189. Rotate Array

July 29, 2016 | 334K views

Average Rating: 4.73 (211 votes)

Given an array, rotate the array to the right by *k* steps, where *k* is non-negative.

Follow up:

- Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.
- Could you do it in-place with O(1) extra space?

Example 1:

```
Input: nums = [1,2,3,4,5,6,7], k = 3
Output: [5,6,7,1,2,3,4]
Explanation:
rotate 1 steps to the right: [7,1,2,3,4,5,6]
rotate 2 steps to the right: [6,7,1,2,3,4,5]
rotate 3 steps to the right: [5,6,7,1,2,3,4]
```

Example 2:

```
Input: nums = [-1,-100,3,99], k = 2
Output: [3,99,-1,-100]
Explanation:
rotate 1 steps to the right: [99,-1,-100,3]
rotate 2 steps to the right: [3,99,-1,-100]
```

Constraints:

- `1 <= nums.length <= 2 * 10^4`
- It's guaranteed that `nums[i]` fits in a 32 bit-signed integer.
- `k >= 0`

Summary

We have to rotate the elements of the given array k times to the right.

Solution

Approach 1: Brute Force

The simplest approach is to rotate all the elements of the array in k steps by rotating the elements by 1 unit in each step.

Java Python3 Copy

```
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        # speed up the rotation
        k %= len(nums)

        for i in range(k):
            previous = nums[-1]
            for j in range(len(nums)):
                nums[j], previous = previous, nums[j]
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n \times k)$. All the numbers are shifted by one step($\mathcal{O}(n)$) k times($\mathcal{O}(n)$).
- Space complexity: $\mathcal{O}(1)$. No extra space is used.

Approach 2: Using Extra Array

Algorithm

We use an extra array in which we place every element of the array at its correct position i.e. the number at index i in the original array is placed at the index $(i+k)\%$ length of array. Then, we copy the new array to the original one.

Java Python3 Copy

```
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        n = len(nums)
        a = [0] * n
        for i in range(n):
            a[(i + k) % n] = nums[i]

        nums[:] = a
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n)$. One pass is used to put the numbers in the new array. And another pass to copy the new array to the original one.
- Space complexity: $\mathcal{O}(n)$. Another array of the same size is used.

