**Problem Statement**

**Title: Reconstruct Binary Tree from Preorder and Inorder Traversals**

**Problem Description:** Emma is reconstructing a project management tree from two lists: one representing the tasks in preorder and the other representing the tasks in inorder. She needs to build the binary tree from these lists. Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and then print the Level order Traversal of the new Tree.

**Input Format:**

* The first line contains an integer N, the number of elements in the tree.
* The second line contains N space-separated elements representing the preorder traversal.
* The third line contains N space-separated elements representing the inorder traversal.

**Output Format:** Print the Level Order Traversal, with each level on a new line.

**Constraints:**

* The number of nodes in the tree is in the range [0, 1000].
* The elements of the tree are integers.

**Examples:**

**Example 1:**

Input:

6

3 9 20 15 7

9 3 15 20 7

Output:

3

9 20

15 7

**Example 2:**

Input:

3

1 2 3

2 1 3

Output:

1

2 3

**Test Cases:**

**Test Case 1:**

Input:

7

3 9 8 5 20 15 7

8 9 5 3 15 20 7

Output:

3

9 20

8 5 15 7

**Test Case 2:**

Input:

5

1 2 4 5 3

4 2 5 1 3

Output:

1

2 3

4 5

**Test Case 3:**

Input:

4

10 8 7 6

8 10 7 6

Output:

10

8 7

6

**Test Case 4:**

Input:

3

5 6 7

6 5 7

Output:

5

6 7

**Test Case 5:**

Input:

6

7 10 4 3 1 2

4 10 3 7 2 1

Output:

7

10 1

4 3 2

**Solution in Python**

Here is a Python solution that reconstructs the binary tree from the given preorder and inorder traversals and then performs a level order traversal:

python

Copy code

class TreeNode:

def \_\_init\_\_(self, x):

self.val = x

self.left = None

self.right = None

def build\_tree(preorder, inorder):

if not preorder or not inorder:

return None

root\_val = preorder.pop(0)

root = TreeNode(root\_val)

inorder\_index = inorder.index(root\_val)

root.left = build\_tree(preorder, inorder[:inorder\_index])

root.right = build\_tree(preorder, inorder[inorder\_index + 1:])

return root

def level\_order\_traversal(root):

if not root:

return []

queue = [root]

result = []

while queue:

level\_size = len(queue)

current\_level = []

for \_ in range(level\_size):

node = queue.pop(0)

current\_level.append(node.val)

if node.left:

queue.append(node.left)

if node.right:

queue.append(node.right)

result.append(current\_level)

return result

def print\_level\_order(levels):

for level in levels:

print(" ".join(map(str, level)))

def main():

import sys

input = sys.stdin.read

data = input().strip().split()

N = int(data[0])

preorder = list(map(int, data[1:N+1]))

inorder = list(map(int, data[N+1:2\*N+1]))

root = build\_tree(preorder, inorder)

levels = level\_order\_traversal(root)

print\_level\_order(levels)

if \_\_name\_\_ == "\_\_main\_\_":

main()

This solution reads the input values, constructs the binary tree using the build\_tree function, and performs a level order traversal using the level\_order\_traversal function. The output is printed with each level on a new line