# LeetCode\_315\_CountOfSmallerNumbersAfterSelf—Hard

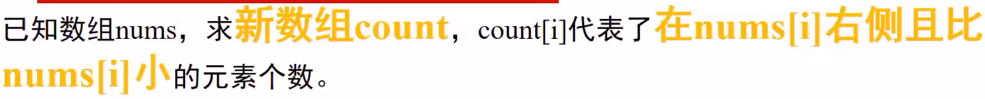
## 题目描述

LeetCode\_315\_CountOfSmallerNumbersAfterSelf—Hard

难度：**Hard**

<https://leetcode.com/problems/count-of-smaller-numbers-after-self/description/>

You are given an integer array nums and you have to return a new counts array. The counts array has the property where counts[i] is the number of smaller elements to the right of **nums[i]**.



Example:

Input: [5,2,6,1]

Output: [2,1,1,0]

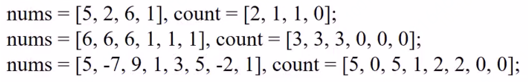
Explanation:

To the right of 5 there are 2 smaller elements (2 and 1).

To the right of 2 there is only 1 smaller element (1).

To the right of 6 there is 1 smaller element (1).

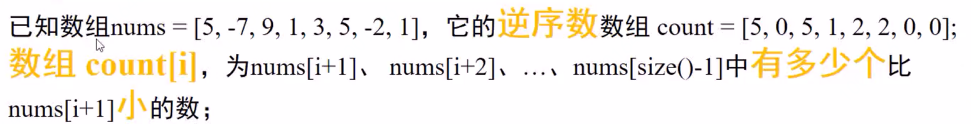
To the right of 1 there is 0 smaller element.

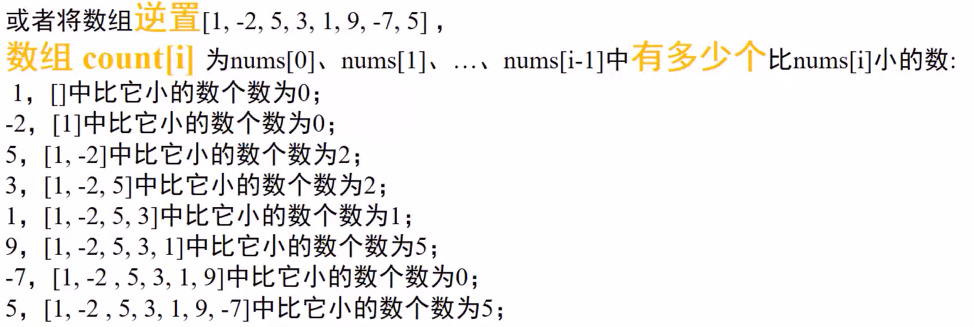


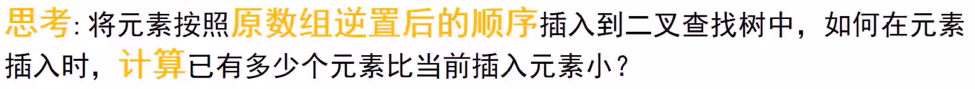
## 解决思路

解决方法1：**归并**；(遇到时再说)

解决方法2：**二叉查找树**；(又称二叉排序树)重点介绍该方法。

**需要逆置**：





TreeNodeWithCount : count用来记录左子树的节点数量。

/\*\*

\* 定义一个带有count属性的二叉树结构

\*/

class TreeNodeWithCount{

int value;

int count;/\*用于记录当前节点的左子树的节点数目

(包括叶子节点和非叶子节点)；意义：当前比该节点值小的节点数目\*/

TreeNodeWithCount left;

TreeNodeWithCount right;

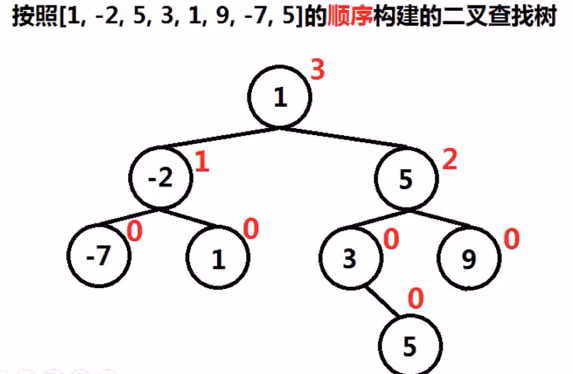
public TreeNodeWithCount(int value){

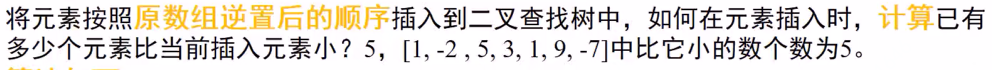
this.value = value;

