Construct a special tree from given preorder traversal

Given an array 'pre[]' that represents Preorder traversal of a spacial binary tree where every node has either 0 or 2 children. One more array 'preLN[]' is given which has only two possible values 'L' and 'N'. The value 'L' in 'preLN[]' indicates that the corresponding node in Binary Tree is a leaf node and value 'N' indicates that the corresponding node is non-leaf node. Write a function to construct the tree from the given two arrays.

Source: Amazon Interview Question

Example:

```
Input:  pre[] = {10, 30, 20, 5, 15},  preLN[] = {'N', 'N', 'L', 'L', 'L'}
Output: Root of following tree
          10
         /  \
        30   15
       /  \
      20   5
```

The first element in pre[] will always be root. So we can easily figure out root. If left subtree is empty, the right subtree must also be empty and preLN[] entry for root must be 'L'. We can simply create a node and return it. If left and right subtrees are not empty, then recursively call for left and right subtrees and link the returned nodes to root.

C

```
/* A program to construct Binary Tree from preorder traversal */
#include<stdio.h>

/* A binary tree node structure */
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

/* Utility function to create a new Binary Tree node */
struct node* newNode (int data)
{
    struct node *temp = new struct node;
    temp->data = data;
    temp->left = NULL;
    temp->right = NULL;
    return temp;
}

/* A recursive function to create a Binary Tree from given pre[]
   preLN[] arrays. The function returns root of tree. index_ptr is used
   to update index values in recursive calls. index must be initially
   passed as 0 */
struct node *constructTreeUtil(int pre[], char preLN[], int *index_ptr, int n)
{
    int index = *index_ptr; // store the current value of index in pre[]

    // Base Case: All nodes are constructed
    if (index == n)
        return NULL;

    // Allocate memory for this node and increment index for
    // subsequent recursive calls
    struct node *temp = newNode ( pre[index] );
    (*index_ptr)++;
```

```
    // If this is an internal node, construct left and right subtrees and link the subtrees
    if (preLN[index] == 'N')
    {
      temp->left  = constructTreeUtil(pre, preLN, index_ptr, n);
      temp->right = constructTreeUtil(pre, preLN, index_ptr, n);
    }

    return temp;
}

// A wrapper over constructTreeUtil()
struct node *constructTree(int pre[], char preLN[], int n)
{
    // Initialize index as 0. Value of index is used in recursion to maintain
    // the current index in pre[] and preLN[] arrays.
    int index = 0;

    return constructTreeUtil (pre, preLN, &index, n);
}


/* This function is used only for testing */
void printInorder (struct node* node)
{
    if (node == NULL)
        return;

    /* first recur on left child */
    printInorder (node->left);

    /* then print the data of node */
    printf("%d ", node->data);

    /* now recur on right child */
    printInorder (node->right);
}

/* Driver function to test above functions */
int main()
{
    struct node *root = NULL;

    /* Constructing tree given in the above figure
          10
         /  \
        30   15
       /  \
      20   5 */
    int pre[] = {10, 30, 20, 5, 15};
    char preLN[] = {'N', 'N', 'L', 'L', 'L'};
    int n = sizeof(pre)/sizeof(pre[0]);

    // construct the above tree
    root = constructTree (pre, preLN, n);

    // Test the constructed tree
    printf("Following is Inorder Traversal of the Constructed Binary Tree: \n");
    printInorder (root);

    return 0;
}
```

Java

```
// Java program to construct a binary tree from preorder traversal

// A Binary Tree node
class Node
{
```

```
    int data;
    Node left, right;

    Node(int item)
    {
        data = item;
        left = right = null;
    }
}

class Index
{
    int index = 0;
}

class BinaryTree
{
    Node root;
    Index myindex = new Index();

    /* A recursive function to create a Binary Tree from given pre[]
       preLN[] arrays. The function returns root of tree. index_ptr is used
       to update index values in recursive calls. index must be initially
       passed as 0 */
    Node constructTreeUtil(int pre[], char preLN[], Index index_ptr,
                                                     int n, Node temp)
    {
        // store the current value of index in pre[]
        int index = index_ptr.index;

        // Base Case: All nodes are constructed
        if (index == n)
            return null;

        // Allocate memory for this node and increment index for
        // subsequent recursive calls
        temp = new Node(pre[index]);
        (index_ptr.index)++;

        // If this is an internal node, construct left and right subtrees
        // and link the subtrees
        if (preLN[index] == 'N')
        {
            temp.left = constructTreeUtil(pre, preLN, index_ptr, n,
                                                               temp.left);
            temp.right = constructTreeUtil(pre, preLN, index_ptr, n,
                                                               temp.right);
        }

        return temp;
    }

    // A wrapper over constructTreeUtil()
    Node constructTree(int pre[], char preLN[], int n, Node node)
    {
        // Initialize index as 0. Value of index is used in recursion to
        // maintain the current index in pre[] and preLN[] arrays.
        int index = 0;

        return constructTreeUtil(pre, preLN, myindex, n, node);
    }

    /* This function is used only for testing */
    void printInorder(Node node)
    {
        if (node == null)
            return;

        /* first recur on left child */
        printInorder(node.left);

        /* then print the data of node */
        System.out.print(node.data + " ");
```

```
        /* now recur on right child */
        printInorder(node.right);
    }

    // driver function to test the above functions
    public static void main(String args[])
    {
        BinaryTree tree = new BinaryTree();
        int pre[] = new int[]{10, 30, 20, 5, 15};
        char preLN[] = new char[]{'N', 'N', 'L', 'L', 'L'};
        int n = pre.length;

        // construct the above tree
        Node mynode = tree.constructTree(pre, preLN, n, tree.root);

        // Test the constructed tree
        System.out.println("Following is Inorder Traversal of the"
                              + "Constructed Binary Tree: ");
        tree.printInorder(mynode);
    }
}

// This code has been contributed by Mayank Jaiswal
```

Output:

```
Following is Inorder Traversal of the Constructed Binary Tree:
20 30 5 10 15
```

Time Complexity: O(n)