LEETCODE 189.
ROTATE ARRAY

Difficulty: Medium

Dao Trung Kien - 6128302
Sochivoath Chiv - 6128304

TABLE OF CONTENTS

01

PROBLEM DESCRIPTION

A brief introduction of the problem

THE QUESTIONS

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

Example 1:

Input: nums = [1, 2, 3, 4, 5, 6, 7], k =3

Output: [5, 6, 7, 1, 2, 3, 4]

Explanation:

Rotate 1 step(s) to the right : [7, 1, 2, 3, 4, 5, 6]
Rotate 2 step(s) to the right : [6, 7, 1, 2, 3, 4, 5]
Rotate 3 step(s) 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 step(s) to the right : [99, -1, -100, 3]
Rotate 2 step(s) to the right : [3, 99, -1, -100]

Constraints:

- $1 \leq nums.length \leq 2*10^4$
- $-2^{31} \leq nums[i] \leq 2^{31} - 1$
- $0 \leq k \leq 10^5$

02

PROBLEM ANALYSIS

Analyze the problem to get more information

THE ANALYSIS

- The problem is very short and straightforward with the only gimmick being that the original array has to be rotated
- The output is ***NOT*** a new array
- The simplicity of the problem signifies that there are multiple different solutions. Instead of just solving, we will explore different techniques in order to find the most optimal solution.

03

SOLUTIONS

5 possible ways to solve the problem using python

BRUTE FORCE

Brute force algorithms can act as a benchmark or starting point to study issues and look for areas of improvement

THE CODE

```python
def rotate(nums, k):
    k %= len(nums)

    for i in range(k):
        prev = nums[-1] # prev is the last element

        for j in range(len(nums)):
            nums[j], prev = prev, nums[j]
```

THE RESULTS

Testing with simple sample data within the console shows the following results:

```
(base) D:\Users\Sochi\Documents\Algorithm_Design_CS3201\Asm5>python test.py
Nums = [1, 100, 99, 50], k = 2
[99, 50, 1, 100]

(base) D:\Users\Sochi\Documents\Algorithm_Design_CS3201\Asm5>python test.py
Nums = [0, 1, 2, 3, 4, 5, 6, 7], k = 13
[3, 4, 5, 6, 7, 0, 1, 2]
```

From these results, we can conclude that the brute force technique works as intended, however it does not pass the time limit set by the question

03/04/2021 01:05	Time Limit Exceeded	N/A	N/A	python

COMPLEXITY ANALYSIS

- After analyzing the code, we can see that the inner loop goes through each digit of the array (n), and the outer loop repeats this k times. The time complexity is then concluded to be (O(n) x k) = ***O(n x k)***
- The space complexity is only ***O(1)*** as no extra space is used.

EXTRA ARRAY

Attempts to reduce number of loops. Uses extra array to be filled with the rotated values. Array is copied back into original array.

THE CODE

```
def rotate(self, nums, k):
        n = len(nums)
        a = [0] * n

        for i in range(n):
            a[(i+k)%n] = nums[i]

        nums[:] = a
```

THE RESULTS

03/05/2021 14:33	Accepted	48 ms	14.5 MB	python

The submission result shows that, unlike the previous algorithm, the use of the extra array and the optimization to the use of only one loop has allowed the program to be accepted without exceeding the time limit.

Runtime: 44 ms, faster than 84.39% of Python online submissions for Rotate Array.

Memory Usage: 14.6 MB, less than 5.32% of Python online submissions for Rotate Array.

COMPLEXITY ANALYSIS

- As we are using only one loop (that loops through all elements in the nums array), we can conclude that the time complexity is ***O(n)***.
- In exchange for a faster runtime, we sacrifice memory space by creating a completely new array with the same length of ***O(n)***.