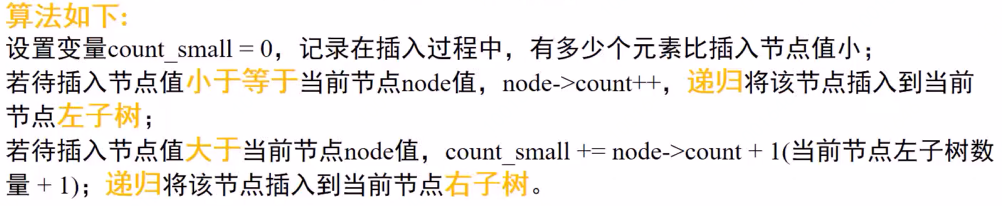
}

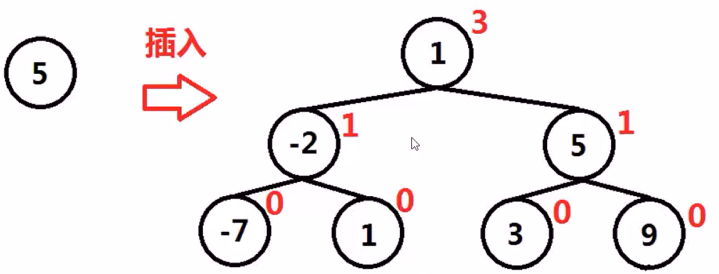
}



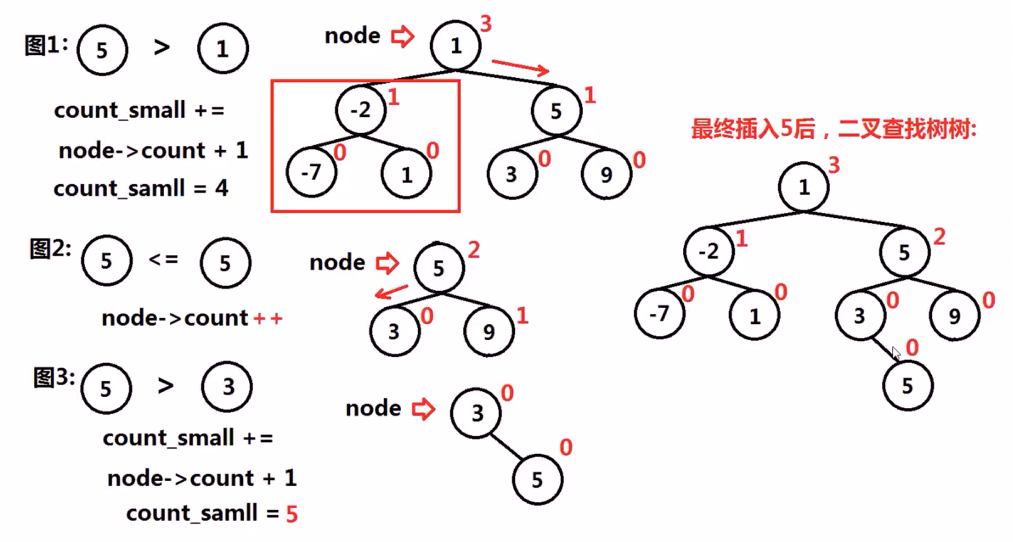








示例：



## Java代码实现

public class LeetCode\_315\_CountOfSmallerNumbersAfterSelf {

/\*\*

\* 测试

\*/

@Test

public void test(){

// int[] nums = {5,2,6,1,32,4,3,4,5,3,-2};

int[] nums = new int[]{9,8,7,6,5,4,3,2,1};

System.out.println("nums :\t"+ Arrays.toString(nums));

List<Integer> result = countSmaller(nums);

System.out.println("rightSmallerNums :\t"+ result.toString());

}

/\*\*

\* 定义一个带有count属性的二叉树结构

\*/

class **BSTNodeWithCount**{//BinarySearchTree二叉查找数

int value;

int count;/\*用于记录当前节点的左子树的节点数目

(包括叶子节点和非叶子节点)；意义：当前比该节点值小的节点数目\*/

BSTNodeWithCount left;

BSTNodeWithCount right;

public BSTNodeWithCount(int value){

this.value = value;

}

}

/\*\*

\* 首先将nums元素转化为节点

\*/

public List<Integer> **countSmaller**(int[] nums) {

List<Integer> rightSmallerNumsReverse = new ArrayList<Integer>();//存放倒序统计结果

List<Integer> rightSmallerNums = new ArrayList<Integer>();//存放正序统计结果

List<BSTNodeWithCount> nodeList = new ArrayList<BSTNodeWithCount>();

for(int i = nums.length - 1;i>= 0;i--){//按照倒序将元素转化为节点

nodeList.add(new BSTNodeWithCount(nums[i]));

}

rightSmallerNumsReverse.add(0);//第一个节点perResultSmaller为0

for(int i = 1;i < nodeList.size();i++){//从1开始

int[] perLeftSmallerNum = {0};//用数组为了递归传递修改值

**BST\_insert(nodeList.get(0),nodeList.get(i),perLeftSmallerNum);//向二叉排序树中插入新节点**

rightSmallerNumsReverse.add(perLeftSmallerNum[0]);//保存当前节点对应的result

}

for(int i = nodeList.size() - 1;i >= 0;i--){

rightSmallerNums.add(rightSmallerNumsReverse.get(i));//将reverseResults再倒置成正序，与nums保持一致

}

return rightSmallerNums;

}

/\*\*

\* 这是一个递归方法，向以root为根节点的二叉排序树中插入新的节点，

\* 并统计该当前二叉排序树中小于当前节点的节点数目

\*/

public void **BST\_insert**(BSTNodeWithCount root,BSTNodeWithCount insert\_node,int[] perLeftSmallerNum){

if(insert\_node.value <= root.value){//小于等于根节点的置于左子树

root.count ++;

if(root.left != null){//若左子树存在，则递归插入，否则当前节点充当左子树

BST\_insert(root.left,insert\_node,perLeftSmallerNum);

}else{

root.left = insert\_node;

}

}else{

perLeftSmallerNum[0] += root.count + 1;/\*\*重要：当前根节点的左子树的节点值肯定比带插入节点值小，

而根节点的count属性表示的就是左子树的节点数目\*/

if(root.right != null){

BST\_insert(root.right,insert\_node,perLeftSmallerNum);//若右子树存在，则递归插入，否则当前节点充当右子树

}else{

root.right = insert\_node;

}

}

}

}

## C语言代码实现

