DAY-6 WWC

NAME: Siddharth Lal

UID: 22BCS13410

Problem-1(VeryEasy)

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

struct TreeNode {
   int val;
   TreeNode* left;
   TreeNode* right;
   TreeNode() : val(0), left(nullptr), right(nullptr) {}
   TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
   TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
   vector<int> inorderTraversal(TreeNode* root) {
      vector<int> result;
      inorderHelper(root, result);
      return result;
   }

private:
   void inorderHelper(TreeNode* node, vector<int>& result) {
      if (node == nullptr) {
         return;
      }
      inorderHelper(node->left, result);
      result.push_back(node->val);
      inorderHelper(node->right, result);
   }
};
int main() {
```

```
    TreeNode* root1 = new TreeNode(1, nullptr, new TreeNode(2, new TreeNode(3),
nullptr));
    Solution solution;
    vector<int> result1 = solution.inorderTraversal(root1);
    for (int val : result1) {
        cout << val << " ";
    }
    cout << endl;

    TreeNode* root2 = new TreeNode(1,
                new TreeNode(2, new TreeNode(4), new TreeNode(5, new TreeNode(6),
new TreeNode(7))),
                new TreeNode(3, nullptr, new TreeNode(8, new TreeNode(9), nullptr)));
    vector<int> result2 = solution.inorderTraversal(root2);
    for (int val : result2) {
        cout << val << " ";
    }
    cout << endl;

    return 0;
}
```

OUTPUT:

```
1 3 2
4 2 6 5 7 1 3 9 8


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem-2(Easy)

CODE:

```
#include <iostream>


using namespace std;


struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};


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

```
};

int main() {
    TreeNode* p1 = new TreeNode(1, new TreeNode(2), new TreeNode(3));
    TreeNode* q1 = new TreeNode(1, new TreeNode(2), new TreeNode(3));
    Solution solution;
    cout << (solution.isSameTree(p1, q1) ? "true" : "false") << endl;

    TreeNode* p2 = new TreeNode(1, new TreeNode(2), nullptr);
    TreeNode* q2 = new TreeNode(1, nullptr, new TreeNode(2));
    cout << (solution.isSameTree(p2, q2) ? "true" : "false") << endl;

    return 0;
}
```

OUTPUT:

```
true
false


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem-3(Medium)

CODE:

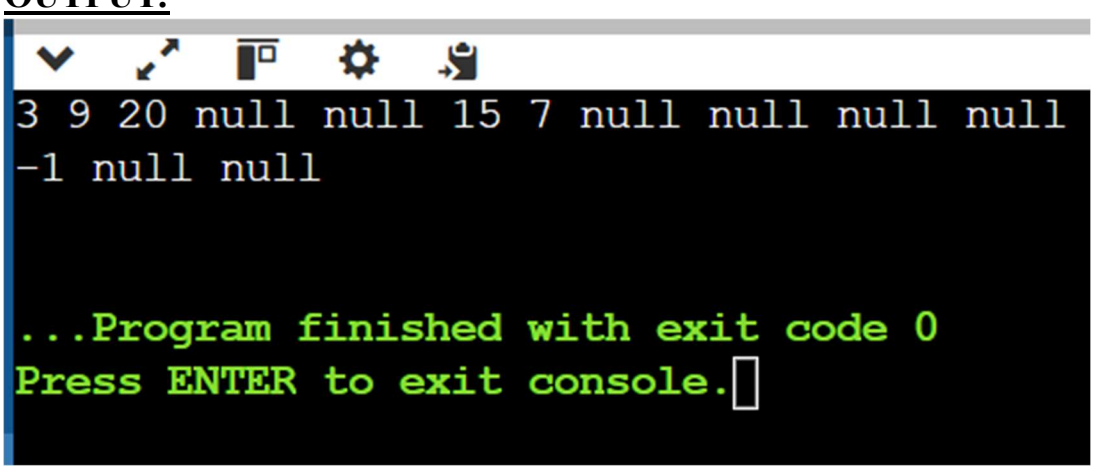
```
#include <iostream>
#include <vector>
#include <unordered_map>

using namespace std;
```

```
struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    TreeNode* buildTree(vector<int>& preorder, vector<int>& inorder) {
        unordered_map<int, int> inorderIndexMap;
        for (int i = 0; i < inorder.size(); i++) {
            inorderIndexMap[inorder[i]] = i;
        }
        int preorderIndex = 0;
        return build(preorder, inorderIndexMap, preorderIndex, 0, inorder.size() - 1);
    }

private:
    TreeNode* build(vector<int>& preorder, unordered_map<int, int>&
inorderIndexMap,
                    int& preorderIndex, int inorderStart, int inorderEnd) {
        if (inorderStart > inorderEnd) {
            return nullptr;
```

```
        }

        int rootVal = preorder[preorderIndex++];
        TreeNode* root = new TreeNode(rootVal);

        int inorderIndex = inorderIndexMap[rootVal];

        root->left = build(preorder, inorderIndexMap, preorderIndex, inorderStart,
inorderIndex - 1);
        root->right = build(preorder, inorderIndexMap, preorderIndex, inorderIndex + 1,
inorderEnd);

        return root;
    }
};
void printLevelOrder(TreeNode* root) {
    if (!root) {
        return;
    }
    vector<TreeNode*> currentLevel = {root};
    while (!currentLevel.empty()) {
        vector<TreeNode*> nextLevel;
        for (TreeNode* node : currentLevel) {
            if (node) {
                cout << node->val << " ";
                nextLevel.push_back(node->left);
                nextLevel.push_back(node->right);
            } else {
```

```
            cout << "null ";
        }
    }
    currentLevel = nextLevel;
  }
  cout << endl;
}
int main() {
  vector<int> preorder1 = {3, 9, 20, 15, 7};
  vector<int> inorder1 = {9, 3, 15, 20, 7};
  Solution solution;
  TreeNode* root1 = solution.buildTree(preorder1, inorder1);
  printLevelOrder(root1);
  vector<int> preorder2 = {-1};
  vector<int> inorder2 = {-1};
  TreeNode* root2 = solution.buildTree(preorder2, inorder2);
  printLevelOrder(root2);
  return 0;
}
```

