```java
/*

    Time complexity: O(N^2)
    Space complexity: O(H)

    where N is the number of nodes in the input tree
    and H is the height of the input tree

 */

public class Solution {

        private static BinaryTreeNode<Integer> buildTreeHelper(int[] preOrder, int preStart, int
preEnd, int[] inOrder, int inStart, int inEnd) {
            if (preStart > preEnd || inStart > inEnd) {
                return null;
            }

            int rootVal = preOrder[preStart];
            BinaryTreeNode<Integer> root = new BinaryTreeNode<>(rootVal);

            // Find root element index from inOrder array
            int k = 0;
            for (int i = inStart; i <= inEnd; i++) {
                if (rootVal == inOrder[i]) {
                    k = i;
                    break;
                }
            }

            root.left = buildTreeHelper(preOrder, preStart + 1, preStart + (k - inStart), inOrder,
inStart, k - 1);
            root.right = buildTreeHelper(preOrder, preStart + (k - inStart) + 1, preEnd, inOrder, k +
1, inEnd);

            return root;
        }

        public static BinaryTreeNode<Integer> buildTree(int[] preOrder, int[] inOrder) {
            int n = preOrder.length;

            int preStart = 0;
            int preEnd = n - 1;
            int inStart = 0;
            int inEnd = n - 1;

            return buildTreeHelper(preOrder, preStart, preEnd, inOrder, inStart, inEnd);
        }

}
```