```cpp
#include<bits/stdc++.h>
#include<algorithm>
using namespace std;
class Node{
    public:
        int data;
        Node *left;
        Node *right;

        Node(int val){
            data = val;
            left = right = NULL;
        }
};

class LNode{
    public:
        int data;
        LNode* next;

        LNode(int val){
            data = val;
            next = NULL;
        }
};

//check for balanced tree
int height(Node* root, bool &valid){
    if(root==NULL){
        return 0;
    }
    int left = height(root->left, valid);
    int right = height(root->right, valid);
    if(abs(left-right)>1){
        valid=false;
    }
    return 1+max(left,right);
}

bool isBalanced(Node* root){
    bool valid = true;
    height(root,valid);
    return valid;
}




//level order traversal in spiral form
void levelOrder(Node* root, vector<int> &ans){
    if(root==NULL){
        return;
    }
```

```
    stack<Node*> LR; //stores ele who will come in the path while traversing from Left
to Right
    stack<Node*> RL;
    RL.push(root);
    while(!LR.empty() || !RL.empty()){

        if(!RL.empty()){
            while(!RL.empty()){
                Node* temp = RL.top();
                ans.push_back(temp->data);
                RL.pop();
                if(temp->right){
                    LR.push(temp->right);
                }
                if(temp->left){
                    LR.push(temp->left);
                }
            }
        }
        else{
            while(!LR.empty()){
                Node* temp = LR.top();
                ans.push_back(temp->data);
                LR.pop();
                if(temp->left){
                    RL.push(temp->left);
                }
                if(temp->right){
                    RL.push(temp->right);
                }
            }
        }
    }
}



//left view of bst
void leftView(Node* root, vector<int> &v){
    queue<Node*> q;
    q.push(root);
    while(!q.empty()){
        int n = q.size();
        v.push_back(q.front()->data);
        while(n--){
            Node* temp = q.front();
            q.pop();
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
```

```
            }
        }
    }




    // Max Path Sum between 2 special Node
    int maxPathSum(Node* root, int &sum){
        if(root==NULL){
            return 0;
        }

        if(!root->left and !root->right){
            return root->data;
        }

        int left = maxPathSum(root->left,sum);
        int right = maxPathSum(root->right,sum);

        if(root->left and root->right){
            sum = max(sum,left+right+root->data);
            return root->data + max(left, right);
        }

        if(root->left){
            return root->data + left;
        }

        if(root->right){
            return root->data + right;
        }

        return 0;
    }


    //Lowest Common Ansestor
    Node* LCA(Node* root, int p, int q){
        if(root==NULL){
            return NULL;
        }
        // if(root->data == p || root->data == q){//one
        //     return root;
        // }
        if(root->data>p and root->data>q){
            return LCA(root->left,p,q);
        }
        if(root->data<p and root->data<q){
            return LCA(root->right,p,q);
        }
        //two
            return root;
```

```
    //one or two both cases are same
}



//Fix nodes in bst
void inOrder(Node* root, vector<int> &v){
    if(root == NULL){
        return;
    }
    inOrder(root->left,v);
    v.push_back(root->data);
    inOrder(root->right,v);
}

void swap(int n1, int n2, vector<int> &ans){
    int temp = ans[n1];
    ans[n1] = ans[n2];
    ans[n2] = temp;
}


vector<int> fixNodes(Node* root) {
    vector<int> ans;
    inOrder(root, ans);
    int first = -1, second = -1;
    for(int i=1; i<ans.size(); i++){
        if(ans[i-1]>ans[i]){
            if(first==-1 and second==-1){
                first = i-1;
                second = i;
            }
            else{
                second = i;
            }
        }
    }
    swap(first,second,ans);
    return ans;
}




// merge 2 bst
