249. 统计前面比自己小的数的个数

给定一个整数数组（下标由 0 到 n-1， n 表示数组的规模，取值范围由 0 到10000）。对于数组中的每个 `ai` 元素，请计算 `ai` 前的数中比它小的元素的数量。

We suggest you finish problem Segment Tree Build, Segment Tree Query II and Count of Smaller Number first.

样例:对于数组`[1,2,7,8,5]` ，返回 `[0,1,2,3,2]`

思路：

1. 如何找到问题的切入点，对于每一个元素A[i]我们查询比它小数，转换成区间的查询就是查询在它前面的数当中有多少在区间[0, A[i] - 1]当中。
2. 因此我们可以为**0-10000**区间建树，并将所有区间count设为0。每一个最小区间（即叶节点）的count代表到目前为止该数的数量。
   然后开始遍历数组，遇到A[i]时，去查`0 ～ A[i]-1`区间的count即这个区间中有多少数存在，这就是比A[i]小的数的数量。查完后将A[i]区间的count加1即可，也就是把A[i]插入到线段树`i`的位置上。
3. 具体举例：A = `[1,2,7,8,5]`, 最大值为8.我们得到一棵空的SegmentTree，对应的前9个位置的区间是`[0, 0, 0, 0, 0, 0, 0, 0, 0]`。即该辅助数组第**5**个元素代表数组中**5**出现的次数。
   A[0] = 1, 查区间[0, 0]的和, 得到0，然后在1的地方加1，得到`[0, 1, 0, 0, 0, 0, 0, 0, 0]`
   A[1] = 2, 查区间[0, 1]的和, 得到1，然后在2的地方加1，得到`[0, 1, 1, 0, 0, 0, 0, 0, 0]`
   A[2] = 7, 查区间[0, 6]的和, 得到2，然后在7的地方加1，得到`[0, 1, 1, 0, 0, 0, 0, 1, 0]`
   A[3] = 8, 查区间[0, 7]的和, 得到3，然后在8的地方加1，得到`[0, 1, 1, 0, 0, 0, 0, 1, 1]`
   A[4] = 5, 查区间[0, 4]的和, 得到2，然后在5的地方加1，得到`[0, 1, 1, 0, 0, 1, 0, 1, 1]`
   所以我们得到结果序列为：`[0, 1, 2, 3, 2]`

总结：本题用到了区间单点修改和区间和查询两个主要的线段树操作。

```java
public class Solution {
    /**
     * @param A: an integer array
     * @return: A list of integers includes the index of the first number and the index of the last number
     */
    public List<Integer> countOfSmallerNumberII(int[] A) {
        // write your code here
        root = build(0, 10000);
        ArrayList<Integer> ans = new ArrayList<Integer>();
        int res;
        for( int i = 0; i < A.length; i++ ){
            res = 0;
            if( A[i] > 0 ){
                res = query(root, 0, A[i] - 1);
            }
            modify(root, A[i], 1);
            ans.add(res);
        }
        return ans;
    }

```

```
    class SegmentTreeNode{
        public int start, end;
        public int count;
        public SegmentTreeNode left, right;
        public SegmentTreeNode(int start, int end, int count){
            this.start = start;
            this.end = end;
            this.count = count;
            this.left = this.right = null;
        }
    }
    SegmentTreeNode root;
    public SegmentTreeNode build(int start, int end){
        if( start > end )
            return null;
        SegmentTreeNode node = new SegmentTreeNode(start, end, 0);
        if( start != end ){
            int mid = (start + end)/2;
            node.left = build(start, mid);
            node.right = build(mid + 1, end);
        }
        return node;
    }
    public int query(SegmentTreeNode root, int start, int end){
        if( start == root.start && end == root.end )
            return root.count;
        int mid = (root.start + root.end)/2;
        int leftCount = 0, rightCount = 0;
        if( start <= mid ){
            leftCount = query( root.left, start, Math.min(end, mid) );
        }
        if( mid < end ){
            rightCount = query( root.right, Math.max(mid+1, start), end );
        }
        return leftCount + rightCount;
    }
    public void modify(SegmentTreeNode root, int index, int value){
        if( root.start == index && root.end == index ){
            root.count += value;
            return;
        }
        int mid = (root.start + root.end)/2;
        if( index <= mid && root.start <= index  ){
            modify(root.left, index, value);
        }
        if( mid < index && index <= root.end ){
            modify(root.right, index, value);
        }
        root.count = root.left.count + root.right.count;
    }
}
```