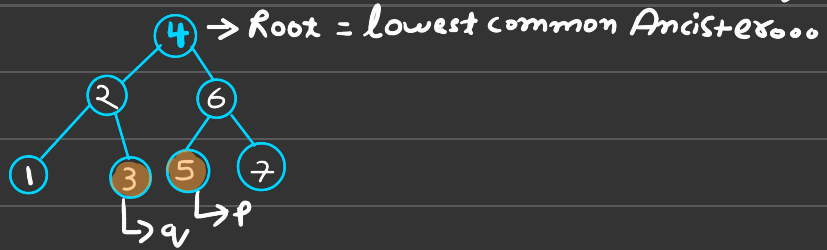
root= 2 P=1 q=2

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val == p->val || root -> val == q->val) return root;
        else if(root->val > p->val && root->val < q->val)return root;
        else if(root->val > q->val && root->val< p->val)return root;
        else if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else return lowestCommonAncestor(root->right,p,q);
    }
};
```

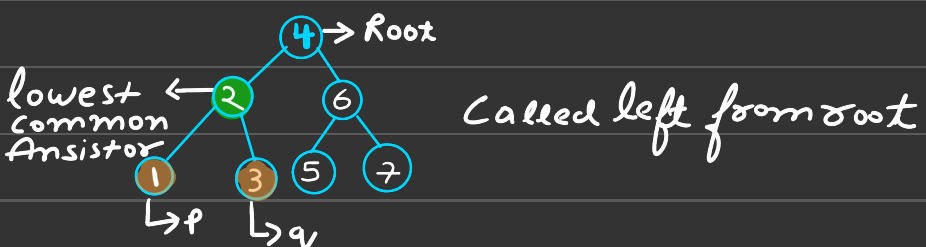


4 → Root = lowest common Ancister...

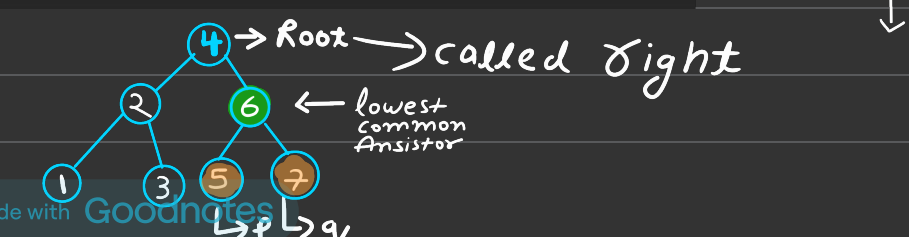
```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val == p->val || root -> val == q->val) return root;
        else if(root->val > p->val && root->val < q->val)return root;
        else if(root->val > q->val && root->val< p->val)return root;
        else if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else return lowestCommonAncestor(root->right,p,q);
    }
};
```




```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val == p->val || root -> val == q->val) return root;
        else if(root->val > p->val && root->val < q->val)return root;
        else if(root->val > q->val && root->val< p->val)return root;
        else if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else return lowestCommonAncestor(root->right,p,q);
    }
};
```




```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val == p->val || root -> val == q->val) return root;
        else if(root->val > p->val && root->val < q->val)return root;
        else if(root->val > q->val && root->val< p->val)return root;
        else if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else return lowestCommonAncestor(root->right,p,q);
    }
};
```



New Code Same Task:-

(In order to remove some unnecessary check list)...

```
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if(root->val > q->val && root->val > p->val) return lowestCommonAncestor(root->left,p,q);
        else if(root->val < p->val && root->val < q->val) return lowestCommonAncestor(root->right,p,q);
        else return root;
    }
};
```