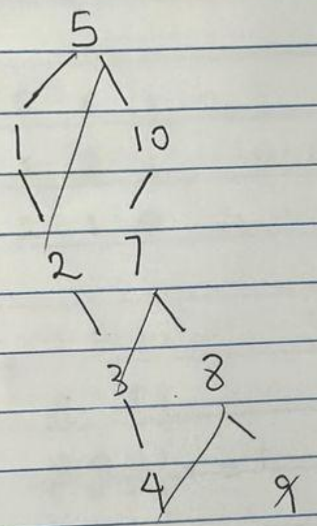
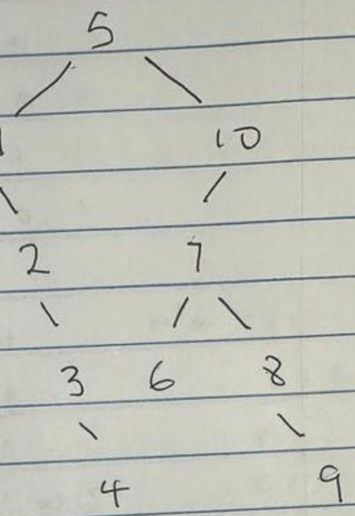
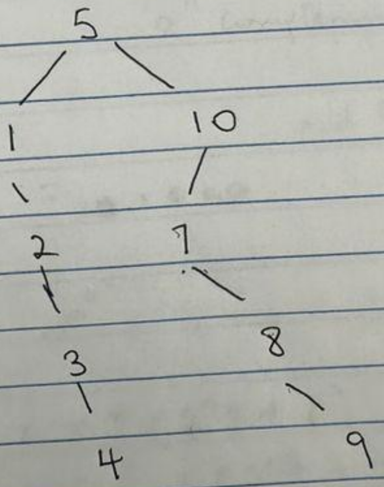
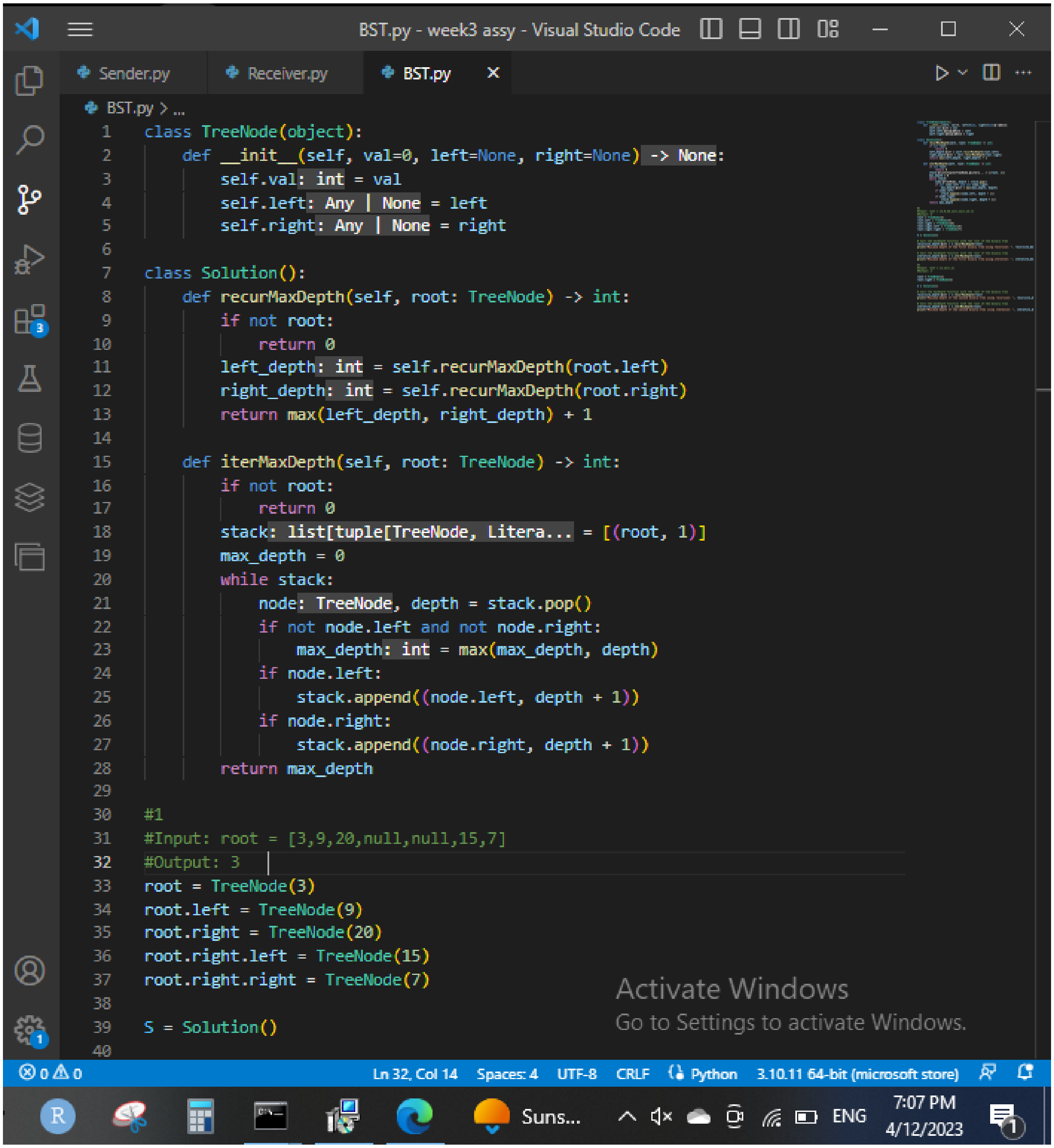
5 1 2 3 4 10 7 8 9 6



