

```
...
```

```
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(preOrder, preStart, preEnd, inOrder, inStart, inEnd) :  
    if (preStart > preEnd) or (inStart > inEnd) :  
        return None
```

```
    rootVal = preOrder[preStart]  
    root = BinaryTreeNode(rootVal)
```

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

```
    root.left = buildTreeHelper(preOrder, preStart + 1, preStart + (k - inStart), inOrder, inStart, k -  
1)  
    root.right = buildTreeHelper(preOrder, preStart + (k - inStart) + 1, preEnd, inOrder, k + 1, inEnd)  
  
    return root
```

```
def buildTree(preOrder, inOrder, n) :  
    preStart = 0  
    preEnd = n - 1  
    inStart = 0  
    inEnd = n - 1  
  
    return buildTreeHelper(preOrder, preStart, preEnd, 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

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

    return preOrder, inOrder, n

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

```