经典的逆序对计数算法是基于**归并排序 (Merge Sort)** 的分治思想。其核心在于将大问题分解为小问题，并在“合并”已排序的子数组时计算跨越子数组边界的逆序对。

1. **分 (Divide):** 将原始数组递归地分成两半，直到每个子数组只包含一个元素。

2. **治 (Conquer):** 对两个子数组分别递归地计算其内部的重要逆序对。

3. **合 (Combine) / 计数**：在将两个已排序的子数组（例如，左半部分 `left_arr` 和右半部分 `right_arr`）归并成一个完整的排序数组的过程中，计算跨越这两个子数组的重要逆序对。

“合”阶段的核心修改

假设 `left_arr` 为 L，`right_arr` 为 R，它们都是已经排序的。我们使用两个指针 p_L 和 p_R 分别指向 L 和 R 的当前元素。

为了计算满足 $L[i] > 2 \times R[j]$ 的对数，在归并过程中，我们对 L 中的每个元素 $L[i]$，在 R 中查找满足条件的元素数量。由于 R 是已排序的，我们可以利用一个辅助指针（例如 p_R）在 R 中线性扫描。

Algorithm 1 计算重要逆序对的归并函数 (`MergeAndCountImportantInversions`)

```
function MergeAndCountImportantInversions(arr, left_start, mid, right_end)
    count = 0
    temp_arr = []                                        ▷ 用于归并的临时数组
    p_R = mid + 1                                        ▷ 右子数组的起始指针
    for i from left_start to mid do                      ▷ 遍历左子数组的每个元素 arr[i]
        while p_R ≤ right_end and arr[i] > 2 × arr[p_R] do
            p_R ← p_R + 1                                ▷ 移动 p_R 直到不满足条件或越界
        end while
        count ← count + (p_R − (mid + 1))                ▷ 累加与 arr[i] 构成重要逆序对的右子数组元素数量
    end for
                                                         ▷ 执行标准的归并排序步骤，将两个子数组合并并排序
    p_L = left_start
    p_R = mid + 1
    while p_L ≤ mid or p_R ≤ right_end do
        if p_L ≤ mid and (p_R > right_end or arr[p_L] ≤ arr[p_R]) then
            temp_arr.Add(arr[p_L])
            p_L ← p_L + 1
        else
            temp_arr.Add(arr[p_R])
            p_R ← p_R + 1
        end if
    end while
                                                         ▷ 将排好序的元素复制回原数组
    for k from 0 to temp_arr.Length - 1 do
        arr[left_start + k] ← temp_arr[k]
    end for
     return count
end function
```

Algorithm 2 计算重要逆序对的主函数 (`CountImportantInversions`)

```
function CountImportantInversions(arr, start, end)
    if start ≥ end then return 0
    end if
    mid = ⌊(start + end)/2⌋
    total_count = 0
    total_count ← total_count + CountImportantInversions(arr, start, mid)
    total_count ← total_count + CountImportantInversions(arr, mid + 1, end)
    total_count ← total_count + MergeAndCountImportantInversions(arr, start, mid, end)
     return total_count
end function
```

时间复杂度分析

该算法的时间复杂度为 $O(n \log n)$。

- **分 (Divide):** 每次递归将问题规模减半，形成 $\log n$ 层递归。
- **合 (Combine) / 计数**：在每一层递归中，`MergeAndCountImportantInversions` 函数的主要操作包括两个部分：
  - 计算重要逆序对的部分：外层循环遍历左子数组 ($O(|L|)$ 次)，内层 `while` 循环的指针 p_R 在整个归并过程中是单调递增的，它最多遍历右子数组一次 ($O(|R|)$ 次)。因此，这部分的总时间复杂度为 $O(|L| + |R|)$。
  - 标准归并排序部分：同样是 $O(|L| + |R|)$。

  所以，在每一层递归中，所有归并操作的总时间复杂度为 $O(n)$。
- **总时间复杂度**：递归关系为 $T(n) = 2T(n/2) + O(n)$。根据主定理 (Master Theorem)，该算法的总时间复杂度为 $O(n \log n)$。