Construct Full Binary Tree from given preorder and postorder traversals

Given two arrays that represent preorder and postorder traversals of a full binary tree, construct the binary tree.

A **Full Binary Tree** is a binary tree where every node has either 0 or 2 children

Following are examples of Full Trees.

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


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


           1
         /   \
       2       3
     /  \     /  \
    4    5   6    7
  /  \
 8    9
```

It is not possible to construct a general Binary Tree from preorder and postorder traversals (See this). But if know that the Binary Tree is Full, we can construct the tree without ambiguity. Let us understand this with the help of following example.

Let us consider the two given arrays as pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7} and post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
In pre[], 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 pre[], must be left child of root. So we know 1 is root and 2 is left child. How to find the all nodes in left subtree? We know 2 is root of all nodes in left subtree. All nodes before 2 in post[] must be in left subtree. Now we know 1 is root, elements {8, 9, 4, 5, 2} are in left subtree, and the elements {6, 7, 3} are in right subtree.

```
                 1
               /   \
              /      \
     {8, 9, 4, 5, 2}     {6, 7, 3}
```

We recursively follow the above approach and get the following tree.

```
           1
         /   \
       2       3
     /  \     /  \
    4    5   6    7
   / \
  8   9
```

```
/* program for construction of full binary tree */
#include <stdio.h>
#include <stdlib.h>
```

```c
/* A binary tree node has data, pointer to left child
   and a pointer to right child */
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

// A utility function to create a node
struct node* newNode (int data)
{
    struct node* temp = (struct node *) malloc( sizeof(struct node) );

    temp->data = data;
    temp->left = temp->right = NULL;

    return temp;
}

// A recursive function to construct Full from pre[] and post[].
// preIndex is used to keep track of index in pre[].
// l is low index and h is high index for the current subarray in post[]
struct node* constructTreeUtil (int pre[], int post[], 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
    struct node* root = newNode ( pre[*preIndex] );
    ++*preIndex;

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

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

    // Use the index of element found in postorder to divide postorder array in
    // two parts. Left subtree and right subtree
    if (i <= h)
    {
        root->left = constructTreeUtil (pre, post, preIndex, l, i, size);
        root->right = constructTreeUtil (pre, post, preIndex, i + 1, h, size);
    }

    return root;
}

// The main function to construct Full Binary Tree from given preorder and
// postorder traversals. This function mainly uses constructTreeUtil()
struct node *constructTree (int pre[], int post[], int size)
{
    int preIndex = 0;
    return constructTreeUtil (pre, post, &preIndex, 0, size - 1, size);
}

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

// Driver program to test above functions
int main ()
{
    int pre[] = {1, 2, 4, 8, 9, 5, 3, 6, 7};
    int post[] = {8, 9, 4, 5, 2, 6, 7, 3, 1};
    int size = sizeof( pre ) / sizeof( pre[0] );

    struct node *root = constructTree(pre, post, size);

    printf("Inorder traversal of the constructed tree: \n");
    printInorder(root);

    return 0;
}
```

Output:

```
Inorder traversal of the constructed tree:
8 4 9 2 5 1 6 3 7
```