OUTPUT:



Problem-4(Hard)

CODE:

```
#include <iostream>
#include <algorithm>
#include <climits>

using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};

class Solution {
public:
    int maxPathSum(TreeNode* root) {
        int globalMax = INT_MIN;
        maxGain(root, globalMax);
        return globalMax;
    }

private:
    int maxGain(TreeNode* node, int& globalMax) {
        if (node == nullptr) {
            return 0;
        }

        int leftGain = max(maxGain(node->left, globalMax), 0);
        int rightGain = max(maxGain(node->right, globalMax), 0);

        int currentPathSum = node->val + leftGain + rightGain;

        globalMax = max(globalMax, currentPathSum);

        return node->val + max(leftGain, rightGain);
    }
};

int main() {
```

```
    TreeNode* root1 = new TreeNode(1, new TreeNode(2), new TreeNode(3));
    Solution solution;
    cout << "Maximum Path Sum: " << solution.maxPathSum(root1) << endl;

    TreeNode* root2 = new TreeNode(-10,
                                   new TreeNode(9),
                                   new TreeNode(20, new TreeNode(15), new TreeNode(7)));
    cout << "Maximum Path Sum: " << solution.maxPathSum(root2) << endl;

    return 0;
}
```

OUTPUT:

```
Maximum Path Sum: 6
Maximum Path Sum: 42


...Program finished with exit code 0
Press ENTER to exit console.
```

Problem-5(VeryHard)

CODE:

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

struct TreeNode {
    int val;
    TreeNode* left;
    TreeNode* right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode* left, TreeNode* right) : val(x), left(left), right(right) {}
};
```

```
class Solution {
public:
    vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
        vector<vector<int>> result;
        if (!root) return result;

        queue<TreeNode*> q;
        q.push(root);
        bool leftToRight = true;

        while (!q.empty()) {
            int levelSize = q.size();
            vector<int> level(levelSize);

            for (int i = 0; i < levelSize; ++i) {
                TreeNode* node = q.front();
                q.pop();

                int index = leftToRight ? i : (levelSize - 1 - i);
                level[index] = node->val;

                if (node->left) q.push(node->left);
                if (node->right) q.push(node->right);
            }

            leftToRight = !leftToRight;
            result.push_back(level);
        }

        return result;
    }
};

int main() {
    Solution solution;

    TreeNode* root1 = new TreeNode(3);
    root1->left = new TreeNode(9);
    root1->right = new TreeNode(20);
```

```
    root1->right->left = new TreeNode(15);
    root1->right->right = new TreeNode(7);

    vector<vector<int>> result1 = solution.zigzagLevelOrder(root1);
    for (const auto& level : result1) {
        for (int val : level) {
            cout << val << " ";
        }
        cout << endl;
    }

    return 0;
}
```

OUTPUT: