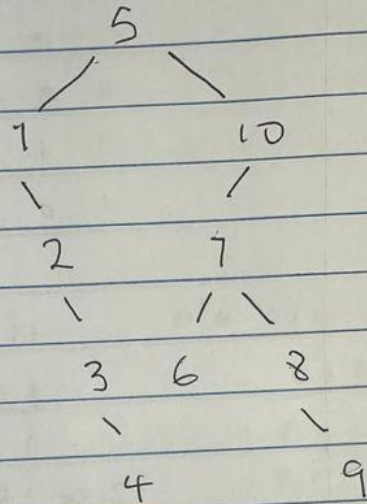
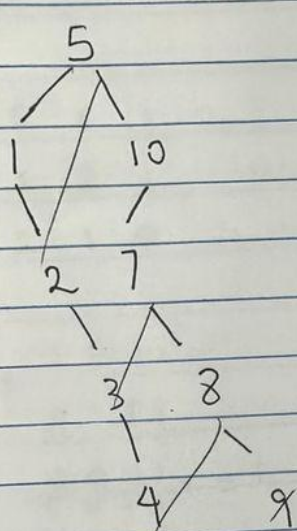
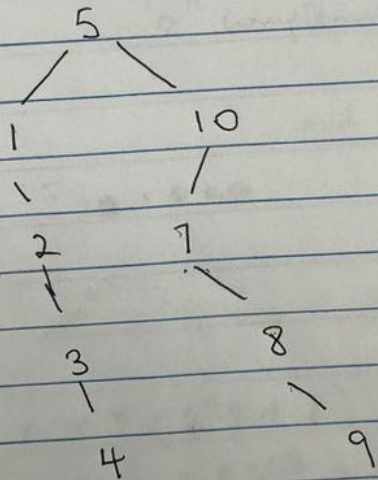


5 1 2 3 4 10 7 8 9 6



Delete 6



VS Code Window Title: BST.py - week3 assy - Visual Studio Code

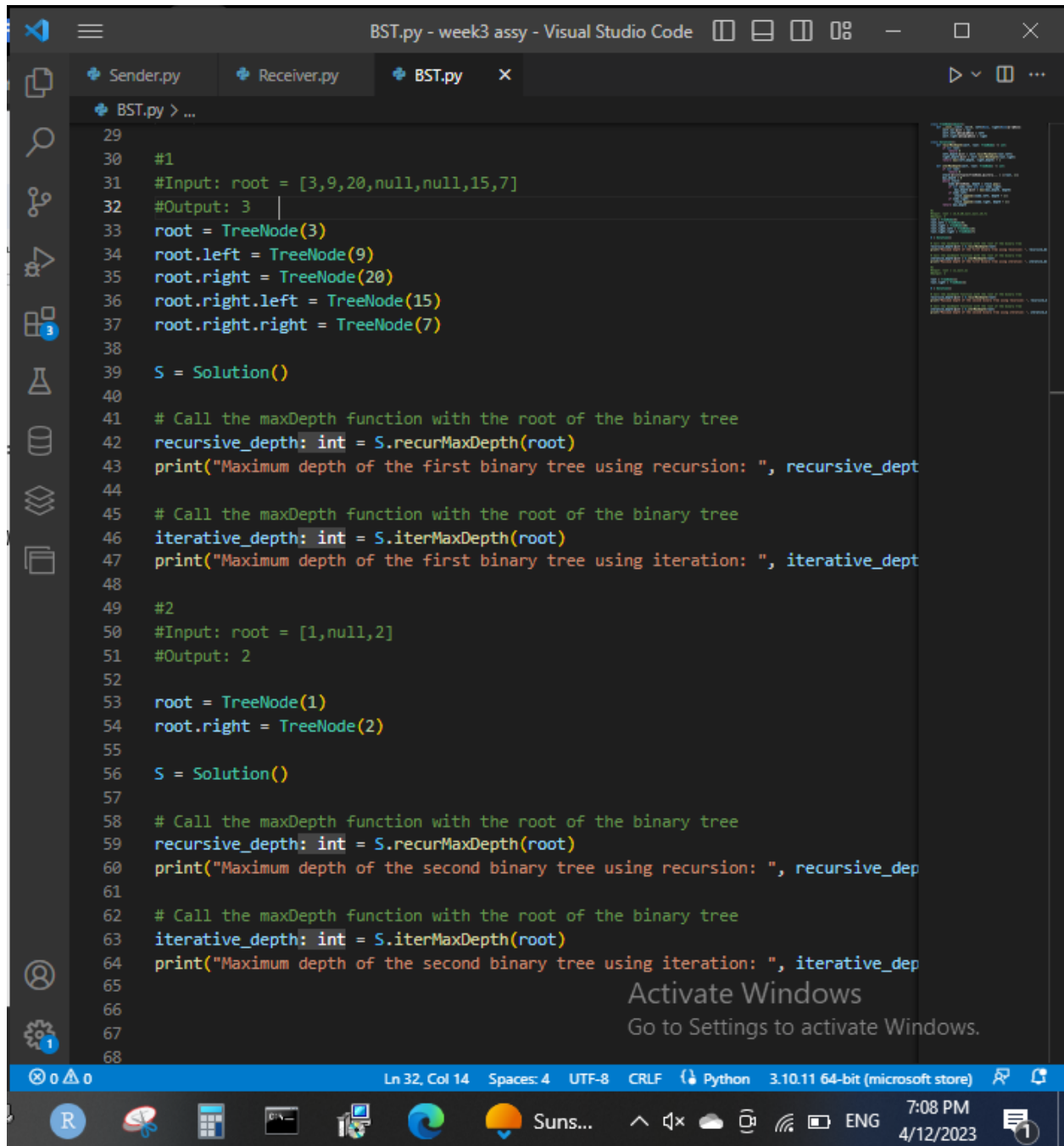
File Explorer: Sender.py, Receiver.py, BST.py

