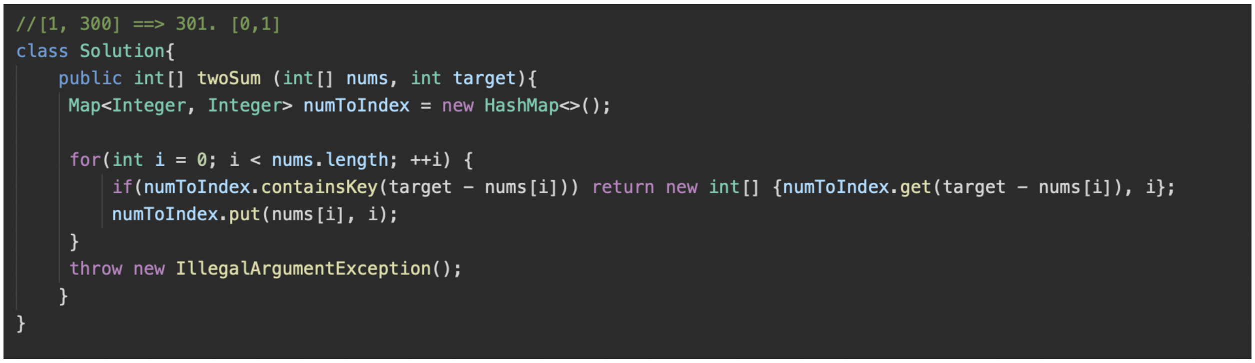
April 7th:

- Solve the question with the test Cases accepted and screenshot it here.
- Explain the code on OBS Studio.
- Figure at least three ways to solve the same question.
- Analyze the runtime complexity of all the three solutions.
- Solve the question in another language, preferably Python.
- Write a brief report on the question, skills learnt e.t.c

1. TWO SUM

Step 1:Solve the question with the test Cases accepted and screenshot it here

O(N)



```java
//[1, 300] ==> 301. [0,1]
class Solution{
    public int[] twoSum (int[] nums, int target){
     Map<Integer, Integer> numToIndex = new HashMap<>();

     for(int i = 0; i < nums.length; ++i) {
         if(numToIndex.containsKey(target - nums[i])) return new int[] {numToIndex.get(target - nums[i]), i};
         numToIndex.put(nums[i], i);
     }
     throw new IllegalArgumentException();
    }
}
```

Step 2:Explain the code on OBS Studio.

https://youtu.be/4UFzvlOq8F8

Step 3:Figure at least three other ways to solve the same question.

~ By BruteForce (C language)

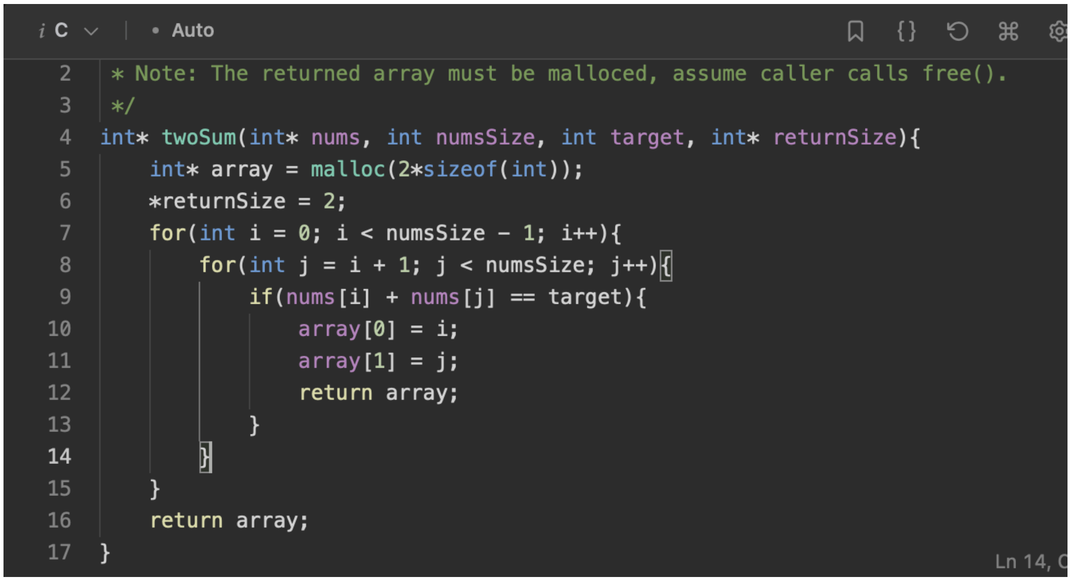
```c
int* twoSum(int* nums, int numsSize, int target, int* returnSize){
int* array = malloc(2*sizeof(int));
*returnSize = 2;
for(int i = 0; i < numsSize - 1; i++){
for(int j = i + 1; j < numsSize; j++){
if(nums[i] + nums[j] == target){
array[0] = i;
array[1] = j;
return array;
}
}
}
return array;
}
```

Step 4:Analyze the runtime complexity of all the three solutions.

