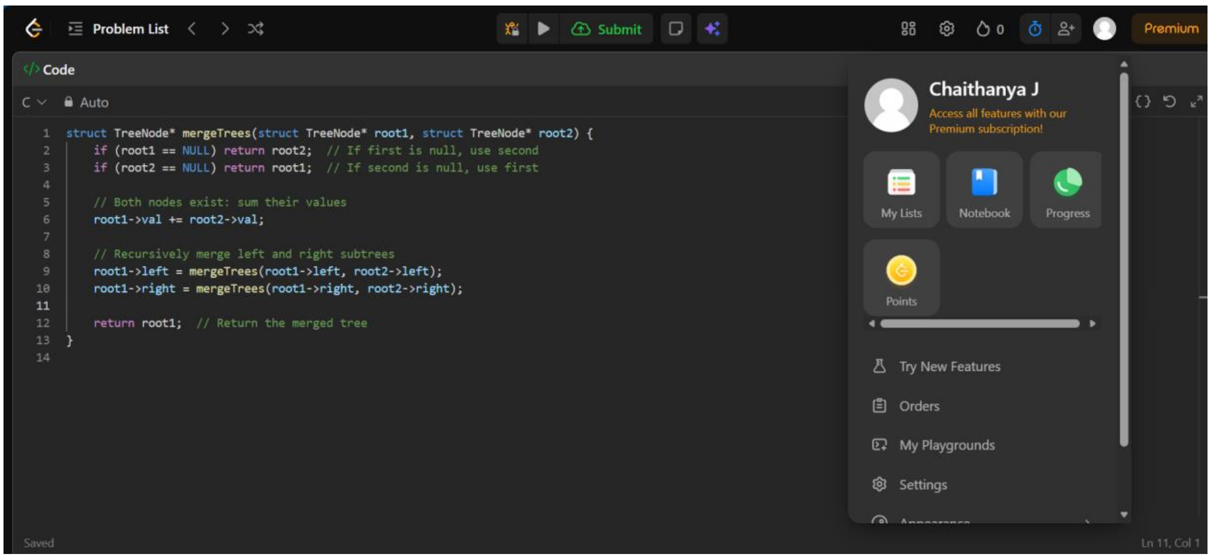
Leetcode:

```
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if (root1 == NULL) return root2;  // If first is null, use second
    if (root2 == NULL) return root1;  // If second is null, use first

    // Both nodes exist: sum their values
    root1->val += root2->val;

    // Recursively merge left and right subtrees
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);

    return root1;  // Return the merged tree
}
```



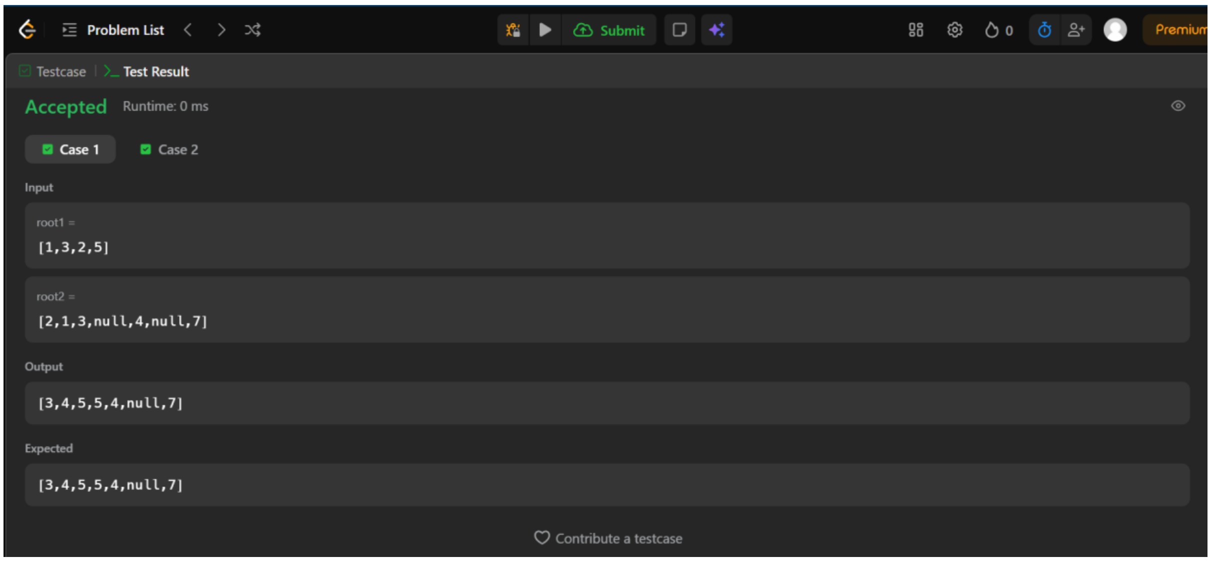
```
struct TreeNode* mergeTrees(struct TreeNode* root1, struct TreeNode* root2) {
    if (root1 == NULL) return root2;  // If first is null, use second
    if (root2 == NULL) return root1;  // If second is null, use first

    // Both nodes exist: sum their values
    root1->val += root2->val;

    // Recursively merge left and right subtrees
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);

    return root1;  // Return the merged tree
}
```

