PW SKILLS

# Assignments

## Binary Tree

PW SKILLS

**1. Product of all nodes in a Binary Tree**

**Solution:**

```
#include <iostream>
using namespace std;
class Node{ // This is a TreeNode
public:
int val;
Node* left;
Node* right;
Node(int val){
this→val = val;
this→left = NULL;
this→right = NULL;
}
};
int product(Node* root){
if(root==NULL) return 1;
return root→val * product(root→left) * product(root→right);
}

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→right→left→right = new Node(8);
int prod = product(root);
cout << "Product of all the nodes is: " << prod << endl;
return 0;
}
```

**2. Find the minimum value in a Binary tree**

**Solution:**

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
class Node {
public:
int data;
Node *left, *right;
Node(int data){
this→data = data;
this→left = NULL;
this→right = NULL;
}
```

```
};
int findMin(Node* root){
if (root == NULL) return INT_MAX;
int res = root->data;
int lres = findMin(root->left);
int rres = findMin(root->right);
if (lres < res) res = lres;
if (rres < res) res = rres;
return res;
}

int main(){
Node* NewRoot = NULL;
Node* root = new Node(2);
root->left = new Node(7);
root->right = new Node(5);
root->left->right = new Node(6);
root->left->right->left = new Node(1);
root->left->right->right = new Node(11);
root->right->right = new Node(9);
root->right->right->left = new Node(4);
cout << "Minimum element is " << findMin(root) << endl;
return 0;
}
```

**3. Balanced Binary Tree**

**Solution:**

```
class Solution {
public:
int levels(TreeNode* root){
if(root==NULL) return 0;
return 1 + max(levels(root->left),levels(root->right));
}
bool isBalanced(TreeNode* root) {
if(root==NULL) return true;
int left = levels(root->left);
int right = levels(root->right);
int diff = abs(left - right);
if(diff>1) return false;
bool leftTreeAns = isBalanced(root->left);
if(leftTreeAns==false) return false;
bool rightTreeAns = isBalanced(root->right);
if(rightTreeAns==false) return false;
return true;
}
};
```

**4. Symmetric Tree**

**Solution:**

```
class Solution {
public:
bool isSameTree(TreeNode* p, TreeNode* q) {
if(p==NULL && q==NULL) return true;
if(p==NULL || q==NULL) return false;
if(p->val != q->val) return false;
return isSameTree(p->left, q->left) && isSameTree(p->right, q
->right);
}
TreeNode* invertTree(TreeNode* root) {
if(root==NULL) return root;
TreeNode* temp = root->left;
root->left = root->right; root->right = temp;
invertTree(root->left); invertTree(root->right);
return root;
}
bool isSymmetric(TreeNode* root) {
if(root==NULL) return true;
invertTree(root->left);
bool flag = isSameTree(root->left,root->right);
invertTree(root->left);
return flag;
}
};
```