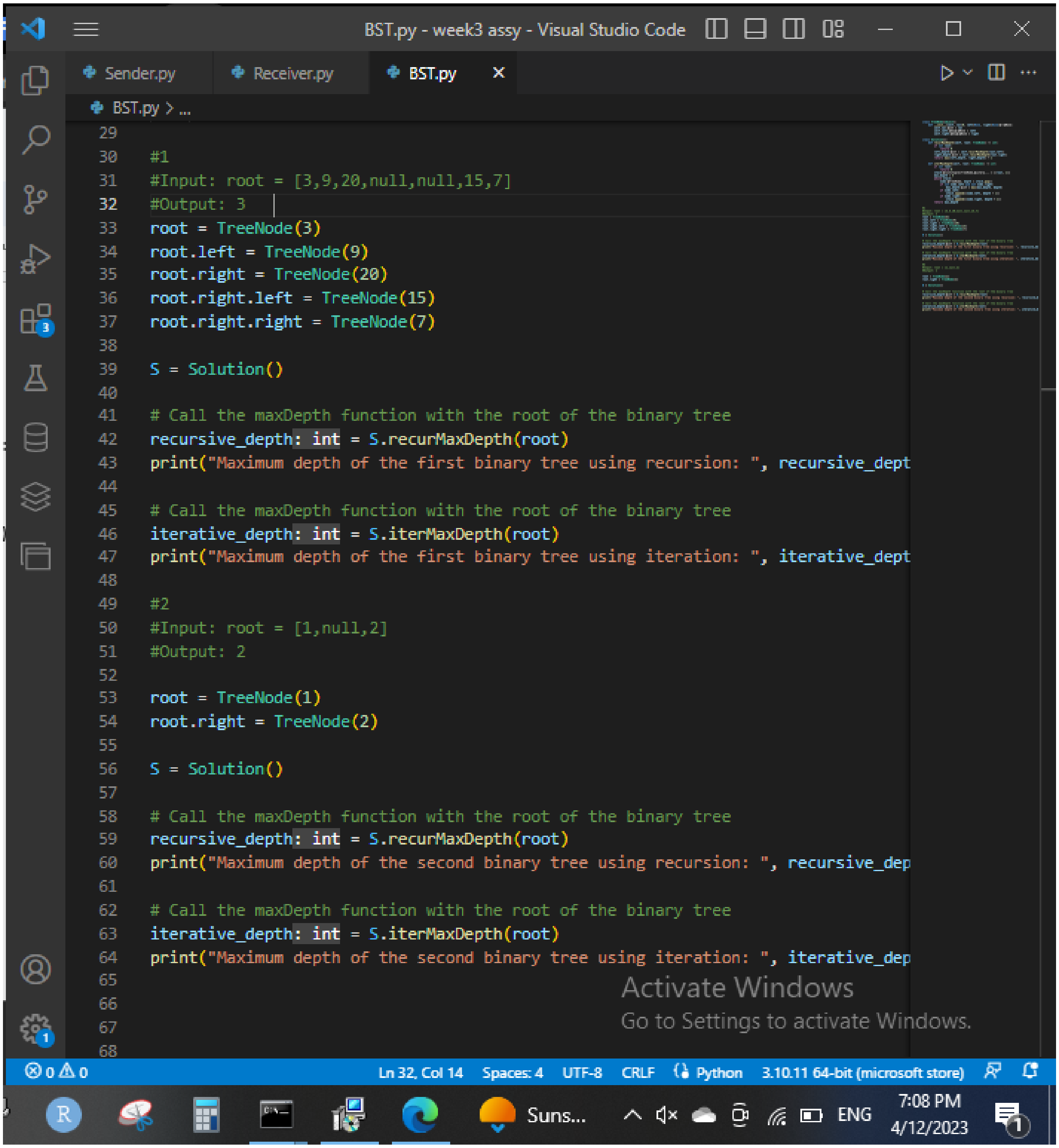


Delete 6

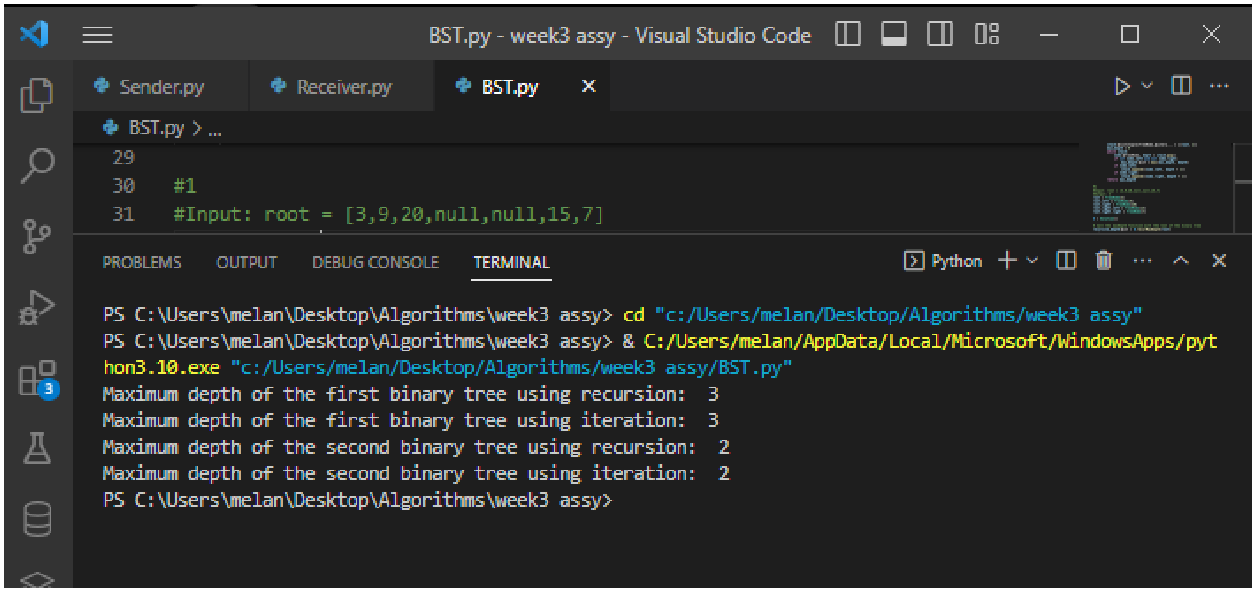




```
class TreeNode(object):
    def __init__(self, val=0, left=None, right=None) -> None:
        self.val: int = val
        self.left: Any | None = left
        self.right: Any | None = right

class Solution():
    def recurMaxDepth(self, root: TreeNode) -> int:
        if not root:
            return 0
        left_depth: int = self.recurMaxDepth(root.left)
        right_depth: int = self.recurMaxDepth(root.right)
        return max(left_depth, right_depth) + 1

    def iterMaxDepth(self, root: TreeNode) -> int:
        if not root:
            return 0
        stack: list[tuple[TreeNode, Litera... = [(root, 1)]
        max_depth = 0
        while stack:
            node: TreeNode, depth = stack.pop()
            if not node.left and not node.right:
                max_depth: int = max(max_depth, depth)
            if node.left:
                stack.append((node.left, depth + 1))
            if node.right:
                stack.append((node.right, depth + 1))
        return max_depth

#1
#Input: root = [3,9,20,null,null,15,7]
#Output: 3
root = TreeNode(3)
root.left = TreeNode(9)
root.right = TreeNode(20)
root.right.left = TreeNode(15)
root.right.right = TreeNode(7)

S = Solution()
```



```python

#1
#Input: root = [3,9,20,null,null,15,7]
#Output: 3
root = TreeNode(3)
root.left = TreeNode(9)
root.right = TreeNode(20)
root.right.left = TreeNode(15)
root.right.right = TreeNode(7)