O(N) for the HashMap Solution - N is the number of elements in the int[ ] nums array. Each element is visited exactly once by the for loop and HashMap operations for an overall complexity of O(n). Linear runtime complexity.

Step 5:Solve the question in another language, preferably Python.

C (Not Optimal) O(N^2)



```c
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* twoSum(int* nums, int numsSize, int target, int* returnSize){
    int* array = malloc(2*sizeof(int));
    *returnSize = 2;
    for(int i = 0; i < numsSize - 1; i++){
        for(int j = i + 1; j < numsSize; j++){
            if(nums[i] + nums[j] == target){
                array[0] = i;
                array[1] = j;
                return array;
            }
        }
    }
    return array;
}
```

Step 6:Write a brief report on the question, skills learnt e.t.c

How to .get() and .put() in a hashmap.

Use of containsKey. - looks up the key/index containing the value of interest in the HashMap's key, value pair arrangement.

Error handling, throw new IllegalArgumentException("Unrecognized value");

Using OBS Studio

4. Median of Two Sorted Arrays

Step 1: Solve the question with the test Cases accepted and screenshot it here.

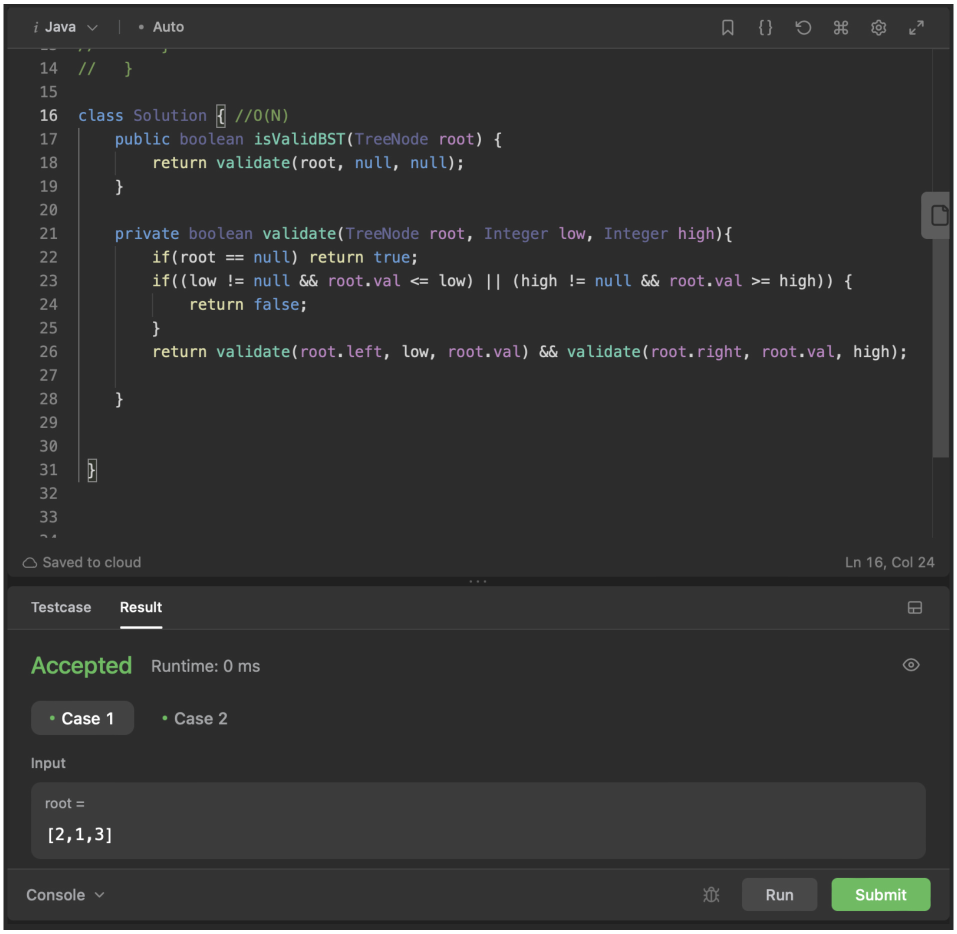
September 22nd

Q.98 VALIDATING A BINARY SEARCH TREE

Given the root of a binary tree, *determine if it is a valid binary search tree (BST).*

