```
        int rMax = posOrder(head.right, record);
        int maxFromRight = record[0];
        int curNodeMax = maxfromLeft + maxFromRight + 1;
        record[0] = Math.max(maxfromLeft, maxFromRight) + 1;
        return Math.max(Math.max(lMax, rMax), curNodeMax);
    }
```

先序、中序和后序数组两两结合重构二叉树

【题目】

已知一棵二叉树的所有节点值都不同，给定这棵二叉树正确的先序、中序和后序数组。请分别用三个函数实现任意两种数组结合重构原来的二叉树，并返回重构二叉树的头节点。

【难度】

先序与中序结合　士　★☆☆☆
中序与后序结合　士　★☆☆☆
先序与后序结合　尉　★★☆☆

【解答】

先序与中序结合重构二叉树的过程如下：

1. 先序数组中最左边的值就是树的头节点值，记为 h，并用 h 生成头节点，记为 head。然后在中序数组中找到 h，假设位置是 i。那么在中序数组中，i 左边的数组就是头节点左子树的中序数组，假设长度为 l，则左子树的先序数组就是先序数组中 h 往右长度也为 l 的数组。

比如：先序数组为[1,2,4,5,8,9,3,6,7]，中序数组为[4,2,8,5,9,1,6,3,7]，二叉树头节点的值是 1，在中序数组中找到 1 的位置，1 左边的数组为[4,2,8,5,9]，是头节点左子树的中序数组，长度为 5；先序数组中 1 的右边长度也为 5 的数组为[2,4,5,8,9]，就是左子树的先序数组。

2. 用左子树的先序和中序数组，递归整个过程建立左子树，返回的头节点记为 left。

3. i 右边的数组就是头节点右子树的中序数组，假设长度为 r。先序数组中右侧等长的部分就是头节点右子树的先序数组。

比如步骤 1 的例子，中序数组中 1 右边的数组为[6,3,7]，长度为 3；先序数组右侧等长的部分为[3,6,7]，它们分别为头节点右子树的中序和先序数组。

4．用右子树的先序和中序数组，递归整个过程建立右子树，返回的头节点记为 right。

5．把 head 的左孩子和右孩子分别设为 left 和 right，返回 head，过程结束。

如果二叉树的节点数为 N，在中序数组中找到位置 i 的过程可以用哈希表来实现，这样整个过程时间复杂度为 $O(N)$。

具体过程请参看如下代码中的 preInToTree 方法。

```
public Node preInToTree(int[] pre, int[] in) {
        if (pre == null || in == null) {
                return null;
        }
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < in.length; i++) {
                map.put(in[i], i);
        }
        return preIn(pre, 0, pre.length - 1, in, 0, in.length - 1, map);
}

public Node preIn(int[] p, int pi, int pj, int[] n, int ni, int nj,
                HashMap<Integer, Integer> map) {
        if (pi > pj) {
                return null;
        }
        Node head = new Node(p[pi]);
        int index = map.get(p[pi]);
        head.left = preIn(p, pi + 1, pi + index - ni, n, ni, index - 1, map);
        head.right = preIn(p, pi + index - ni + 1, pj, n, index + 1, nj, map);
        return head;
}
```

中序和后序重构的过程与先序和中序的过程类似。先序和中序的过程是用先序数组最左的值来对中序数组进行划分，因为这是头节点的值。后序数组中头节点的值是后序数组最右的值，所以用后序最右的值来划分中序数组即可。

具体过程请参看如下代码中的 inPosToTree 方法。

```
public Node inPosToTree(int[] in, int[] pos) {
        if (in == null || pos == null) {
                return null;
        }
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < in.length; i++) {
                map.put(in[i], i);
        }
        return inPos(in, 0, in.length - 1, pos, 0, pos.length - 1, map);
}

public Node inPos(int[] n, int ni, int nj, int[] s, int si, int sj,
```

```
                HashMap<Integer, Integer> map) {
        if (si > sj) {
                return null;
        }
        Node head = new Node(s[sj]);
        int index = map.get(s[sj]);
        head.left = inPos(n, ni, index - 1, s, si, si + index - ni - 1, map);
        head.right = inPos(n, index + 1, nj, s, si + index - ni, sj - 1, map);
        return head;
}
```

先序和后序结合重构二叉树。要求面试者首先分析出节点值都不同的二叉树，即便得到了正确的先序与后序数组，在大多数情况下也不能通过这两个数组把原来的树重构出来。这是因为很多结构不同的树中，先序与后序数组是一样的，比如，头节点为 1、左孩子为 2、右孩子为 null 的树，先序数组为[1,2]，后序数组为[2,1]。而头节点为 1、左孩子为 null、右孩子为 2 的树也是同样的结果。然后需要分析出什么样的树可以被先序和后序数组重建，如果一棵二叉树除叶节点之外，其他所有的节点都有左孩子和右孩子，只有这样的树才可以被先序和后序数组重构出来。最后才是通过划分左右子树各自的先序与后序数组的方式重建整棵树，具体过程请参看如下代码中的 prePosToTree 方法。

```
// 每个节点的孩子数都为 0 或 2 的二叉树才能被先序与后序重构出来
public Node prePosToTree(int[] pre, int[] pos) {
        if (pre == null || pos == null) {
                return null;
        }
        HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();
        for (int i = 0; i < pos.length; i++) {
                map.put(pos[i], i);
        }
        return prePos(pre, 0, pre.length - 1, pos, 0, pos.length - 1, map);
}

public Node prePos(int[] p, int pi, int pj, int[] s, int si, int sj,
                HashMap<Integer, Integer> map) {
        Node head = new Node(s[sj--]);
        if (pi == pj) {
                return head;
        }
        int index = map.get(p[++pi]);
        head.left = prePos(p, pi, pi + index - si, s, si, index, map);
        head.right = prePos(p, pi + index - si + 1, pj, s, index + 1, sj, map);
        return head;
}
```