Construct Full Binary Tree using its Preorder traversal and Preorder traversal of its mirror tree

Given two arrays that represent Preorder traversals of a full binary tree and its mirror tree, we need to write a program to construct the binary tree using these two Preorder traversals.

A **Full Binary Tree** is a binary tree where every node has either 0 or 2 children.
Note: It is not possible to construct a general binary tree using these two traversals. But we can create a full binary tree using the above traversals without any ambiguity.

Examples:
Input : preOrder[] = {1,2,4,5,3,6,7}

preOrderMirror[] = {1,3,7,6,2,5,4}

```
Output :          1

               /    \

              2      3

           /   \   /  \

          4     5 6    7
```

- **Method 1**: Let us consider the two given arrays as preOrder[] = {1, 2, 4, 5, 3, 6, 7} and preOrderMirror[] = {1 ,3 ,7 ,6 ,2 ,5 ,4}.
  In both preOrder[] and preOrderMirror[], the leftmost element is root of tree. Since the tree is full and array size is more than 1.
- The value next to 1 in preOrder[], must be left child of the root and value next to 1 in preOrderMirror[] must be right child of root. So we know 1 is root and 2 is left child and 3 is the right child. How to find the all nodes in left subtree? We know 2 is root of all nodes in left subtree and 3 is root of all nodes in right subtree.
- All nodes from and 2 in preOrderMirror[] must be in left subtree of root node 1 and all node after 3 and before 2 in preOrderMirror[] must be in right subtree of root node 1.
- Now we know 1 is root, elements {2, 5, 4} are in left subtree, and the elements {3, 7, 6} are in the right subtree.

```
       1

     /    \

    /      \

 {2,5,4}  {3,7,6}
```

- We will recursively follow the above approach and get the below tree:

```
            1
          /    \
         2       3
       /   \   /  \
      4     5 6    7
```

Below is the implementation of above approach:

```
C++ program to construct full binary tree
// using its preorder traversal and preorder
// traversal of its mirror tree

#include<bits/stdc++.h>
using namespace std;

// A Binary Tree Node
struct Node
{
    int data;
    struct Node *left, *right;
};

// Utility function to create a new tree node

Node* newNode(int data)
{
    Node *temp = new Node;
    temp->data = data;
    temp->left = temp->right = NULL;
    return temp;
}

// A utility function to print inorder traversal
// of a Binary Tree
void printInorder(Node* node)
{
    if (node == NULL)
        return;
    printInorder(node->left);
    printf("%d ", node->data);
    printInorder(node->right);
}

// A recursive function to construct Full binary tree
//  from pre[] and preM[]. preIndex is used to keep
// track of index in pre[]. l is low index and h is high
//index for the current subarray in preM[]

Node* constructBinaryTreeUtil(int pre[], int preM[],
                    int &preIndex, int l,int h,int size)
{
    // Base case
    if (preIndex >= size || l > h)
        return NULL;

    // The first node in preorder traversal is root.
    // So take the node at preIndex from preorder and
    // make it root, and increment preIndex

  Node* root = newNode(pre[preIndex]);
        ++(preIndex);

    // If the current subarray has only one element,
    // no need to recur
    if (l == h)
        return root;

    // Search the next element of pre[] in preM[]
    int i;
    for (i = l; i <= h; ++i)
        if (pre[preIndex] == preM[i])
            break;

    // construct left and right subtrees recursively
    if (i <= h)
    {
        root->left = constructBinaryTreeUtil (pre, preM,
                                    preIndex, i, h, size);
        root->right = constructBinaryTreeUtil (pre, preM,
                                 preIndex, l+1, i-1, size);
    }

     // return root
    return root;
```

```
}

// function to construct full binary tree
// using its preorder traversal and preorder
// traversal of its mirror tree

void constructBinaryTree(Node* root,int pre[],
                        int preMirror[], int size)
{
    int preIndex = 0;
    int preMIndex = 0;

    root =  constructBinaryTreeUtil(pre,preMirror,
                            preIndex,0,size-1,size);

    printInorder(root);
}

// Driver program to test above functions
int main()
{
    int preOrder[] = {1,2,4,5,3,6,7};
    int preOrderMirror[] = {1,3,7,6,2,5,4};

    int size = sizeof(preOrder)/sizeof(preOrder[0]);

    Node* root = new Node;

    constructBinaryTree(root,preOrder,preOrderMirror,size);

    return 0;
}
```

Output

```
4 2 5 1 6 3 7
```

Time Complexity: O(n^2)
Auxiliary Space: O(n), The extra space is used due to the recursion call stack