- Solve the question with the test Cases accepted and screenshot it here.
- Explain the code on OBS Studio.
- Figure at least three ways to solve the same question.
- Analyze the runtime complexity of all the three solutions.
- Solve the question in another language, preferably Python.
- Write a brief report on the question, skills learnt e.t.c

Approach 1 (**The recursive approach)**



```java
//   }

class Solution { //O(N)
    public boolean isValidBST(TreeNode root) {
        return validate(root, null, null);
    }

    private boolean validate(TreeNode root, Integer low, Integer high){
        if(root == null) return true;
        if((low != null && root.val <= low) || (high != null && root.val >= high)) {
            return false;
        }
        return validate(root.left, low, root.val) && validate(root.right, root.val, high);

    }


}


```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

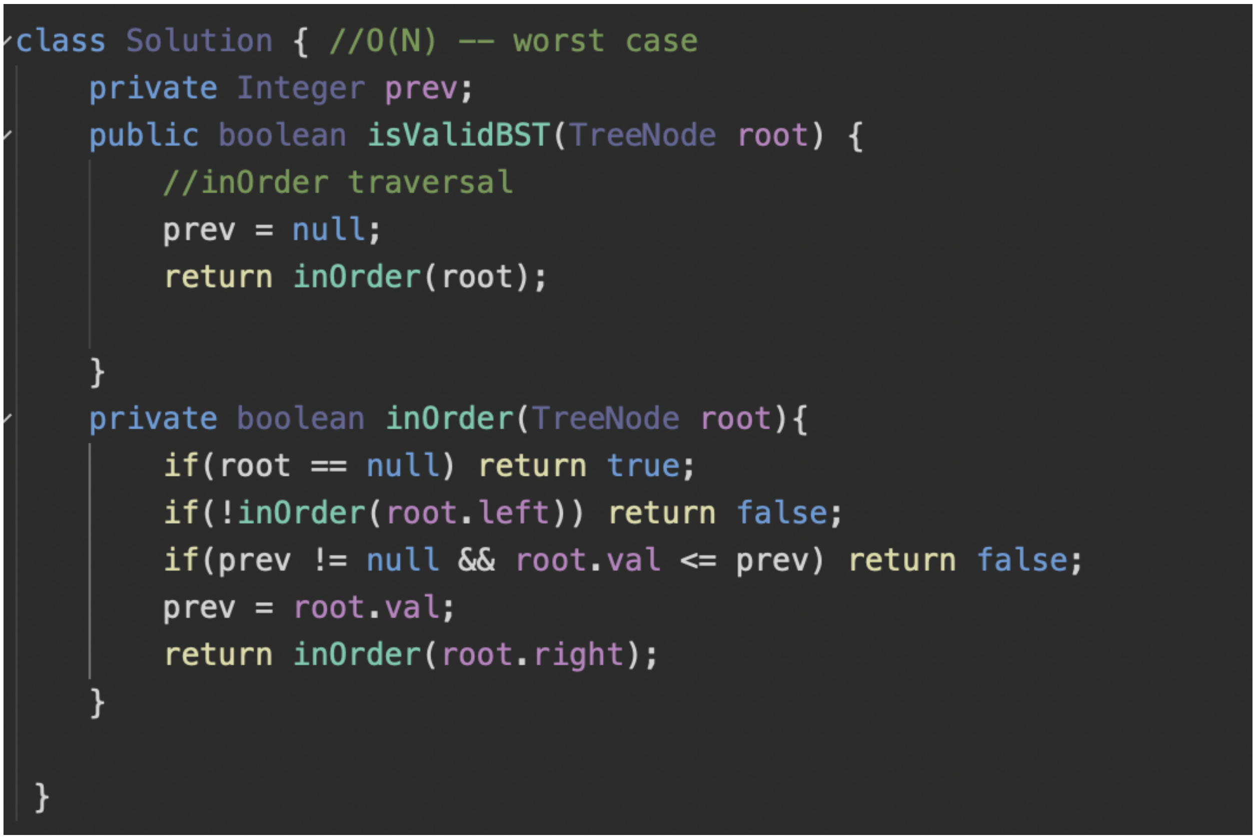
root =

[2,1,3]