S = Solution()

# Call the maxDepth function with the root of the binary tree
recursive_depth: int = S.recurMaxDepth(root)
print("Maximum depth of the first binary tree using recursion: ", recursive_dept

# Call the maxDepth function with the root of the binary tree
iterative_depth: int = S.iterMaxDepth(root)
print("Maximum depth of the first binary tree using iteration: ", iterative_dept

#2
#Input: root = [1,null,2]
#Output: 2

root = TreeNode(1)
root.right = TreeNode(2)

S = Solution()

# Call the maxDepth function with the root of the binary tree
recursive_depth: int = S.recurMaxDepth(root)
print("Maximum depth of the second binary tree using recursion: ", recursive_dep

# Call the maxDepth function with the root of the binary tree
iterative_depth: int = S.iterMaxDepth(root)
print("Maximum depth of the second binary tree using iteration: ", iterative_dep
```



```
#1
#Input: root = [3,9,20,null,null,15,7]
```

```
PS C:\Users\melan\Desktop\Algorithms\week3 assy> cd "c:/Users/melan/Desktop/Algorithms/week3 assy"
PS C:\Users\melan\Desktop\Algorithms\week3 assy> & C:/Users/melan/AppData/Local/Microsoft/WindowsApps/python3.10.exe "c:/Users/melan/Desktop/Algorithms/week3 assy/BST.py"
Maximum depth of the first binary tree using recursion:  3
Maximum depth of the first binary tree using iteration:  3
Maximum depth of the second binary tree using recursion:  2
Maximum depth of the second binary tree using iteration:  2
PS C:\Users\melan\Desktop\Algorithms\week3 assy>
```

