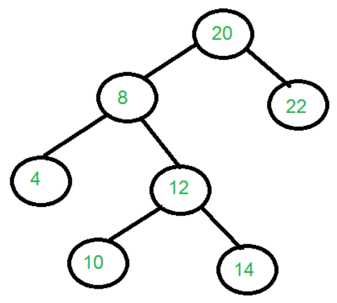
Construct a tree from Inorder and Level order traversals

Given inorder and level-order traversals of a Binary Tree, construct the Binary Tree. Following is an example to illustrate the problem.

```
Input: Two arrays that represent Inorder
       and level order traversals of a
       Binary Tree
in[]    = {4, 8, 10, 12, 14, 20, 22};
level[] = {20, 8, 22, 4, 12, 10, 14};

Output: Construct the tree represented
        by the two arrays.
        For the above two arrays, the
        constructed tree is shown in
        the diagram on right side
```



We strongly recommend to minimize the browser and try this yourself first.

The following post can be considered as a prerequisite for this.

Construct Tree from given Inorder and Preorder traversals

Let us consider the above example.

in[] = {4, 8, 10, 12, 14, 20, 22};
level[] = {20, 8, 22, 4, 12, 10, 14};

In a Levelorder sequence, the first element is the root of the tree. So we know '20' is root for given sequences. By searching '20' in Inorder sequence, we can find out all elements on left side of '20' are in left subtree and elements on right are in right subtree. So we know below structure now.

```
              20
            /    \
           /      \
 {4,8,10,12,14}  {22}
```

Let us call {4,8,10,12,14} as left subarray in Inorder traversal and {22} as right subarray in Inorder traversal.
In level order traversal, keys of left and right subtrees are not consecutive. So we extract all nodes from level order traversal which are in left subarray of Inorder traversal. To construct the left subtree of root, we recur for the extracted elements from level order traversal and left subarray of inorder traversal. In the above example, we recur for following two arrays.

```
// Recur for following arrays to construct the left subtree
In[]    = {4, 8, 10, 12, 14}
level[] = {8, 4, 12, 10, 14}
```

Similarly, we recur for following two arrays and construct the right subtree.

```
// Recur for following arrays to construct the right subtree
In[]    = {22}
level[] = {22}
```

Following is the implementation of the above approach.

C

```
/* program to construct tree using inorder and levelorder traversals */
#include <iostream>
using namespace std;

/* A binary tree node */
struct Node
```

```
{
    int key;
    struct Node* left, *right;
};

/* Function to find index of value in arr[start...end] */
int search(int arr[], int strt, int end, int value)
{
    for (int i = strt; i <= end; i++)
        if (arr[i] == value)
            return i;
    return -1;
}

// n is size of level[], m is size of in[] and m < n. This
// function extracts keys from level[] which are present in
// in[].  The order of extracted keys must be maintained
int *extrackKeys(int in[], int level[], int m, int n)
{
    int *newlevel = new int[m], j = 0;
    for (int i = 0; i < n; i++)
        if (search(in, 0, m-1, level[i]) != -1)
            newlevel[j] = level[i], j++;
    return newlevel;
}

/* function that allocates a new node with the given key  */
Node* newNode(int key)
{
    Node *node = new Node;
    node->key = key;
    node->left = node->right = NULL;
    return (node);
}

/* Recursive function to construct binary tree of size n from
   Inorder traversal in[] and Level Order traversal level[].
   inSrt and inEnd are start and end indexes of array in[]
   Initial values of inStrt and inEnd should be 0 and n -1.
   The function doesn't do any error checking for cases
   where inorder and levelorder do not form a tree */
Node* buildTree(int in[], int level[], int inStrt, int inEnd, int n)
{

    // If start index is more than the end index
    if (inStrt > inEnd)
        return NULL;

    /* The first node in level order traversal is root */
    Node *root = newNode(level[0]);

    /* If this node has no children then return */
    if (inStrt == inEnd)
        return root;

    /* Else find the index of this node in Inorder traversal */
    int inIndex = search(in, inStrt, inEnd, root->key);

    // Extract left subtree keys from level order traversal
    int *llevel  = extrackKeys(in, level, inIndex, n);

    // Extract right subtree keys from level order traversal
    int *rlevel  = extrackKeys(in + inIndex + 1, level, n-inIndex-1, n);

    /* construct left and right subtress */
    root->left = buildTree(in, llevel, inStrt, inIndex-1, n);
    root->right = buildTree(in, rlevel, inIndex+1, inEnd, n);

    // Free memory to avoid memory leak
    delete [] llevel;
    delete [] rlevel;

    return root;
}

/* Uti;ity function to print inorder traversal of binary tree */
void printInorder(Node* node)
{
    if (node == NULL)
       return;
    printInorder(node->left);
    cout << node->key << " ";
    printInorder(node->right);
}

/* Driver program to test above functions */
int main()
{
    int in[]    = {4, 8, 10, 12, 14, 20, 22};
    int level[] = {20, 8, 22, 4, 12, 10, 14};
    int n = sizeof(in)/sizeof(in[0]);
    Node *root = buildTree(in, level, 0, n - 1, n);

    /* Let us test the built tree by printing Insorder traversal */
    cout << "Inorder traversal of the constructed tree is \n";
    printInorder(root);

    return 0;
}
```