Time Complexity

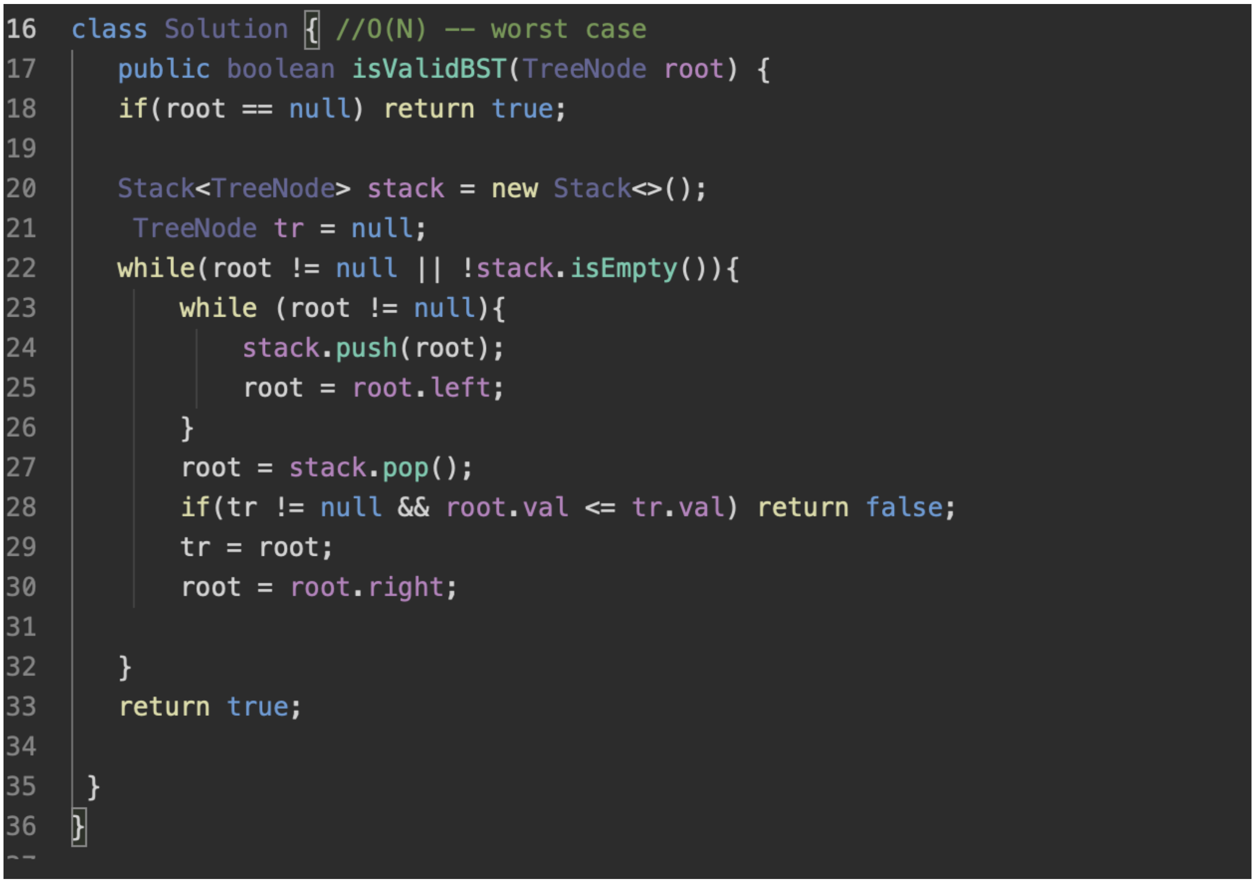
Runs in linear time - O(N)

Approach 2
Using a **recursive inOrder traversal**



```
class Solution { //O(N) -- worst case
    private Integer prev;
    public boolean isValidBST(TreeNode root) {
        //inOrder traversal
        prev = null;
        return inOrder(root);

    }
    private boolean inOrder(TreeNode root){
        if(root == null) return true;
        if(!inOrder(root.left)) return false;
        if(prev != null && root.val <= prev) return false;
        prev = root.val;
        return inOrder(root.right);
    }

 }
```