CODE

```
class TreeNode(object):
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

class Solution():
    def recurMaxDepth(self, root: TreeNode) -> int:
        if not root:
            return 0
        left_depth = self.recurMaxDepth(root.left)
        right_depth = self.recurMaxDepth(root.right)
        return max(left_depth, right_depth) + 1

    def iterMaxDepth(self, root: TreeNode) -> int:
        if not root:
            return 0
        stack = [(root, 1)]
        max_depth = 0
        while stack:
            node, depth = stack.pop()
            if not node.left and not node.right:
                max_depth = max(max_depth, depth)
            if node.left:
                stack.append((node.left, depth + 1))
            if node.right:
                stack.append((node.right, depth + 1))
        return max_depth

#1
#Input: root = [3,9,20,null,null,15,7]
#Output: 3
root = TreeNode(3)
root.left = TreeNode(9)
root.right = TreeNode(20)
root.right.left = TreeNode(15)
root.right.right = TreeNode(7)

S = Solution()

# Call the maxDepth function with the root of the binary tree
recursive_depth = S.recurMaxDepth(root)
print("Maximum depth of the first binary tree using recursion: ", recursive_depth)

# Call the maxDepth function with the root of the binary tree
iterative_depth = S.iterMaxDepth(root)
print("Maximum depth of the first binary tree using iteration: ", iterative_depth)

#2
#Input: root = [1,null,2]
#Output: 2

root = TreeNode(1)
root.right = TreeNode(2)

S = Solution()

# Call the maxDepth function with the root of the binary tree
recursive_depth = S.recurMaxDepth(root)
print("Maximum depth of the second binary tree using recursion: ", recursive_depth)

# Call the maxDepth function with the root of the binary tree
iterative_depth = S.iterMaxDepth(root)
print("Maximum depth of the second binary tree using iteration: ", iterative_depth)
```

COMPARISON

Time Complexity of Iterative Solution: The time complexity of the iterative solution is also O(N), where N is the number of nodes in the binary tree. This is because we visit each node exactly once using a depth-first traversal with a stack.

Space Complexity of Iterative Solution: The space complexity of the iterative solution is O(M), where M is the maximum number of nodes at any level of the binary tree. In the worst case, the binary tree can be completely balanced, resulting in M = N/2 nodes at the maximum level. Therefore, the space complexity is O(N) in the worst case and O(1) in the best case.

Time Complexity of Recursive Solution: The time complexity of the recursive solution is O(N), where N is the number of nodes in the binary tree. This is because we visit each node exactly once in a depth-first manner.

Space Complexity of Recursive Solution: The space complexity of the recursive solution is O(H), where H is the height of the binary tree. In the worst case, the binary tree can be skewed, resulting in a height of N. In the best case, the binary tree can be balanced, resulting in a height of log(N). Therefore, the space complexity ranges from O(log(N)) to O(N).