void mergeBST(Node* root1, Node* root2, vector<int> &ans){
    vector<int> first;
    vector<int> second;
    inOrder(root1,first);
    inOrder(root2, second);
    int i = 0, j=0;
    while(i<first.size() and j<second.size()){
        if(first[i]>second[j]){
            ans.push_back(second[j++]);
```

```
            ans.push_back(first[i++]);
        }
        else{
            ans.push_back(first[i++]);
            ans.push_back(second[j++]);
        }
    }
    while(i<first.size()){
        ans.push_back(first[i++]);
    }
    while(j<second.size()){
        ans.push_back(second[j++]);
    }
}


int main(){
    Node *root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->left = new Node(6);
    root->right->right = new Node(7);
    root->left->left->left = new Node(8);
    root->left->left->right = new Node(9);

    bool ans = isBalanced(root);
    ans ? cout<<"balanced" : cout<<"UnBalanced";

    vector<int> ans;
    levelOrder(root,ans);
    for(auto ele: ans){
        cout<<ele<<" ";
    }

    vector<int> ans;
    leftView(root,ans);
    for(auto ele: ans){
        cout<<ele<<" ";
    }

    int sum = INT_MIN;
    int temp = maxPathSum(root, sum);

    if (root->left && root->right)
        cout<<sum;
    else{
        int ans =  max(sum, temp);
        cout<<ans;
    }


    create a bst
```

```
    Node* root1 = new Node(5);
    root1->left = new Node(3);
    root1->right = new Node(8);
    root1->left->left = new Node(2);
    root1->left->right = new Node(4);
    root1->left->left->left = new Node(1);
    root1->right->right = new Node(9);
    root1->right->right->left = new Node(7);
    root1->right->right->left->left = new Node(6);

    Node* ans = LCA(root1,2,4);
    if (ans)
    cout << "LCA is: " << ans->data << endl;
    else
    cout << "LCA not found (one or both nodes not in tree)." << endl;




    Node* root2 = new Node(10);
    root2->left = new Node(5);
    root2->right = new Node(8);
    root2->left->left = new Node(2);
    root2->left->right = new Node(20);
    vector<int> ans = fixNodes(root2);
    cout << "Fixed nodes in BST: ";
    for (auto ele : ans) {
        cout << ele << " ";
    }



    Node* root1 = new Node(10);
    root1->left = new Node(5);
    root1->right = new Node(20);
    root1->left->left = new Node(2);
    root1->left->right = new Node(8);

    Node* root2 = new Node(10);
    root2->left = new Node(5);
    root2->right = new Node(20);
    root2->left->left = new Node(2);
    root2->left->right = new Node(8);

    vector<int> ans;
    mergeBST(root1,root2,ans);
    for(auto ele: ans){
        cout<<ele<<" ";
    }

    return 0;
}
```

```cpp
//// Convert Sorted Linked List to Balanced BST
#include<bits/stdc++.h>
using namespace std;
class LNode{
    public:
        int data;
        LNode* next;

        LNode(int val){
            data = val;
            next = NULL;
        }
};

class Node{
    public:
        int data;
        Node *left;
        Node *right;

        Node(int val){
            data = val;
            left = right = NULL;
        }
};

void inOrder(Node* root){
    if(root == NULL){
        return;
    }
    inOrder(root->left);
    cout<<root->data<<" ";
    inOrder(root->right);
}

Node* buildBST(int s, int e, vector<int> ans){
    if(s>e){
        return NULL;
    }
    int mid = (s+e)/2;
    Node* root = new Node(ans[mid]);
    root->left = buildBST(s,mid-1,ans);
    root->right = buildBST(mid+1,e,ans);
    return root;
}

Node* sortedListToBST(LNode* head){
    vector<int> ans;
    while(head){
        ans.push_back(head->data);
        head = head->next;
```

```
        }
        return buildBST(0,ans.size()-1,ans);
    }

    int main(){
        LNode* head = new LNode(1);
        head->next = new LNode(2);
        head->next->next = new LNode(3);
        head->next->next->next = new LNode(4);
        head->next->next->next->next = new LNode(5);
        head->next->next->next->next->next = new LNode(6);
        head->next->next->next->next->next->next = new LNode(7);

        Node* ans = sortedListToBST(head);

        inOrder(ans);
        return 0;
    }
```