Java

```
// Java program to construct a tree from level order and
// and inorder traversal

// A binary tree node
class Node
{
    int data;
    Node left, right;

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

    public void setLeft(Node left)
    {
        this.left = left;
    }

    public void setRight(Node right)
    {
        this.right = right;
    }
}

class Tree
{
    Node root;

    Node buildTree(int in[], int level[])
    {
        Node startnode = null;
        return constructTree(startnode, level, in, 0, in.length - 1);
    }

    Node constructTree(Node startNode, int[] levelOrder, int[] inOrder,
            int inStart, int inEnd)
    {

        // if start index is more than end index
        if (inStart > inEnd)
            return null;

        if (inStart == inEnd)
            return new Node(inOrder[inStart]);

        boolean found = false;
        int index = 0;

        // it represents the index in inOrder array of element that
        // appear first in levelOrder array.
        for (int i = 0; i < levelOrder.length - 1; i++)
        {
            int data = levelOrder[i];
            for (int j = inStart; j < inEnd; j++)
            {
                if (data == inOrder[j])
                {
                    startNode = new Node(data);
                    index = j;
                    found = true;
                    break;
                }
            }
            if (found == true)
                break;
        }

        //elements present before index are part of left child of startNode.
        //elements present after index are part of right child of startNode.
        startNode.setLeft(constructTree(startNode, levelOrder, inOrder,
                                                    inStart, index - 1));
        startNode.setRight(constructTree(startNode, levelOrder, inOrder,
                                                     index + 1, inEnd));

        return startNode;
    }

    /* Utility function to print inorder traversal of binary tree */
    void printInorder(Node node)
    {
        if (node == null)
            return;
        printInorder(node.left);
        System.out.print(node.data + " ");
        printInorder(node.right);
    }

    // Driver program to test the above functions
    public static void main(String args[])
    {
        Tree tree = new Tree();
        int in[] = new int[]{4, 8, 10, 12, 14, 20, 22};
        int level[] = new int[]{20, 8, 22, 4, 12, 10, 14};
        int n = in.length;
        Node node = tree.buildTree(in, level);

        /* Let us test the built tree by printing Inorder traversal */
        System.out.print("Inorder traversal of the constructed tree is ");
        tree.printInorder(node);
    }
}

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

Output:

```
Inorder traversal of the constructed tree is
4 8 10 12 14 20 22
```

An upper bound on time complexity of above method is O(n^3). In the main recursive function, extractNodes() is called which takes O(n^2)

time.

The code can be optimized in many ways and there may be better solutions. Looking for improvements and other optimized approaches to solve this problem.