Approach 3: Using Cyclic Replacements

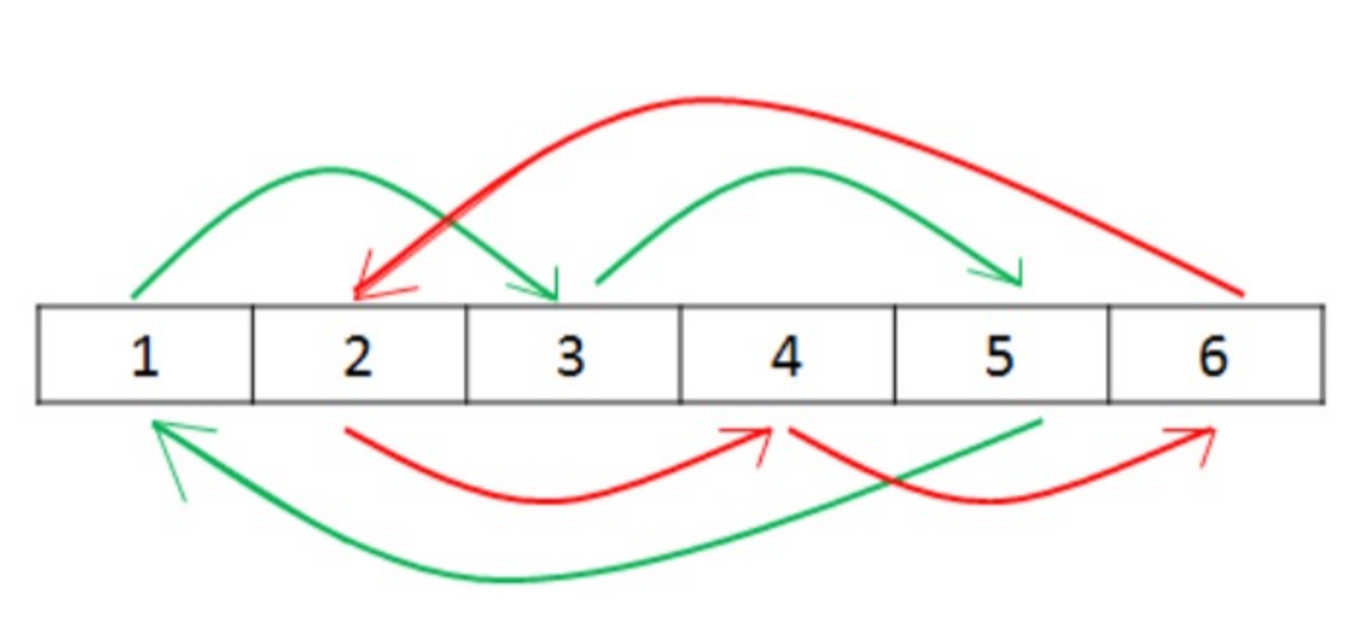
Algorithm

We can directly place every number of the array at its required correct position. But if we do that, we will destroy the original element. Thus, we need to store the number being replaced in a $temp$ variable. Then, we can place the replaced number(temp) at its correct position and so on, n times, where n is the length of array. We have chosen n to be the number of replacements since we have to shift all the elements of the array(which is n). But, there could be a problem with this method, if $n\%k=0$ where $k=k\%n$ (since a value of k larger than n eventually leads to a k equivalent to $k\%n$). In this case, while picking up numbers to be placed at the correct position, we will eventually reach the number from which we originally started. Thus, in such a case, when we hit the original number's index again, we start the same process with the number following it.

Now let's look at the proof of how the above method works. Suppose, we have n as the number of elements in the array and k is the number of shifts required. Further, assume $n\%k=0$. Now, when we start placing the elements at their correct position, in the first cycle all the numbers with their index i satisfying $i\%k=0$ get placed at their required position. This happens because when we jump k steps every time, we will only hit the numbers k steps apart. We start with index $i=0$, having $i\%k=0$. Thus, we hit all the numbers satisfying the above condition in the first cycle. When we reach back the original index, we have placed $\frac{n}{k}$ elements at their correct position, since we hit only that many elements in the first cycle. Now, we increment the index for replacing the numbers. This time, we place other $\frac{n}{k}$ elements at their correct position, different from the ones placed correctly in the first cycle, because this time we hit all the numbers satisfy the condition $i\%k=1$. When we hit the starting number again, we increment the index and repeat the same process from $i=1$ for all the indices satisfying $i\%k==1$. This happens till we reach the number with the index $i\%k=0$ again, which occurs for $i=k$. We will reach such a number after a total of k cycles. Now, the total count of numbers exclusive numbers placed at their correct position will be $k \times \frac{n}{k}=n$. Thus, all the numbers will be placed at their correct position.

Look at the following example to clarify the process:

```
nums: [1, 2, 3, 4, 5, 6]
k: 2
```



k = 2

Java Python3 Copy

```
class Solution:
    def rotate(self, nums: List[int], k: int) -> None:
        n = len(nums)
        k %= n

        start = count = 0
        while count < n:
            current, prev = start, nums[start]
            while True:
                next_idx = (current + k) % n
                nums[next_idx], prev = prev, nums[next_idx]
                current = next_idx
                count += 1

                if start == current:
                    break
            start += 1
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n)$. Only one pass is used.
- Space complexity: $\mathcal{O}(1)$. Constant extra space is used.

Approach 4: Using Reverse

Algorithm

This approach is based on the fact that when we rotate the array k times, k elements from the back end of the array come to the front and the rest of the elements from the front shift backwards.

In this approach, we firstly reverse all the elements of the array. Then, reversing the first k elements followed by reversing the rest $n-k$ elements gives us the required result.

Let $n=7$ and $k=3$.

