```
'''

    Time complexity: O(N^2)
    Space complexity: O(H)

    where N is the number of nodes in the input tree
    and H is the height of the input tree

'''

from sys import stdin, setrecursionlimit
import queue

setrecursionlimit(10 ** 6)


#Following is the structure used to represent the Binary Tree Node
class BinaryTreeNode:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None



def buildTreeHelper(postOrder, postStart, postEnd, inOrder, inStart, inEnd) :
    if (postStart > postEnd) or (inStart > inEnd) :
        return None

    rootVal = postOrder[postEnd]
    root =  BinaryTreeNode(rootVal)

    # Find parent element index from inOrder array
    k = 0
    for i in range(inStart, inEnd + 1) :
        if (rootVal == inOrder[i]) :
            k = i
            break

    root.left = buildTreeHelper(postOrder, postStart, postStart + k - inStart - 1, inOrder, inStart, k
- 1)
    root.right = buildTreeHelper(postOrder, postStart + k - inStart, postEnd - 1, inOrder, k + 1,
inEnd)

    return root


def buildTree(postOrder, inOrder, n) :
    postStart = 0
    postEnd = n - 1
    inStart = 0
    inEnd = n - 1

    return buildTreeHelper(postOrder, postStart, postEnd, inOrder, inStart, inEnd)




'''------------------------- Utility Functions -------------------------'''

def printLevelWise(root):
    if root is None :
        return
```

```
    pendingNodes = queue.Queue()
    pendingNodes.put(root)
    pendingNodes.put(None)

    while not pendingNodes.empty():
        frontNode = pendingNodes.get()

        if frontNode is None :
            print()

            if not pendingNodes.empty() :
                pendingNodes.put(None)

        else :
            print(frontNode.data, end = " ")

            if frontNode.left is not None :
                pendingNodes.put(frontNode.left)


            if frontNode.right is not None :
                pendingNodes.put(frontNode.right)




#Taking level-order input using fast I/O method
def takeInput():
    n = int(stdin.readline().strip())

    if n == 0 :
        return list(), list(), 0

    postOrder = list(map(int, stdin.readline().strip().split(" ")))
    inOrder = list(map(int, stdin.readline().strip().split(" ")))

    return postOrder, inOrder, n


# Main
postOrder, inOrder, n = takeInput()
root = buildTree(postOrder, inOrder, n)
printLevelWise(root)
```