88. Merge Sorted Arrays (/problems/merge-sorted-array/)

Feb. 7, 2019 | 25.7K views

Average Rating: 4.36 (22 votes)

Given two sorted integer arrays *nums1* and *nums2*, merge *nums2* into *nums1* as one sorted array.

Note:

- The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.
- You may assume that *nums1* has enough space (size that is greater or equal to *m* + *n*) to hold additional elements from *nums2*.

Example:

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

Output: [1,2,2,3,5,6]
```

Solution

Approach 1 : Merge and sort

Intuition

The naive approach would be to merge both lists into one and then to sort. It's a one line solution (2 lines in Java) with a pretty bad time complexity $\mathcal{O}((n+m)\log(n+m))$ because here one doesn't profit from the fact that both arrays are already sorted.

Implementation

Java | Python — Articles > 88. Merge Sorted Arrays — Copy

```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: void Do not return anything, modify nums1 in-place instead.
        """
        nums1[:] = sorted(nums1[:m] + nums2)
```

- Time complexity : $\mathcal{O}((n+m)\log(n+m))$.
- Space complexity : $\mathcal{O}(1)$.

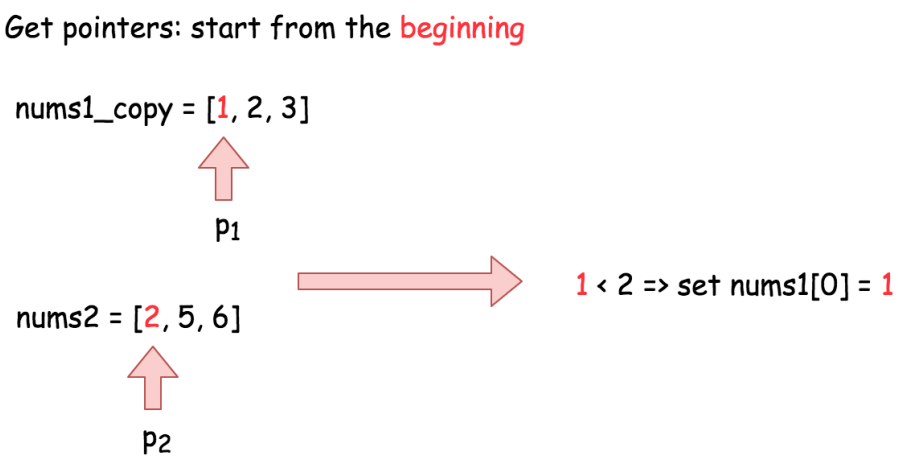
Approach 2 : Two pointers / Start from the beginning

Intuition

Typically, one could achieve $\mathcal{O}(n+m)$ time complexity in a sorted array(s) with the help of *two pointers approach*.

The straightforward implementation would be to set get pointer `p1` in the beginning of `nums1`, `p2` in the beginning of `nums2`, and push the smallest value in the output array at each step.

Since `nums1` is an array used for output, one has to keep first `m` elements of `nums1` somewhere aside, that means $\mathcal{O}(m)$ space complexity for this approach.



Implementation

Java | Python — Articles > 88. Merge Sorted Arrays — Copy

```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: void Do not return anything, modify nums1 in-place instead.
        """
        # Make a copy of nums1.
        nums1_copy = nums1[:m]
        nums1[:] = []