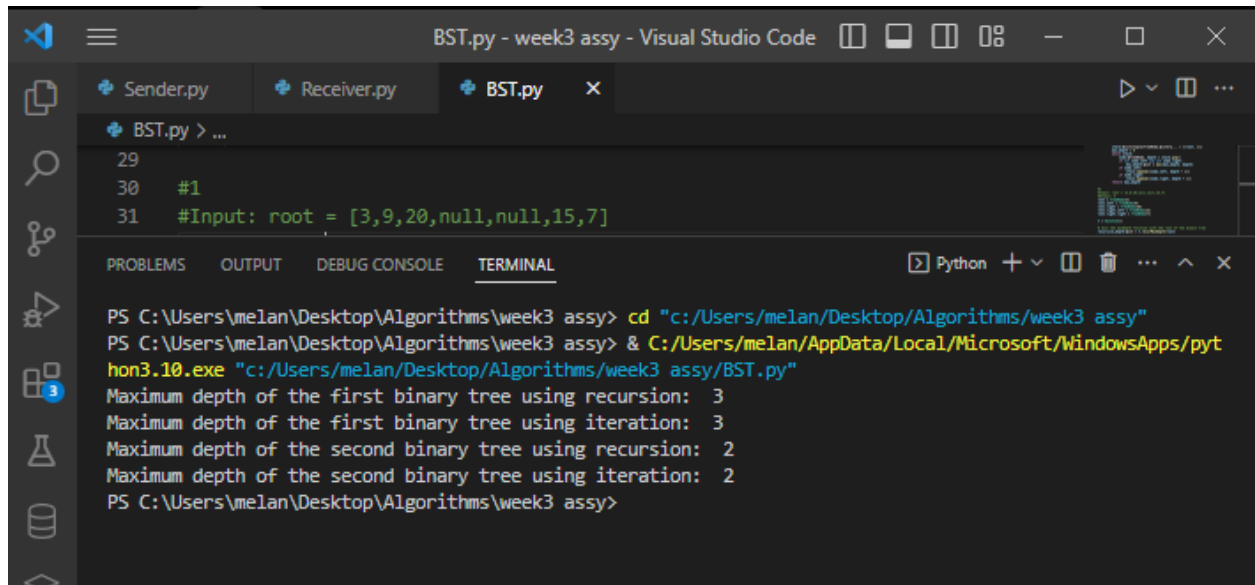
Code Editor: BST.py > ...

```
1 class TreeNode(object):
2     def __init__(self, val=0, left=None, right=None) -> None:
3         self.val: int = val
4         self.left: Any | None = left
5         self.right: Any | None = right
6
7 class Solution():
8     def recurMaxDepth(self, root: TreeNode) -> int:
9         if not root:
10             return 0
11         left_depth: int = self.recurMaxDepth(root.left)
12         right_depth: int = self.recurMaxDepth(root.right)
13         return max(left_depth, right_depth) + 1
14
15     def iterMaxDepth(self, root: TreeNode) -> int:
16         if not root:
17             return 0
18         stack: list[tuple[TreeNode, Litera... = [(root, 1)]
19         max_depth = 0
20         while stack:
21             node: TreeNode, depth = stack.pop()
22             if not node.left and not node.right:
23                 max_depth: int = max(max_depth, depth)
24             if node.left:
25                 stack.append((node.left, depth + 1))
26             if node.right:
27                 stack.append((node.right, depth + 1))
28         return max_depth
29
30 #1
31 #Input: root = [3,9,20,null,null,15,7]
32 #Output: 3
33 root = TreeNode(3)
34 root.left = TreeNode(9)
35 root.right = TreeNode(20)
36 root.right.left = TreeNode(15)
37 root.right.right = TreeNode(7)
38
39 s = Solution()
40
```

Output Console: [Empty]

Taskbar: Windows 11 taskbar with icons for R, File Explorer, Calculator, Task View, Edge, and others. System tray shows date 4/12/2023 and time 7:07 PM.





The screenshot shows the Visual Studio Code editor with a file named `BST.py` open. The code in the editor includes a comment `#1` and an input list `[3,9,20,null,null,15,7]`. The terminal window at the bottom shows the command prompt running the script, which outputs the maximum depth of the first binary tree using recursion (3), the maximum depth of the first binary tree using iteration (3), the maximum depth of the second binary tree using recursion (2), and the maximum depth of the second binary tree using iteration (2).

```
29
30 #1
31 #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 $O(N)$.