Insert a node into BST:

```
class Solution {
public:
    typedef TreeNode Node;
    TreeNode* insertIntoBST(TreeNode* root, int val) {
        if(root==NULL){
            return new Node(val);
        }
        Node*prev=NULL;
        Node*curr=root;
        while(curr!=NULL){
            if(curr->val>=val){
                prev=curr;
                curr=curr->left;
            }
            else if(curr->val<val){
                prev=curr;
                curr=curr->right;
            }
        }
        if(prev->val>val){
            prev->left=new Node(val);
        }
        else{
            prev->right=new Node(val);
        }
        return root;
    }
};
```

Time:O(Height)
Space:O(1)

Balance a BST:

```
class Solution {
public:

    typedef TreeNode Node;
    void inorderTraversal(Node*root,vector<int>&vec){
        if(root==NULL){
            return;
        }
        inorderTraversal(root->left,vec);
        vec.push_back(root->val);
        inorderTraversal(root->right,vec);
    }
    Node* getABST(int lo,int hi,vector<int>&vec){
        //base case
        if(lo>hi){
            return NULL;
        }
        // if(lo==hi){
        //     return new Node(vec[lo]);
        // }

        int mid=(lo+hi)/2;
        Node*root=new Node(vec[mid]);
        Node*left=getABST(lo,mid-1,vec);
        Node*right=getABST(mid+1,hi,vec);
        root->left=left;
        root->right=right;
        return root;
    }
    TreeNode* balanceBST(TreeNode* root) {
        vector<int>vec;
        inorderTraversal(root,vec);
        int lo=0;
        int hi=vec.size()-1;
        Node*newRoot=getABST(lo,hi,vec);
        return newRoot;

    }
};
```

Time:O(N)
Space:O(n)

Lowest Common Ancestor in a BST:

```
class Solution {
public:

    typedef TreeNode Node;
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode*
q) {
        if(root==NULL){
            return NULL;
        }
        Node*curr=root;
        while(curr!=NULL){
            if(p->val<curr->val&&q->val<curr->val){
                curr=curr->left;
            }
            else if(p->val>curr->val&&q->val>curr->val){
                curr=curr->right;
            }
            else{
                return curr;
            }
        }
        return root;
    }
};
```

Time:O(max Height of BST)
Space:O(1)