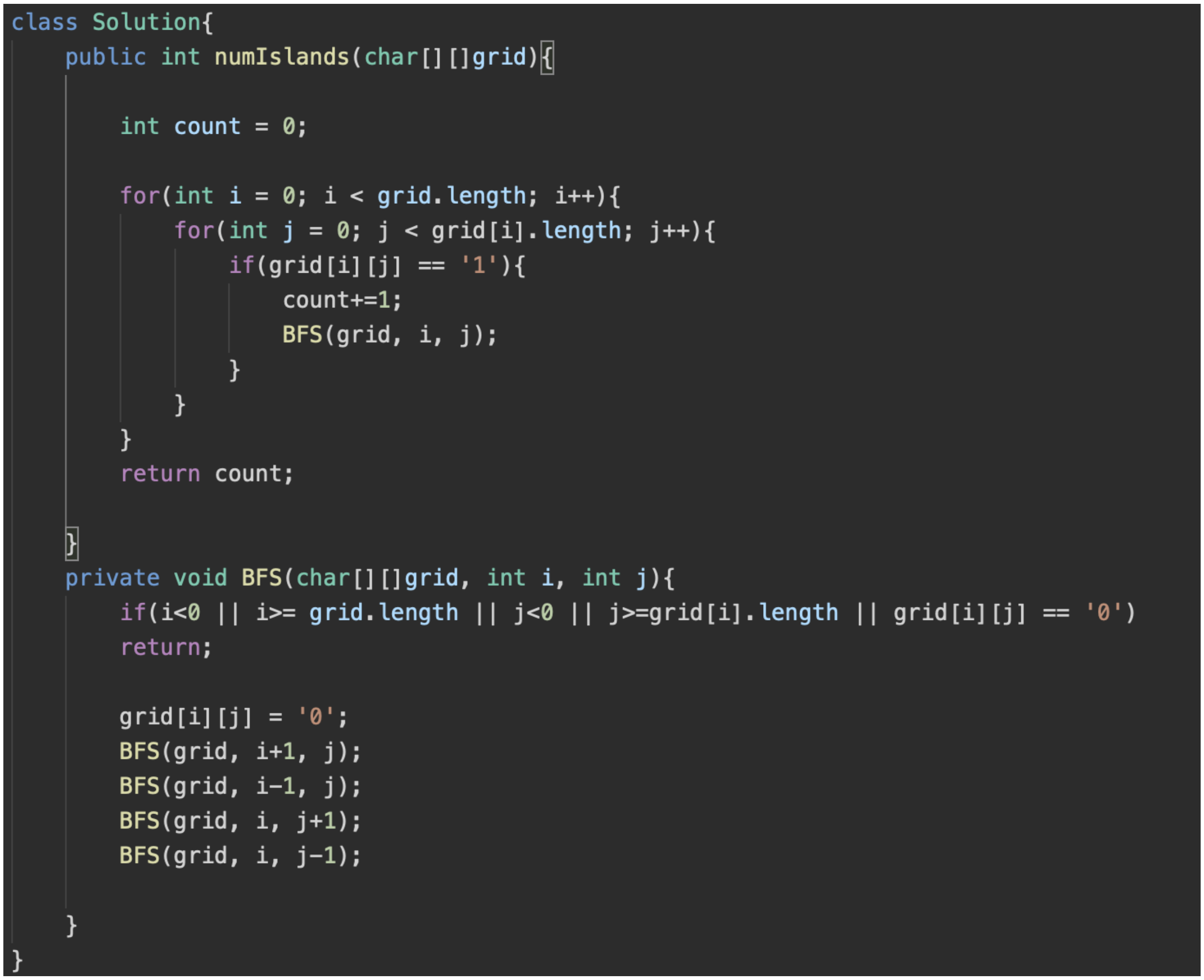
Approach 3 **(Iterative inOrder Traversal)**



```
class Solution { //O(N) -- worst case
   public boolean isValidBST(TreeNode root) {
   if(root == null) return true;

   Stack<TreeNode> stack = new Stack<>();
    TreeNode tr = null;
   while(root != null || !stack.isEmpty()){
       while (root != null){
           stack.push(root);
           root = root.left;
       }
       root = stack.pop();
       if(tr != null && root.val <= tr.val) return false;
       tr = root;
       root = root.right;

   }
   return true;

 }
}
```

October 1st

Number of Islands #200



```java
class Solution{
    public int numIslands(char[][]grid){

        int count = 0;

        for(int i = 0; i < grid.length; i++){
            for(int j = 0; j < grid[i].length; j++){
                if(grid[i][j] == '1'){
                    count+=1;
                    BFS(grid, i, j);
                }
            }
        }
        return count;

    }
    private void BFS(char[][]grid, int i, int j){
        if(i<0 || i>= grid.length || j<0 || j>=grid[i].length || grid[i][j] == '0')
        return;

        grid[i][j] = '0';
        BFS(grid, i+1, j);
        BFS(grid, i-1, j);
        BFS(grid, i, j+1);
        BFS(grid, i, j-1);

    }
}
```