INTERNSHALA

DATA STRUCTURE AND ALGORITHM

ASSIGNIMENT- 5

Problem 1

The school cafeteria offers circular and square sandwiches at lunch break, referred to by numbers 0 and 1 respectively. All students stand in a queue. Each student either prefers square or circular sandwichesThe number of sandwiches in the cafeteria is equal to the number of students. The sandwiches are placed in a stack. At each step:If the student at the front of the queue prefers the sandwich on the top of the stack, they will take it and leave the queue.Otherwise, they will leave it and go to the queue's end.This continues until none of the queue students want to take the top sandwich and are thus unable to eat.

You are given two integer arrays students and sandwiches where sandwiches[i] is the type of the ith sandwich in the stack (i = 0 is the top of the stack) and students[j] is the preference of the jth student in the initial queue (j = 0 is the front of the queue). Return the number of students that are unable to eat.

Example 1:

Input: students = [1,1,0,0], sandwiches = [0,1,0,1]

Output: 0

Question link

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/description/

Code

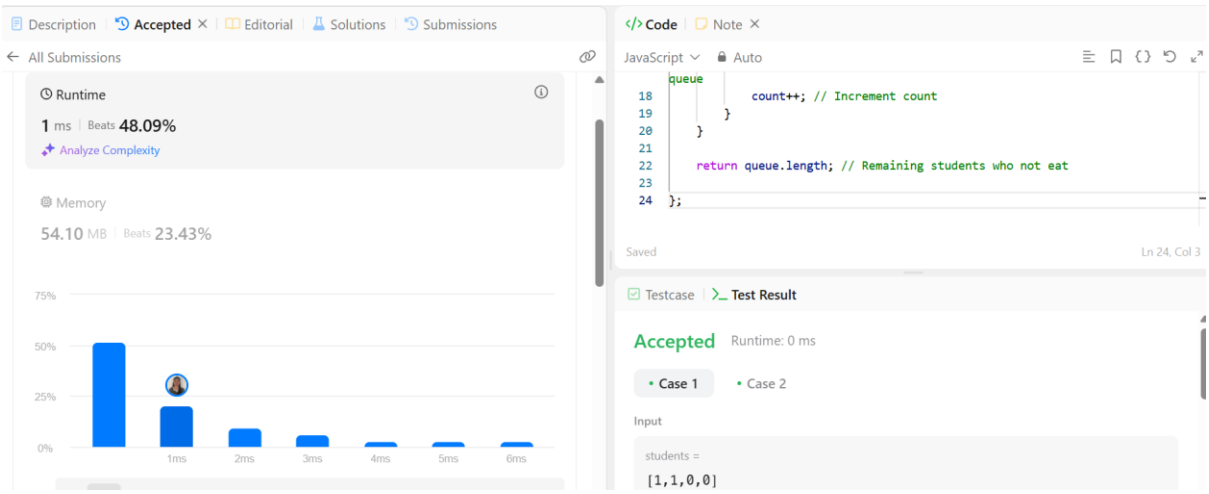
```
function countStudents(students, sandwiches) {
    let queue = [...students];
    let i = 0;
    let count = 0;

    while (queue.length > 0 && count < queue.length) {
        if (queue[0] === sandwiches[i]) {
            queue.shift();
            i++;
            count = 0;
        } else {
            queue.push(queue.shift());
            count++;
        }
    }

    return queue.length;

}
console.log("Number of students who couldn't eat:",countStudents([1,1,0,0],
[0,1,0,1]));
```

```javascript
function countStudents(students, sandwiches) {
    let queue = [...students];
    let i = 0;
    let count = 0;

    while (queue.length > 0 && count < queue.length) {
        if (queue[0] === sandwiches[i]) {
            queue.shift();
            i++;
            count = 0;
        } else {
            queue.push(queue.shift());
            count++;
        }
```

```
OUTPUT ··· Filter Code
[Done] exited with code=0 in 0.177 seconds

[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-5\problem1.js"
Number of students who couldn't eat: 0
```

Submission link

https://leetcode.com/problems/number-of-students-unable-to-eat-lunch/submissions/1583175960/



Conclusion

Time complexity $O(n^2)$

- In worst case , if students continuously rotating in the line goes worst case

Space complexity O(n)

- Using of queue algorithm takes extra space

Problem 2

Minimum depth of Binary tree

Given a binary tree, find its minimum depth .The minimum depth is the number of nodes along the shortest path from the root node down to the nearest leaf node.

Note: A leaf is a node with no children.

