Сору

Сору

Feb. 7, 2019 | 96.5K views

nums1 = [1,2,3,0,0,0], m = 3

88. Merge Sorted Arrays 💆

Average Rating: 4.36 (59 votes)

Note: • The number of elements initialized in *nums1* and *nums2* are *m* and *n* respectively.

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

• You may assume that nums1 has enough space (size that is **equal** to m + n) to hold additional

- elements from nums2.
- **Example:** Input:

```
nums2 = [2,5,6], n = 3
 Output: [1,2,2,3,5,6]
Constraints:
```

```
• -10^9 <= nums1[i], nums2[i] <= 10^9
nums1.length == m + n
nums2.length == n
```

Approach 1: Merge and sort

Intuition

Solution

Implementation

class Solution(object):

Python

fact that both arrays are already sorted.

Java

8 9

2 def merge(self, nums1, m, nums2, n): 3 :type nums1: List[int] 4 5 :type m: int 6 :type nums2: List[int] 7 :type n: int

:rtype: void Do not return anything, modify nums1 in-place instead.

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

nums1[:] = sorted(nums1[:m] + nums2) 10

```
• 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

 $1 < 2 \Rightarrow set nums1[0] = 1$

Since nums1 is an array used for output, one has to keep first m elements of nums1 somewhere aside, that

beginning of nums2, and push the smallest value in the output array at each step.

 $nums1_copy = [1, 2, 3]$

Make a copy of nums1. $nums1_copy = nums1[:m]$

p1 += 1

Two get pointers for nums1_copy and nums2.

Compare elements from nums1_copy and nums2

nums1.append(nums1_copy[p1])

and add the smallest one into nums1.

if nums1_copy[p1] < nums2[p2]:</pre>

nums1[:] = []

p1 = 0p2 = 0

nums2 = [2, 5, 6]

Get pointers: start from the beginning

means $\mathcal{O}(m)$ space complexity for this approach.

```
Implementation
         Python
  Java
       class Solution(object):
   1
   2
           def merge(self, nums1, m, nums2, n):
   3
   4
               :type nums1: List[int]
   5
               :type m: int
   6
               :type nums2: List[int]
   7
               :type n: int
               :rtype: void Do not return anything, modify nums1 in-place instead.
   8
```

20 while p1 < m and p2 < n: 21 22

9 10

11 12

13

14 15

16 17

18

19

23

```
24
  25
                     nums1.append(nums2[p2])
  26
                     p2 += 1
  27
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.
```

 $3 < 6 \Rightarrow set nums1[p] = 6$

1. $3 < 6 \Rightarrow \text{set nums1}[p = 5] = 6$ and move p_2

Copy

Next 🕑

Sort By ▼

Implementation

Get pointers: start from the end

nums1 = [1, 2, 3, 0, 0, 0]

nums2 = [2, 5, 6]

1 2

Get pointers: start from the end

nums1 = [1, 2, 3, 0, 0, 0]

nums2 = [2, 5, 6]

```
Python
Java
    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
```

```
3
   4
   5
   6
   7
   8
  9
  10
  11
  12
  13
  14
  15
              # while there are still elements to compare
  16
  17
              while p1 >= 0 and p2 >= 0:
  18
                  if nums1[p1] < nums2[p2]:</pre>
  19
                     nums1[p] = nums2[p2]
  20
                      p2 -= 1
  21
                  else:
                      nums1[p] = nums1[p1]
  22
  23
                      p1 -= 1
                  p -= 1
  24
  25
  26
              # add missing elements from nums2
  27
              nums1[:p2 + 1] = nums2[:p2 + 1]
Complexity Analysis
   • Time complexity : \mathcal{O}(n+m).
   • Space complexity : \mathcal{O}(1).
Rate this article: * * * * *
 O Previous
Comments: 37
              Type comment here... (Markdown is supported)
```

```
Preview
                                                                                           Post
willye * 861 • August 16, 2019 9:31 PM
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
88 \Lambda 🗸 🗁 Share 🦘 Reply
SHOW 4 REPLIES
nwadhwa12345 * 34 • July 9, 2019 8:19 AM
                                                                                       A Report
 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
13 A V 🗗 Share 👆 Reply
SHOW 1 REPLY
addy_boy ★43 ② December 21, 2019 3:18 PM
                                                                                       A Report
Approach 3 is the best but the given Java implementation for that seems somewhat too terse,
confusing, and cavalier. I hope the following is a more easy to follow implementation which also beat
100% of the Java submissions in both runtime and memory usage:
 class Solution
                                           Read More
9 A V C Share    Reply
crisa * 109 • May 7, 2020 9:53 AM
The problem description says You may assume that nums1 has enough space (size that is
greater or equal to m + n) to hold additional elements from nums2.
Then why is one of the tests cases
[0] 0 [1] 1 ? Nums1 is initialized to an empty array, which means it can't hold anything
8 A V C Share  Reply
SHOW 1 REPLY
comparing from the right side instead of the left side. It's really good!!!
7 \Lambda 🗸 🗁 Share 👆 Reply
xma17 ★ 15 ② August 31, 2019 10:02 AM
                                                                                       A Report
Method 2 is wrong if m + n < len(nums1)
[1,2,3,0,0,0,0]
[2,5,6]
                                           Read More
3 A V C Share  Reply
SHOW 4 REPLIES
poream3387 🖈 23 🗿 April 13, 2019 10:27 AM
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?
3 A V C Share   Reply
SHOW 2 REPLIES
ab3rd ★ 5 ② February 9, 2020 6:40 AM
                                                                                       A Report
Honestly I don't understand why this question receives so many dislikes. I think it is interesting, clever,
and elegant.
2 A V C Share  Reply
thepatriot 🖈 260 🗿 January 3, 2020 2:20 AM
Line 16 from solution #3 is not intuitive and I don't see any mention of it in the article.
I would appreciate someone explaining that to me.
1 A V C Share   Reply
```

SHOW 1 REPLY

SHOW 1 REPLY

(1234)

Can anyone tell me

1 A V C Share Reply

SJSU153 * 15 • November 25, 2019 8:25 AM

1. Why does solution 3 assumes that only nums2 will have missing elements after 1st loop?

2. How does comparing from the back avoids overwriting elements?