```
Original List                   : 1 2 3 4 5 6 7
After reversing all numbers     : 7 6 5 4 3 2 1
After reversing first k numbers : 5 6 7 4 3 2 1
After revering last n-k numbers : 5 6 7 1 2 3 4 --> Result
```

Java Python3 Copy

```
class Solution:
    def reverse(self, nums: list, start: int, end: int) -> None:
        while start < end:
            nums[start], nums[end] = nums[end], nums[start]
            start, end = start + 1, end - 1

    def rotate(self, nums: List[int], k: int) -> None:
        n = len(nums)
        k %= n

        self.reverse(nums, 0, n - 1)
        self.reverse(nums, 0, k - 1)
        self.reverse(nums, k, n - 1)
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n)$. n elements are reversed a total of three times.
- Space complexity: $\mathcal{O}(1)$. No extra space is used.

Rate this article: ★★★★★

Comments: 148 Sort By

Type comment here... (Markdown is supported)

Preview Post

rajatmittal18 ★ 276 February 3, 2019 11:05 PM
I don't know why so many dislikes to this question. This is a great question I believe and should be put under medium category with the restriction of solving in-place and in O(n).
221 Share Reply
SHOW 4 REPLIES

meghaharlalka ★ 133 February 11, 2019 5:22 AM
this should be under medium category
128 Share Reply
SHOW 2 REPLIES

mglezer ★ 146 August 13, 2018 4:18 AM
@vinod23 really nice writeup! As others have mentioned the proof for Approach #3 is wrong. What you're actually calculating is, mathematically speaking, the order (e.g., size) of a subgroup of a cyclic group--in this case, the order of the subgroup generated by `k` of the cyclic group `n`. Point is, the order of this subgroup is actually `lcm(n,k)/k` rather than `n/k`, where `lcm := least common multiple`. So, for example, when `n = 9` and `k = 4`, `lcm(9,4)/4 = 36/4 = 9` (in this case, `4` is
Read More
34 Share Reply
SHOW 3 REPLIES

Cheney1993 ★ 50 March 17, 2019 3:11 AM
Python3, slice solution:

```
def rotate(self, nums: List[int], k: int) -> None:
    k %= len(nums)
    nums[k:], nums[:k] = nums[:-k], nums[-k:]
```

Read More
35 Share Reply
SHOW 4 REPLIES

terrible_whiteboard ★ 628 May 19, 2020 6:11 PM
I made a video if anyone is having trouble understanding the reverse method (clickable link)
https://youtu.be/T9zQP1IkMDE
Read More
20 Share Reply

Kaa1el ★ 44 December 4, 2018 10:31 PM
Proper explanation for solution 3 using group theory:

Let `n = len(nums)`. The group action `x -> (x+j*k) mod n` (for `j = 0,1,...,n-1`, i.e., the cyclic group of order `n`) acts transitively on `nums`. By orbit-stabilizer theorem, for each element `x`, the size of the orbit `Orb(x)` times the size of the stabilizer `Stab(x)` is exactly the size of the group `n`. In this
Read More
14 Share Reply
SHOW 1 REPLY

korte ★ 13 September 17, 2018 8:12 AM
I can't understand the approach#3 vrey much.
13 Share Reply
SHOW 4 REPLIES

spicam ★ 16 July 18, 2018 11:32 AM
The Proof of Approach 3 is wrong. But the method works. Here is the correct proof (intuitively). For each cycle it is not necessarily n/k elements moved cuz the original position may not be hit until you finish the entire array. However, if at some point we hit the original position with some elements not being moved, then the next position is guaranteed not moved (Assume if the next element is moved, it's not difficult to prove that all the element would be moved). That's why you MUST have a counter
Read More
11 Share Reply
SHOW 3 REPLIES

logical_paradox ★ 255 February 12, 2019 5:10 PM
What do you mean by "Futher, assume n". Also, what is "i satisfying i"? What does that even mean??
10 Share Reply
SHOW 2 REPLIES

foxyz ★ 330 June 8, 2019 12:14 AM
I thought this one was hard....
9 Share Reply
SHOW 1 REPLY