CYCLIC REPLACEMENTS

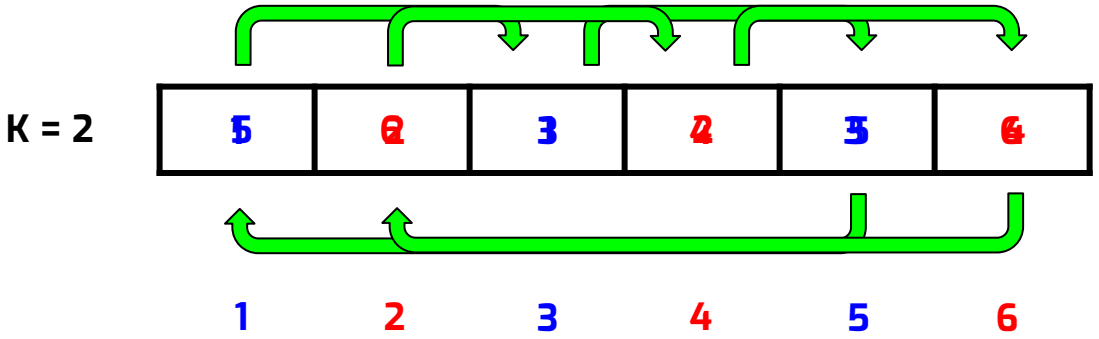
Attempts to reduce the extra space from ***O(n)*** to ***O(1)*** by utilizing only one temp variable to store the number being replaced rather than saving an entire array

THE CODE

```python
def rotate(nums, k):
    n = len(nums)
    k %= n

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

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



THE RESULTS

The results show that the runtime of the algorithm remained exactly the same as with the extra array method, however the memory usage of is much lower as we store only 1 extra variable at all times.

Runtime: 44 ms, faster than 84.39% of Python online submissions for Rotate Array.

Memory Usage: 13.9 MB, less than 70.59% of Python online submissions for Rotate Array.

COMPLEXITY ANALYSIS

- Despite the code containing nested loops, the running time of the algorithm remains ***O(n)***.
- Each element is passed over only ***once***.
- Only 1 extra variable regardless of the size of the array,
- The space complexity is reduced to ***O(1)***.

USING REVERSALS

Another approach utilizes the reversal of arrays in order to place them correctly

THE CODE

```python
def reverse(nums, start, end):
    while start<end:
        nums[start], nums[end] = nums[end], nums[start]
        start, end = start + 1 , end - 1
```

```python
def rotate (nums, k):
    n = len(nums)
    k %= n

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

THE RESULTS

Runtime: 40 ms, faster than 96.23% of Python online submissions for Rotate Array.

Memory Usage: 13.9 MB, less than 70.59% of Python online submissions for Rotate Array.

The submission shows that the algorithm is slightly faster than the previous techniques, however, this 4ms is admissible. The memory usage was also the same as the cyclic replacement algorithm.

COMPLEXITY ANALYSIS

- The time complexity remains ***O(n)*** as n elements are reversed a total of 3 times.
- The space complexity also remains ***O(1)*** as no extra space is ever used. The function calls may add some memory overhead.

Slicing

Utilizes the slices, where we slice the array into 2 segments and swap the position of these segments to produce a rotated array

THE CODE

```python
def rotate(nums, k):
    n = len(nums)
    k %= n

    a = nums[n-k:] + nums[:n-k]
    nums[:] = a
```

THE RESULTS

From the figure above, we can see that this approach shares the same memory usage as the previous 2 algorithms, however, this approach is significantly faster than the previous approaches

COMPLEXITY ANALYSIS

- As the approach utilizes slicing by direct reference to index rather than looping through the arrays, regardless of the size of the array, the time complexity is ***O(1)*** constant time.

04

CONCLUSION

Last words about the problem
and attempt

LAST WORDS

EXPERIENCE

We explore multiple approaches to the same problem: rotating an array in place. Despite the question being relatively simple and straightforward, it has allowed us to fully understand and refine algorithmic techniques.

EFFICIENCY

From the most intuitive method (brute force), we have incrementally improved each solution. Finally, we have successfully reduced the both time complexity and space complexity from O(n) to constant time O(1).

THANKS!

Do you have any questions?

Dao Trung Kien - 6128302
Sochivoath Chiv - 6128304

References:

Extra Array, Cyclic Replacements, Using Reversals:
https://leetcode.com/problems/rotate-array/solution/