<https://leetcode.com/problems/rotate-array/description/>

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

**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. 1 <= nums.length <= 10^5
2. -2^31 <= nums[i] <= 2^31 - 1
3. 0 <= k <= 10^5

**Follow up:**

Try to come up with as many solutions as you can. There are at least three different ways to solve this problem.

Could you do it in-place with O(1) extra space?

**Attempt 1: 2024-05-08**

**Solution 1: Reverse -> Reverse -> Reverse (30 min)**

class Solution {

    public void rotate(int[] nums, int k) {

        k %= nums.length;

        reverse(nums, 0, nums.length - 1);

        reverse(nums, 0, k - 1);

        reverse(nums, k, nums.length - 1);

    }

    public void reverse(int[] nums, int start, int end) {

        while (start < end) {

            int temp = nums[start];

            nums[start] = nums[end];

            nums[end] = temp;

            start++;

            end--;

        }

    }

}

Time Complexity: O(N)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/rotate-array/solutions/54250/easy-to-read-java-solution/>

nums = "----->-->"; k =3

result = "-->----->";

reverse "----->-->" we can get "<--<-----"

reverse "<--" we can get "--><-----"

reverse "<-----" we can get "-->----->"

this visualization help me figure it out :)

public void rotate(int[] nums, int k) {

k %= nums.length;

reverse(nums, 0, nums.length - 1);

reverse(nums, 0, k - 1);

reverse(nums, k, nums.length - 1);

}

public void reverse(int[] nums, int start, int end) {

while (start < end) {

int temp = nums[start];

nums[start] = nums[end];

nums[end] = temp;

start++;

end--;

}

}

Time and Space Complexity

The time complexity of the code is O(n) where n is the length of the array,

because it involves slicing the array into two parts and then concatenating them,

both of which take O(n) operations.

The space complexity is O(1) because the rotation operation modifies the array in-place.

Although slicing the array appears to create new arrays, this is handled under the hood

by Python and does not require additional space proportional to the size of the input array.

**Refer to**

[L61.P7.5.Rotate List](note://3DFFBC943C4B4314949C42B00615E488)

public class Solution {

public void rotate(int[] nums, int k) {

int length = nums.length;

// Note 1: Create a new array which concatenate original array at head

int[] concatenate = new int[length \* 2];

for(int i = 0; i < length; i++) {

concatenate[i] = nums[i];

concatenate[length + i] = nums[i];

}

// Note 2: Most important case is relation between length and k, if given

// k larger than length, e.g length = 1, but need to rotate k = 2 steps,

// we need to get remainder with k % length, then create item index of

// concatenate aray based on length and remainder

for(int i = 0; i < length; i++) {

nums[i] = concatenate[length - (k % length) + i];

}

}

}