ASSIGNMENT - 3

(2022129)

Q1. For each statement, indicate whether it is possible to uniquely identify a tree for the given traversal combinations and provide a proper reason for the answer.

NOTE: *For Q1 level order will not contain 'N' as input.*

(a) **NO**

In a postorder traversal, the root node is visited last, while in a level-order traversal, the root node is visited first. This discrepancy in the order of visiting the root node leads to ambiguity. Thus, combining postorder and level-order traversals does not give sufficient information to uniquely identify a binary tree.

(b) **YES**

In a preorder traversal, the root node is visited first, followed by the left subtree and then the right subtree. In an inorder traversal, nodes are visited in the order: left subtree, root, right subtree. Given both inorder and preorder traversals, the first element in the preorder traversal must be the root. Using this root, we can locate its position in the inorder traversal, thereby dividing the inorder traversal into left and right subtrees recursively.

Pseudocode:

```
ConstructTree(Inorder[], Preorder[])
    // Assume there exist functions FindIndex and ExtractSubtree
    // that find the index of an element in an array and extract a subtree respectively
    if length(Inorder) = 0 or length(Preorder) = 0
        return null

    root_value = Preorder[0]
    root_index_inorder = FindIndex(Inorder, root_value)

    root = new TreeNode(root_value)

    root.left = ConstructTree(ExtractSubtree(Inorder, 0, root_index_inorder),
                              ExtractSubtree(Preorder, 1, root_index_inorder + 1))
    root.right = ConstructTree(ExtractSubtree(Inorder, root_index_inorder + 1, length(Inorder)),
                               ExtractSubtree(Preorder, root_index_inorder + 1, length(Preorder)))

    return root
```

(c) **YES**

The level-order traversal provides the order in which nodes appear at each level. By taking the first node from the level-order traversal as the root, we can partition the inorder traversal into left and right subtrees recursively.

Pseudocode:

```
ConstructTree(Inorder[], LevelOrder[])
    // Assume there exist functions FindIndex and ExtractSubtree
    // that find the index of an element in an array and extract a subtree respectively
    if length(Inorder) = 0 or length(LevelOrder) = 0
        return null

    root_value = LevelOrder[0]
    root_index_inorder = FindIndex(Inorder, root_value)

    root = new TreeNode(root_value)

    root.left = ConstructTree(ExtractSubtree(Inorder, 0, root_index_inorder),
                              ExtractSubtree(LevelOrder, 1, root_index_inorder + 1))
    root.right = ConstructTree(ExtractSubtree(Inorder, root_index_inorder + 1, length(Inorder)),
                               ExtractSubtree(LevelOrder, root_index_inorder + 1, length(LevelOrder)))

    return root
```

(d) **YES**

Given the postorder traversal, the last element corresponds to the root node. By using this root node, we can partition the inorder traversal into left and right subtrees recursively.

Pseudocode:

```
ConstructTree(Inorder[], Postorder[])
    // Assume there exist functions FindIndex and ExtractSubtree
    // that find the index of an element in an array and extract a subtree respectively
    if length(Inorder) = 0 or length(Postorder) = 0
        return null

    root_value = Postorder[length(Postorder) - 1]
    root_index_inorder = FindIndex(Inorder, root_value)

    root = new TreeNode(root_value)

    root.left = ConstructTree(ExtractSubtree(Inorder, 0, root_index_inorder),
                              ExtractSubtree(Postorder, 0, root_index_inorder))
    root.right = ConstructTree(ExtractSubtree(Inorder, root_index_inorder + 1, length(Inorder)),
                               ExtractSubtree(Postorder, root_index_inorder, length(Postorder) - 1))

    return root
```

(e) **NO**

While pre-order and post-order traversals provide information about the root and the subtrees, they don't specify the order of the children relative to each other. Without the inorder traversal, we can't determine which nodes belong to the left subtree and which belong to the right subtree.