剑指 Offer 51. 数组中的逆序对

Thursday, June 9, 2022 11:36 AM

https://leetcode.cn/problems/shu-zu-zhong-de-ni-xu-dui-lcof/

困难

在数组中的两个数字，如果前面一个数字 **大于** 后面的数字，则这两个数字组成一个逆序对。

输入一个数组，求出这个数组中的逆序对的总数。

示例 1:

输入: [7,5,6,4]

输出: 5

0 <= 数组长度 <= 50000

我的思路：

双层for循环，遍历所有逆序对。

（这么简单？不对头啊，果然 超时。）

code：

```
class Solution {
    public int reversePairs(int[] nums) {
        if(nums.length<2){
            return 0;
        }
        int count = 0;
        for(int i=0; i<nums.length-1; i++){
            for(int j=i+1; j<nums.length; j++){
                if(nums[i]>nums[j]){
                    count++;
                }
            }
        }
        return count;
    }
}
```

正解：

首先：什么是归并排序呢？

先思考一个问题：如何将两个有序的列表合并成一个有序的列表？

开辟一个长度等同于两个数组长度之和的新数组，

使用两个指针来遍历原有的两个数组，

不断将较小的数字添加到新数组中，并移动对应的指针即可。

排序时用的都是无序数组，那么上哪里去找这两个有序的数组呢？

可以把数组不断地拆成两份，直到只剩下一个数字时，

这一个数字组成的数组我们就可以认为它是有序的。

归并排序的过程就是：

先将原数组进行二分划分，不断递归划分直至只剩一个数；

返回该数放入 左或右之一的数组，此时左右两数组内均只有一个数，故为有序。

将这两个有序数组进行有序合并，合并的新数组再返回到上一层递归。

1	3	2	4	1
1	3	2	4	1
1	3	2 right[]	4 left[]	1rigth[]
1 left[]	3rigth[]			

↓

11234				
123left[]			14rigth[]	
13 left[]		2 right[]	4 left[]	1rigth[]
1 left[]	3rigth[]			

然后：从上表 可以注意到：

- 划分后，每一次合并，若right数组中 数字小，则逆序对+1
- 对每一层递归左右合并时，左边排好序了，依旧**不影响**右边与其合并的逆序数计算。

  例：
  - 左13，右2；左31，右2；

    考虑合并右边的逆序数都是 1对。
  - 左边132，右边41；左边123，右边41；左边132，右边14；

    考虑合并右边的逆序数都是 2对。

即：在归并排序的过程中：

每次左边区间小于等于右边区间，直接插入排序结果数组，逆序对不变。

每次右边区间小于左边区间数时，直接插入排序结果数组，逆序对增加，

增加个数取决于 左边区间剩下的个数；因为归并排序将每个子区间都排成了有序的，

code：

```
class Solution{
    // 记录 逆序对 数量
    private int count ;
    public int reversePairs(int[] nums){
        // 判空：非必要，通常没有问题，但这题测试案例有 []
        if(nums.length==0 || nums==null){
            return 0;
        }
        // 初始化 count
        count=0;
        // 创建res[]数组保存排序结果，不然每步排序都需要创建临时数组
```

```
        int[] res=new int[nums.length];
        // 在归并排序的过程中计算 逆序对
        mergeSort(nums, res, 0, nums.length-1);
        return count;
    }
    // 归并排序算法
    private void mergeSort(int[] nums, int[] res, int start, int end){
        // 递归出口 -- 即只剩一个数，无法再二分时
        if(start ==end) return;
        // 二分
        int mid = (start + end)/2;
        // 区分左右区间的前后位置指针，后面合并时要用到
        // 为什么能用到，每层递归出到此层时，当前层的临时变量和递归进去时一样。
        int start1 = start;
        int end1 = mid;
        int start2 = mid+1;
        int end2= end;
        // 对左右区间递归排序
        mergeSort(nums, res, start1, end1);
        mergeSort(nums, res, start2, end2);

        //递归出来时，合并数组
        // k 用于本段区间 开始存数的位置
        // 递归是一层一层出来的，故 k取每一层起始的 start，不会破坏其他已经排好序的区间
        int k = start;
        // 合并两个有序数组，结果存入 res[]
        while(start1<=end1 && start2<=end2){
            if(nums[start1]<=nums[start2]){
                // 左边数小 时
                res[k] = nums[start1];
                start1++;
                k++;
            }else{
                // 右边数小 时，有逆序对
                res[k] = nums[start2];
                // 因为左右都是有序数组，故当前右边数 要小于左边数组剩下的数，故逆序对数如下：
                count+=(end1-start1)+1;
                start2++;
                k++;
            }
        }

        // 如果左区间中还有元素
        while (start1 <= end1){
            res[k] = nums[start1];
            k++;
            start1++;
        }
```

```
            // 求的逆序对，小的在前，故此时右边数组要是剩了 也都比左边大了。也就没有
            逆序对了
            while (start2 <= end2){
                res[k] = nums[start2];
                k++;
                start2++;
            }

            // 最后，把结果赋值回 nums 中，下一层比较需要上一层的结果
            // 依旧使用当前层的 start,end 不会影响其他区间
            for (k = start; k <= end; k++)
                nums[k] = res[k];
        }
    }
```