Example

Input: root = [3,9,20,null,null,15,7]

Output: 2

Question link

https://leetcode.com/problems/minimum-depth-of-binary-tree/description/

Code

```
class TreeNode {
    constructor(data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}
let root=new TreeNode(3);
root.left=new TreeNode(9);
```

```
root.right=new TreeNode(20);
root.right.left=new TreeNode(15);
root.right.right=new TreeNode(7);
function minDepth(root) {
    if (root == null) {
        return 0;
    }
    let leftSide = minDepth(root.left);
    let rightSide = minDepth(root.right);

    if (root.left == null || root.right == null) {
        return Math.max(leftSide, rightSide) + 1;
    }
    return Math.min(leftSide, rightSide) + 1;
}
let result=minDepth(root);
console.log("Minimum depth of tree is ",result);
```

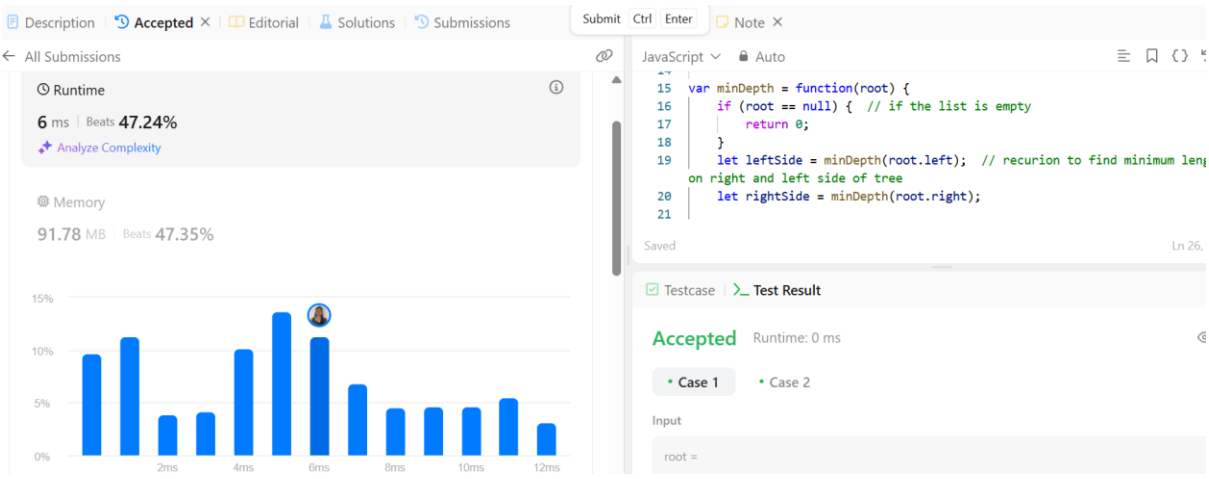
```
DSA assign-5 > JS problem2.js > ...
```

```
1
2
3   class TreeNode {
4       constructor(data) {
5           this.data = data;
6           this.left = null;
7           this.right = null;
8       }
9   }
10  let root=new TreeNode(3);
11  root.left=new TreeNode(9);
12  root.right=new TreeNode(20);
13  root.right.left=new TreeNode(15);
14  root.right.right=new TreeNode(7);
```

```
OUTPUT ··· Filter Code
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-5\problem2.js"
Minimum depth of tree is  2

[Done] exited with code=0 in 0.154 seconds
```

Submission link

https://leetcode.com/problems/minimum-depth-of-binary-tree/submissions/1582938323/



Conclusion

Time complexity O(n)

- It makes a recursive call for both left and right sub-trees
- The function traverses each node exactly once – n nodes

Space complexity

- O(n)- for skewed tree(unbalanced tree)
- O(logn)- for best case (balanced tree) .this tree is balanced

Problem 3

Binary tree post order traversal

Given the root of a binary tree, return the postorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [3,2,1]

Question link

https://leetcode.com/problems/binary-tree-postorder-traversal/description/

Code

```
 function postOrder(root,ans){
    if(root==null){
        return;
    }
    postOrder(root.left,ans);
    postOrder(root.right,ans);
    ans.push(root.val);
 }
var postorderTraversal = function(root) {
    let ans=[];
    postOrder(root,ans);
    return ans;
};
```