OUTPUT:



Accepted Runtime: 0 ms

Case 1 Case 2

Input

root1 =

```
[1,3,2,5]
```

root2 =

```
[2,1,3,null,4,null,7]
```

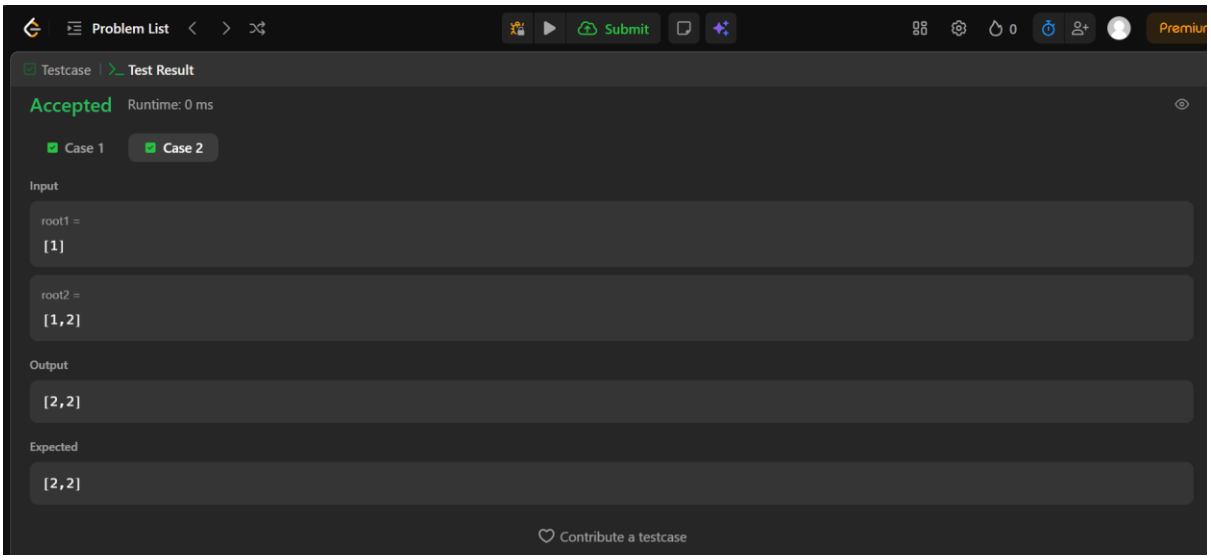
Output

```
[3,4,5,5,4,null,7]
```

Expected

```
[3,4,5,5,4,null,7]
```

Contribute a testcase



Accepted Runtime: 0 ms

Case 1 Case 2

Input

root1 =

```
[1]
```

root2 =

```
[1,2]
```

Output

```
[2,2]
```

Expected

```
[2,2]
```

Contribute a testcase

OBSERVATION:

21/12/25 Prg - 8b.

Leetcode :-

Merge two Binary Trees.

Code:-

```
struct TreeNode* mergeTrees(struct TreeNode*
                         root1, struct TreeNode* root2)
{
    if (root1 == NULL) return root2;
    if (root2 == NULL) return root1;

    root1->val += root2->val;
    root1->left = mergeTrees(root1->left, root2->left);
    root1->right = mergeTrees(root1->right, root2->right);

    return root1;
}
```

Output of Case 1 :-

Input => root 1 = [1, 3, 2, 5]
root 2 = [2, 1, 3, null, 4, null, 7]

Ouput => [3, 4, 5, 5, 4, null, 7]

Case 2 :-

Input :- root 1 = [1]   root 2 = [1, 2]
Ouput : [2, 2].