LeetCode 第 88 号问题：合并两个有序数组

> *本文首发于公众号「图解面试算法」，是 图解 LeetCode 系列文章之一。*
>
> *同步博客：https://www.algomooc.com*

题目来源于 LeetCode 上第 88 号问题：合并两个有序数组。题目难度为 Easy

题目描述

给你两个有序整数数组 *nums1* 和 *nums2*，请你将 *nums2* 合并到 *nums1* 中，使 *nums1* 成为一个有序数组。

说明:

- 初始化 *nums1* 和 *nums2* 的元素数量分别为 *m* 和 *n* 。
- 你可以假设 *nums1* 有足够的空间（空间大小大于或等于 *m* + *n*）来保存 *nums2* 中的元素。

示例:

```
输入：
nums1 = [1,2,3,0,0,0], m = 3
nums2 = [2,5,6],       n = 3

输出：[1,2,2,3,5,6]
```

题目解析

将这个题目放到现实中就容易多了, 不信你看看.

假如你是玩具店的老板, 其中两个货架依次摆放了形状相同大小不同的小汽车, 这些小汽车都按照从小到大摆放着, 现在你想把第二个货架的小汽车移到第一个货架上, 为了顾客看起来直观, 这些小汽车要摆放的有序, 你会怎么做呢?

很显现, 你不会把所有小汽车放到一起, 然后在一个个排序好放到第一个货架上.

你肯定会比较两个货架小汽车的大小, 把第二个货架小汽车移到第一个货架的相应位置上.

那么问题来了, 是从小的比较呢还是从大的比较呢?

先从小的比较来看, 现在第二个货架第一个汽车是最小的, 那么你得把第一个货架所有玩具往后挪一位, 然后才能放下这个汽车, **好像有点费力了**.

不想费力, 我们可以先把第一个货架的玩具移到第三个货架, 比较第二个货架和第三个货架, 把比较小的汽车放到第一个货架上. 看来还得先搬移第一个货架, **需要占用其他空间了**.

如果从后面比较呢, 也就是先比较大的汽车, 现在第二个货架最后一个是最大的汽车, 我只需要把最大的汽车拿到第一货架的最后面就可以了, 是不是很轻松, 这样依次比较, **不费力也不用费空间**就挪到了第一个货架后面了. 和第一个货架都比较完了, 发现第二个货架还剩一个最小的汽车, 这个时候你会发现第一个货架的第一个位置是空的, 我们直接拿过去就可以啦.

故事讲完啦, 通过这几种方法的尝试, 你也许已经发现了:

> *第一种方法 对应的算法是 '* ***合并后排序*** *', 时间复杂度比较大*
>
> *第二种方法 对应的算法是 '* ***双指针 + 从前向后比较*** *'*
>
> - *往后挪动汽车时间复杂度高*
> - *移到第三个货架空间复杂度高*

> *第三种方法 对应的算法是 ' **双指针 + 从后向前比较** ' , 省时又不占空间, 完美!*

下面说下 ' **双指针 + 从后向前比较** ' 的具体思路:

1. 设置双指针, 分别指向有序数组的最后一位;
2. 从后向前
    - 终止条件: 其中一个指针不在指向数组
    - 比较双指针指向的值
    - 大的或相同的值放到 *num1* 空间的尾部(尾部从后向前依次填充), 对应的指针向前挪一位
    - 循环上面步骤
3. 遍历完成后检查
    - 若指向 *num2* 的指针还有效, 说明 *num2* 中还有小于 *num1* 最小值的存在
    - 将这些值搬移到 *num1* 最前面

动画描述

 Animation

参考代码

C++ Code:

```
class Solution {
public:
    void merge(vector<int>& nums1, int m, vector<int>& nums2, int n) {
        int i=m-1, j=n-1, k=m+n-1;
        // 合并
        while(i>=0 && j>=0)
        {
            if(nums1[i] > nums2[j])
            {
                nums1[k--] = nums1[i--];
            }
            else
            {
                nums1[k--] = nums2[j--];
            }
        }
        // 合并剩余的nums2
        while(j>=0)
        {
            nums1[k--] = nums2[j--];
        }
    }
};
```

Java Code:

```java
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i=m-1, j=n-1, k=m+n-1;
        // 合并
        while(i>=0 && j>=0)
        {
            if(nums1[i] > nums2[j])
            {
                nums1[k--] = nums1[i--];
            }
            else
            {
                nums1[k--] = nums2[j--];
            }
        }
        // 合并剩余的nums2
        while(j>=0)
        {
            nums1[k--] = nums2[j--];
        }
    }
}
```

Python Code:

```python
class Solution(object):
    def merge(self, nums1, m, nums2, n):
        """
        :type nums1: List[int]
        :type m: int
        :type nums2: List[int]
        :type n: int
        :rtype: None Do not return anything, modify nums1 in-place instead.
        """
        i,j,k = m-1, n-1, m+n-1

        while i >= 0 and j >= 0:
            # print(i,j,k, nums1)
            # print(nums1[i], nums2[j])
            if nums1[i] > nums2[j]:
                nums1[k] = nums1[i]
                k-=1
                i-=1
            else:
                nums1[k] = nums2[j]
                k-=1
                j-=1
        while j >= 0:
            nums1[k] = nums2[j]
```

```
            k-=1
            j-=1
```

JavaScript Code:

```
/**
 * JavaScript 描述
 * 双指针 + 从后向前
 */
var merge = function(nums1, m, nums2, n) {
    let len = m + n;
    while(m > 0 && n > 0){
        // '>=' 相比 '>' 在某些值相同的情况下能少比较一次
        nums1[--len] = nums2[n-1] >= nums1[m-1] ? nums2[--n]: nums1[--m];
    }
    if(n > 0){
        nums1.splice(0,n,...nums2.slice(0,n));
    }
};
```

复杂度分析

- 时间复杂度: **O(m+n)**
- 空间复杂度: **O(1)**