# 0315. 计算右侧小于当前元素的个数

ITCharge 大约 4 分钟

- 标签：树状数组、线段树、数组、二分查找、分治、有序集合、归并排序
- 难度：困难

## 题目链接

- 0315. 计算右侧小于当前元素的个数 - 力扣

## 题目大意

**描述**：给定一个整数数组 $nums$ 。

**要求**：返回一个新数组 $counts$ 。其中 $counts[i]$ 的值是 $nums[i]$ 右侧小于 $nums[i]$ 的元素的数量。

**说明**：

- $1 \leq nums.length \leq 10^5$。
- $-10^4 \leq nums[i] \leq 10^4$。

**示例**：

- 示例 1：

```py
输入：nums = [5,2,6,1]
输出：[2,1,1,0]
解释：
5 的右侧有 2 个更小的元素 (2 和 1)
2 的右侧仅有 1 个更小的元素 (1)
6 的右侧有 1 个更小的元素 (1)
1 的右侧有 0 个更小的元素
```

- 示例 2：

```py
输入：nums = [-1]
输出：[0]
```

# 解题思路

## 思路 1：归并排序

在使用归并排序对数组进行排序时，每当遇到 $left\_nums[left\_i] \leq right\_nums[right\_i]$ 时，意味着：在合并前，左子数组当前元素 $left\_nums[left\_i]$ 右侧一定有 $left\_i$ 个元素比 $left\_nums[left\_i]$ 小。则我们可以在归并排序的同时，记录 $nums[i]$ 右侧小于 $nums[i]$ 的元素的数量。

1. 将元素值、对应下标、右侧小于 nums[i] 的元素的数量存入数组中。
2. 对其进行归并排序。
3. 当遇到 $left\_nums[left\_i] \leq right\_nums[right\_i]$ 时，记录 $left\_nums[left\_i]$ 右侧比 $left\_nums[left\_i]$ 小的元素数量，即：`left_nums[left_i][2] += right_i` 。
4. 当合并时 $left\_nums[left\_i]$ 仍有剩余时，说明 $left\_nums[left\_i]$ 右侧有 $right\_i$ 个小于 $left\_nums[left\_i]$ 的元素，记录下来，即：`left_nums[left_i][2] += right_i` 。
5. 根据下标及右侧小于 $nums[i]$ 的元素数量，组合出答案数组，并返回答案数组。

## 思路 1：代码

```py
class Solution:
    # 合并过程
    def merge(self, left_nums, right_nums):
        nums = []
        left_i, right_i = 0, 0
        while left_i < len(left_nums) and right_i < len(right_nums):
            # 将两个有序子数组中较小元素依次插入到结果数组中
            if left_nums[left_i] <= right_nums[right_i]:
                nums.append(left_nums[left_i])
                # left_nums[left_i] 右侧有 right_i 个比 left_nums[left_i] 小的
                left_nums[left_i][2] += right_i
                left_i += 1
            else:
                nums.append(right_nums[right_i])
                right_i += 1
```

```
        # 如果左子数组有剩余元素，则将其插入到结果数组中
        while left_i < len(left_nums):
            nums.append(left_nums[left_i])
            # left_nums[left_i] 右侧有 right_i 个比 left_nums[left_i] 小的
            left_nums[left_i][2] += right_i
            left_i += 1

        # 如果右子数组有剩余元素，则将其插入到结果数组中
        while right_i < len(right_nums):
            nums.append(right_nums[right_i])
            right_i += 1

        # 返回合并后的结果数组
        return nums

    # 分解过程
    def mergeSort(self, nums) :
        # 数组元素个数小于等于 1 时，直接返回原数组
        if len(nums) <= 1:
            return nums

        mid = len(nums) // 2                          # 将数组从中间位置分为左右两个
数组
        left_nums = self.mergeSo     ums[0: mid])     # 递归将左子数组进行分解和排序
        right_nums =  self.mergeSort(nums[mid:])      # 递归将右子数组进行分解和排序
        return self.merge(left_nums, right_nums)      # 把当前数组组中有序子数组逐层
向上，进行两两合并


    def countSmaller(self, nums: List[int]) -> List[int]:
        size = len(nums)

        # 将元素值、对应下标、右侧小于 nums[i] 的元素的数量存入数组中
        nums = [[num, i, 0] for i, num in enumerate(nums)]
        nums = self.mergeSort(nums)
        ans = [0 for _ in range(size)]

        for num in nums:
            ans[num[1]] = num[2]

        return ans
```

## 思路 1：复杂度分析

- **时间复杂度**：$O(n \times \log n)$。
- **空间复杂度**：$O(n)$。

## 思路 2：树状数组

1. 首先对数组进行离散化处理。把原始数组中的数据映射到 $[0, len(nums) - 1]$ 这个区间。
2. 然后逆序顺序从数组 $nums$ 中遍历元素 $nums[i]$。
    1. 计算其离散化后的排名 $index$，查询比 $index$ 小的数有多少个。将其记录到答案数组的对应位置 $ans[i]$ 上。
    2. 然后在树状数组下标为 $index$ 的位置上，更新值为 1。
3. 遍历完所有元素，最后输出答案数组 $ans$ 即可。

## 思路 2：代码

```py
import bisect

class BinaryIndexTree:

    def __init__(self, n):
        self.size = n
        self.tree = [0 for _ in range(n + 1)]

    def lowbit(self, index):
        return index & (-index)

    def update(self, index, delta):
        while index <= self.size:
            self.tree[index] += delta
            index += self.lowbit(index)

    def query(self, index):
        res = 0
        while index > 0:
            res += self.tree[index]
            index -= self.lowbit(index)
```

```
        return res

class Solution:
    def countSmaller(self, nums: List[int]) -> List[int]:
        size = len(nums)
        if size == 0:
            return []
        if size == 1:
            return [0]

        # 离散化
        sort_nums = list(set(nums))
        sort_nums.sort()
        size_s = len(sort_nums)
        bit = BinaryIndexTree(size_s)

        ans = [0 for _ in range(size)]
        for i in range(size - 1, -1, -1):
            index = bisect.bisect_left(sort_nums, nums[i]) + 1
            ans[i] = bit.query(index - 1)
            bit.update(index, 1)

        return ans
```

## 思路 2：复杂度分析

- **时间复杂度**：$O(n \times \log n)$。
- **空间复杂度**：$O(n)$。