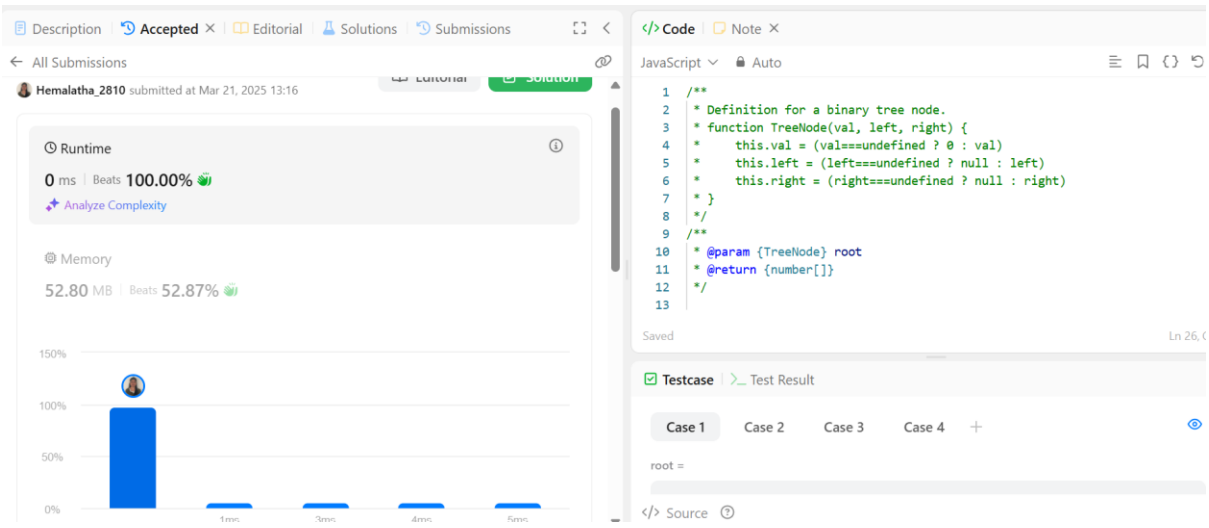
```
3   class TreeNode{
4       constructor(data){
6           this.left=null;
7           this.right=null;
8       }
9   }
10  let root=new TreeNode(1);
11  root.left=new TreeNode(null);
12  root.right=new TreeNode(2);
13  root.right.left=new TreeNode(3);
14
15  function postOrder(root){
16      if(root==null){
17          return ;
```

OUTPUT ··· Filter Code

```
[Running] node "c:\Users\Sangeetha\js_intro\DSA assign-5\problem3.js"
null
3
2
1
```

Submission link

https://leetcode.com/problems/binary-tree-postorder-traversal/submissions/1581112616/



Conclusion

Time complexity O(N)

The function recursively calls itself for the left and right child of each node. N is the number of nodes in the tree.

Space complexity

- Worst case (skewed tree): O(N) (recursion depth + result array).
- Best case (balanced tree): O(log N) (recursion depth) + O(N) for ans , which still simplifies to O(N).

Problem 4

Binary-tree-preorder-traversal

Given the root of a binary tree, return the preorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,2,3]

Question link

https://leetcode.com/problems/binary-tree-preorder-traversal/description/

Code

```
 function preOrder(root,ans){
    if(root==null){
        return;
    }
    ans.push(root.val);
    preOrder(root.left,ans);
    preOrder(root.right,ans);
 }
var preorderTraversal = function(root) {
    let ans=[];
    preOrder(root,ans);
    return ans;
};
```

DSA assign-5 > JS problem4.js > ...

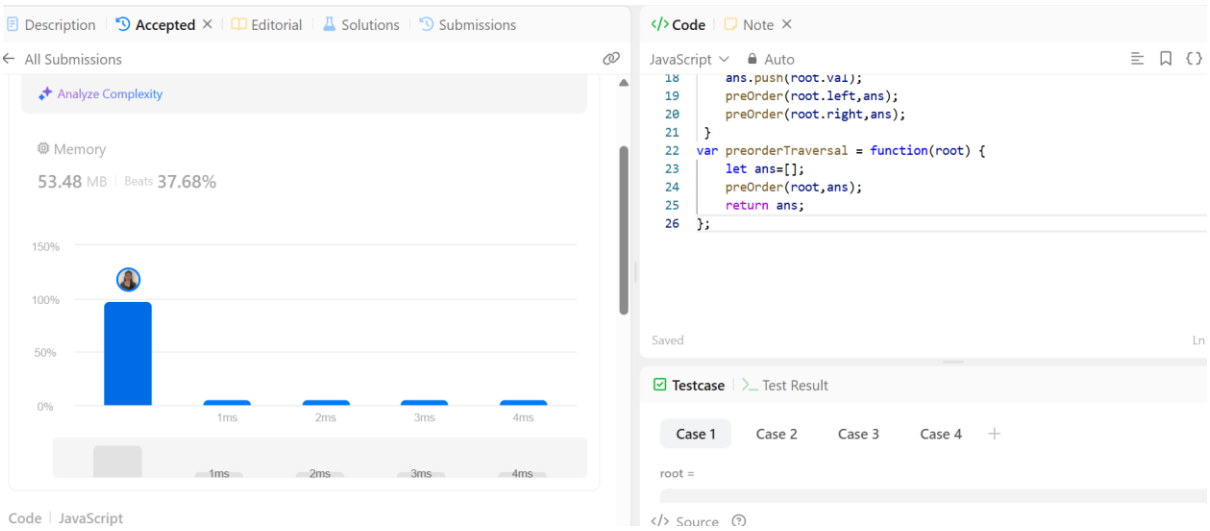
```
1
2   // preOrder traversal
3   class TreeNode{
4       constructor(data){
5           this.data=data;
6           this.left=null;
7           this.right=null;
8       }
9   }
10  let root=new TreeNode(1);
11  root.left=new TreeNode(null);
12  root.right=new TreeNode(2);
13  root.right.left=new TreeNode(3);
14
```