        # Two get pointers for nums1_copy and nums2.
        p1 = 0
        p2 = 0

        # Compare elements from nums1_copy and nums2
        # and add the smallest one into nums1.
        while p1 < m and p2 < n:
            if nums1_copy[p1] < nums2[p2]:
                nums1.append(nums1_copy[p1])
                p1 += 1
            else:
                nums1.append(nums2[p2])
                p2 += 1
```

Complexity Analysis

- Time complexity : $\mathcal{O}(n+m)$.
- Space complexity : $\mathcal{O}(m)$.

Approach 3 : Two pointers / Start from the end

Intuition

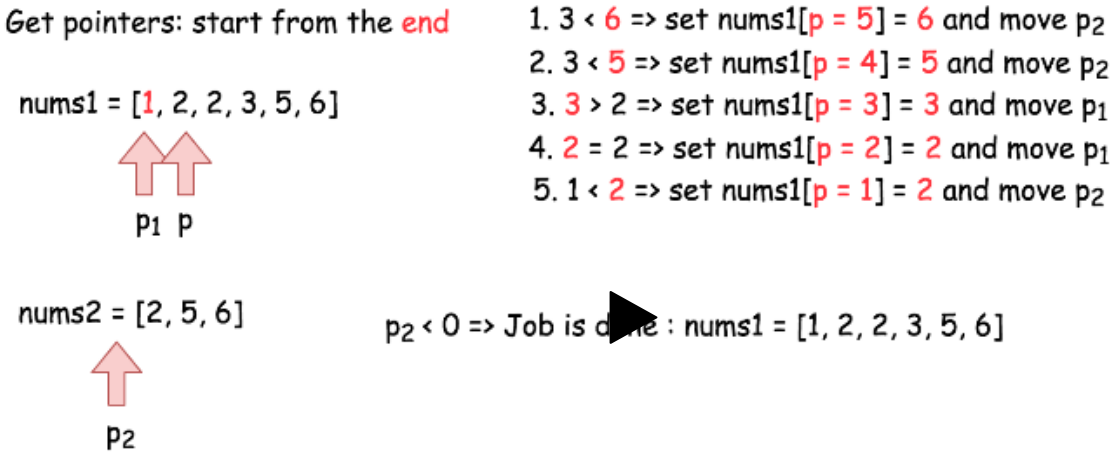
Approach 2 already demonstrates the best possible time complexity $\mathcal{O}(n+m)$ but still uses an additional space. This is because one has to keep somewhere the elements of array `nums1` while overwriting it starting from the beginning.

> What if we start to overwrite `nums1` from the end, where there is no information yet? Then no additional space is needed.

The set pointer `p` here is used to track the position of an added element.

Get pointers: start from the end

Implementation



Java | Python

Articles > 88. Merge Sorted Arrays

Copy

```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: void Do not return anything, modify nums1 in-place instead.
        """
        # two get pointers for nums1 and nums2
        p1 = m - 1
        p2 = n - 1
        # set pointer for nums1
        p = m + n - 1

        # while there are still elements to compare
        while p1 >= 0 and p2 >= 0:
            if nums1[p1] < nums2[p2]:
                nums1[p] = nums2[p2]
                p2 -= 1
            else:
                nums1[p] =  nums1[p1]
                p1 -= 1
            p -= 1

        # add missing elements from nums2
        nums1[:p2 + 1] = nums2[:p2 + 1]
```

Complexity Analysis

- Time complexity : $\mathcal{O}(n + m)$.
- Space complexity : $\mathcal{O}(1)$.

Analysis written by @liaison (https://leetcode.com/liaison/) and @andvary (https://leetcode.com/andvary/)

Rate this article:

Previous (/articles/interval-list-intersections/)

Next (/articles/satisfiability-of-equality-equations/)

Comments: 8

Sort By

Type comment here... (Markdown is supported)

Preview

Post

(/nwadhwa12345)

nwadhwa12345 (nwadhwa12345) ★22 July 8, 2019 7:49 PM


```
Simple and Easy to Understand Soln-
class Solution {
    public void merge(int[] nums1, int m, int[] nums2, int n) {
        int i=m-1;
```

Read More

5 ▲ ▼ Share Reply

(/willye)

willye (willye) ★200 August 16, 2019 9:01 AM

Very clever to have it come from the right side to avoid overwriting numbers... people can "dislike" this question all they want but it really is quite clever

3 ▲ ▼ Share Reply

(/park29)

park29 (park29) ★74 August 11, 2019 7:19 PM

comparing from the right side instead of the left side. It's really good!!!

2 ▲ ▼ Share Reply

(/xma17)

xma17 (xma17) ★8 August 30, 2019 9:32 PM

Method 2 is wrong if m +n < len(nums1)

[1,2,3,0,0,0,0]
3
[2,5,6]

Read More

1 ▲ ▼ Share Reply

SHOW 4 REPLIES

(/poream3387)

poream3387 (poream3387) ★7 April 12, 2019 9:57 PM

In approach 1, can anyone explain why I have to use `nums1[:]` instead of `nums1` for assignment?
Doing shallow copy just make a new array object? and it won't be referencing the original `nums1` ?

1 ▲ ▼ Share Reply

SHOW 1 REPLY

(/qlightman)

qlightman (qlightman) ★1 April 2, 2019 8:27 AM

very good analysis, thank you

1 ▲ ▼ Share Reply

ncakpan1 (ncakpan1) ★ -1 July 20, 2019 3:14 AM

Simple and easy :

(/ncakpan1)

```
public void merge(int[] nums1, int m, int[] nums2, int n) {
    int i = 0;
    int j =0;
```

Read More

0 Share Reply

dragonitedd (dragonitedd) ★ 27 July 8, 2019 3:01 PM

space complexity of approach 3 is O(n) if the extra n space has not been allocated to nums1 already

(/dragonitedd)

0 Share Reply