Count of Smaller Numbers After Self

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:

```
Given nums = [5, 2, 6, 1]

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.
```

Return the array `[2, 1, 1, 0]`.

Solution 1

The smaller numbers on the right of a number are exactly those that jump from its right to its left during a stable sort. So I do mergesort with added tracking of those right-to-left jumps.

Update, new version

```python
def countSmaller(self, nums):
    def sort(enum):
        half = len(enum) / 2
        if half:
            left, right = sort(enum[:half]), sort(enum[half:])
            for i in range(len(enum))[::-1]:
                if not right or left and left[-1][1] > right[-1][1]:
                    smaller[left[-1][0]] += len(right)
                    enum[i] = left.pop()
                else:
                    enum[i] = right.pop()
        return enum
    smaller = [0] * len(nums)
    sort(list(enumerate(nums)))
    return smaller
```

Old version

```python
def countSmaller(self, nums):
    def sort(enum):
        half = len(enum) / 2
        if half:
            left, right = sort(enum[:half]), sort(enum[half:])
            m, n = len(left), len(right)
            i = j = 0
            while i < m or j < n:
                if j == n or i < m and left[i][1] <= right[j][1]:
                    enum[i+j] = left[i]
                    smaller[left[i][0]] += j
                    i += 1
                else:
                    enum[i+j] = right[j]
                    j += 1
        return enum
    smaller = [0] * len(nums)
    sort(list(enumerate(nums)))
    return smaller
```

written by StefanPochmann original link here

Solution 2

Traverse from the back to the beginning of the array, maintain an sorted array of numbers have been visited. Use findIndex() to find the first element in the sorted array which is larger or equal to target number. For example, [5,2,3,6,1], when we reach 2, we have a sorted array[1,3,6], findIndex() returns 1, which is the index where 2 should be inserted and is also the number smaller than 2. Then we insert 2 into the sorted array to form [1,2,3,6].

```java
public List<Integer> countSmaller(int[] nums) {
    Integer[] ans = new Integer[nums.length];
    List<Integer> sorted = new ArrayList<Integer>();
    for (int i = nums.length - 1; i >= 0; i--) {
        int index = findIndex(sorted, nums[i]);
        ans[i] = index;
        sorted.add(index, nums[i]);
    }
    return Arrays.asList(ans);
}
private int findIndex(List<Integer> sorted, int target) {
    if (sorted.size() == 0) return 0;
    int start = 0;
    int end = sorted.size() - 1;
    if (sorted.get(end) < target) return end + 1;
    if (sorted.get(start) >= target) return 0;
    while (start + 1 < end) {
        int mid = start + (end - start) / 2;
        if (sorted.get(mid) < target) {
            start = mid + 1;
        } else {
            end = mid;
        }
    }
    if (sorted.get(start) >= target) return start;
    return end;
}
```

Due to the O(n) complexity of ArrayList insertion, the total runtime complexity is not very fast, but anyway it got AC for around 53ms.

written by mayanist original link here

Solution 3

Every node will maintain a val **sum** recording the total of number on it's left bottom side, **dup** counts the duplication. For example, [3, 2, 2, 6, 1], from back to beginning,we would have:

```
                1(0, 1)
                     \
                     6(3, 1)
                     /
                  2(0, 2)
                       \
                        3(0, 1)
```

When we try to insert a number, the total number of smaller number would be **adding dup and sum of the nodes where we turn right** . for example, if we insert 5, it should be inserted on the way down to the right of 3, the nodes where we turn right is 1(0,1), 2,(0,2), 3(0,1), so the answer should be (0 + 1)+(0 + 2)+ (0 + 1) = 4

if we insert 7, the right-turning nodes are 1(0,1), 6(3,1), so answer should be (0 + 1) + (3 + 1) = 5

```
public class Solution {
    class Node {
        Node left, right;
        int val, sum, dup = 1;
        public Node(int v, int s) {
            val = v;
            sum = s;
        }
    }
    public List<Integer> countSmaller(int[] nums) {
        Integer[] ans = new Integer[nums.length];
        Node root = null;
        for (int i = nums.length - 1; i >= 0; i--) {
            root = insert(nums[i], root, ans, i, 0);
        }
        return Arrays.asList(ans);
    }
    private Node insert(int num, Node node, Integer[] ans, int i, int preSum) {
        if (node == null) {
            node = new Node(num, 0);
            ans[i] = preSum;
        } else if (node.val == num) {
            node.dup++;
            ans[i] = preSum + node.sum;
        } else if (node.val > num) {
            node.sum++;
            node.left = insert(num, node.left, ans, i, preSum);
        } else {
            node.right = insert(num, node.right, ans, i, preSum + node.dup + node
.sum);
        }
        return node;
    }
}
```

written by mayanist original link here