OUTPUT ··· Filter Code

```
1
null
2
3
```

Submission link

https://leetcode.com/problems/binary-tree-preorder-traversal/submissions/1581145964/



Conclusion

Time complexity O(n)

Each node is visited at once

Space complexity

Best case : O(log n)

Worst case:O(n)

Problem 5

Binary-tree-inorder-traversal

Given the root of a binary tree, return the inorder traversal of its nodes' values.

Example 1:

Input: root = [1,null,2,3]

Output: [1,3,2]

Question link

https://leetcode.com/problems/binary-tree-inorder-traversal/description/

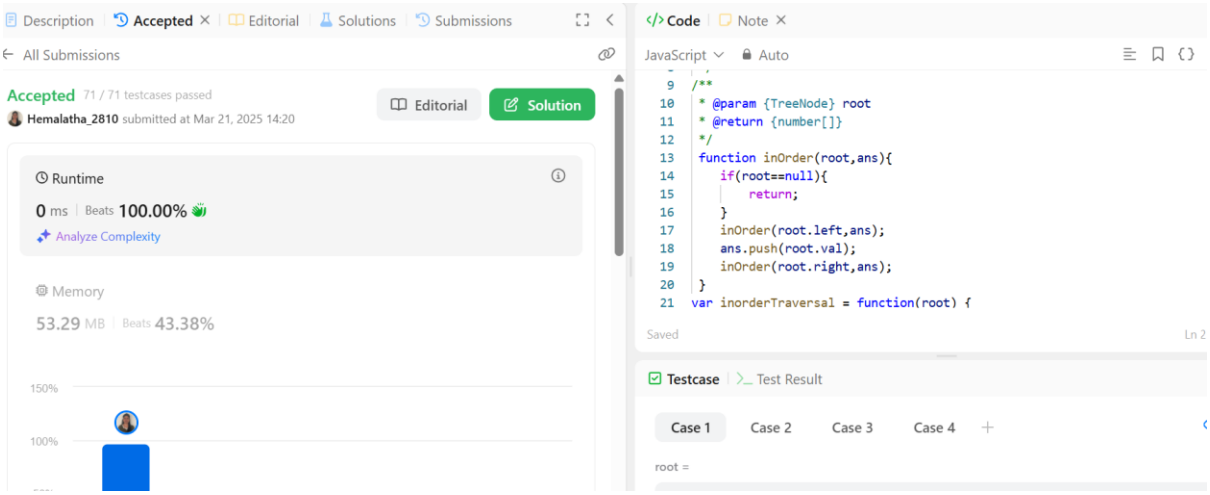
Code

```
 function inOrder(root,ans){
    if(root==null){
        return;
    }
    inOrder(root.left,ans);
    ans.push(root.val);
    inOrder(root.right,ans);
 }
var inorderTraversal = function(root) {
    let ans=[];
    inOrder(root,ans);
    return ans;
};
```

```
    }
}
let root=new TreeNode(1);
root.left=new TreeNode(null);
root.right=new TreeNode(2);
root.right.left=new TreeNode(3);

function inOrder(root){
    if(root==null){
        return ;
    }

OUTPUT ... Filter Code
null
1
3
2
```

Submission screenshot

https://leetcode.com/problems/binary-tree-inorder-traversal/submissions/1581153426/



Conclusion

Time complexity O(n)

Each node is visited at once

Space complexity

Best case : O(log n)

Worst case:O(n)