Rotate Array

Rotate an array of n elements to the right by k steps.

For example, with $n = 7$ and $k = 3$, the array `[1,2,3,4,5,6,7]` is rotated to `[5,6,7,1,2,3,4]`.

Note:
Try to come up as many solutions as you can, there are at least 3 different ways to solve this problem.

[show hint]

Hint:
Could you do it in-place with O(1) extra space?
Related problem: Reverse Words in a String II

Credits:
Special thanks to @Freezen for adding this problem and creating all test cases.

Solution 1

1. Make an extra copy and then rotate.

Time complexity: O(n). Space complexity: O(n).

```
class Solution
{
public:
    void rotate(int nums[], int n, int k)
    {
        if ((n == 0) || (k <= 0))
        {
            return;
        }

        // Make a copy of nums
        vector<int> numsCopy(n);
        for (int i = 0; i < n; i++)
        {
            numsCopy[i] = nums[i];
        }

        // Rotate the elements.
        for (int i = 0; i < n; i++)
        {
            nums[(i + k)%n] = numsCopy[i];
        }
    }
};
```

2. Start from one element and keep rotating until we have rotated n different elements.

Time complexity: O(n). Space complexity: O(1).

```
class Solution
{
public:
    void rotate(int nums[], int n, int k)
    {
        if ((n == 0) || (k <= 0))
        {
            return;
        }

        int cntRotated = 0;
        int start = 0;
        int curr = 0;
        int numToBeRotated = nums[0];
        int tmp = 0;
        // Keep rotating the elements until we have rotated n
        // different elements.
        while (cntRotated < n)
        {
            do
            {
                tmp = nums[(curr + k)%n];
                nums[(curr+k)%n] = numToBeRotated;
                numToBeRotated = tmp;
                curr = (curr + k)%n;
                cntRotated++;
            } while (curr != start);
            // Stop rotating the elements when we finish one cycle,
            // i.e., we return to start.

            // Move to next element to start a new cycle.
            start++;
            curr = start;
            numToBeRotated = nums[curr];
        }
    }
};
```

3. Reverse the first n - k elements, the last k elements, and then all the n elements.

Time complexity: O(n). Space complexity: O(1).

```
    class Solution
    {
    public:
        void rotate(int nums[], int n, int k)
        {
            k = k%n;

            // Reverse the first n - k numbers.
            // Index i (0 <= i < n - k) becomes n - k - i.
            reverse(nums, nums + n - k);

            // Reverse tha last k numbers.
            // Index n - k + i (0 <= i < k) becomes n - i.
            reverse(nums + n - k, nums + n);

            // Reverse all the numbers.
            // Index i (0 <= i < n - k) becomes n - (n - k - i) = i + k.
            // Index n - k + i (0 <= i < k) becomes n - (n - i) = i.
            reverse(nums, nums + n);
        }
    };
```

4. Swap the last k elements with the first k elements.

Time complexity: O(n). Space complexity: O(1).

```
class Solution
{
public:
    void rotate(int nums[], int n, int k)
    {
        for (; k = k%n; n -= k, nums += k)
        {
            // Swap the last k elements with the first k elements.
            // The last k elements will be in the correct positions
            // but we need to rotate the remaining (n - k) elements
            // to the right by k steps.
            for (int i = 0; i < k; i++)
            {
                swap(nums[i], nums[n - k + i]);
            }
        }
    }
};
```

5. Keep swapping two subarrays.

Time complexity: O(n). Space complexity: O(1).

```
class Solution
{
public:
    void rotate(int nums[], int n, int k)
```

```
    {
        if ((n == 0) || (k <= 0) || (k%n == 0))
        {
            return;
        }

        k = k%n;
        // Rotation to the right by k steps is equivalent to swapping
        // the two subarrays nums[0,...,n - k - 1] and nums[n - k,...,n - 1].
        int start = 0;
        int tmp = 0;
        while (k > 0)
        {
            if (n - k >= k)
            {
                // The left subarray with size n - k is longer than
                // the right subarray with size k. Exchange
                // nums[n - 2*k,...,n - k - 1] with nums[n - k,...,n - 1].
                for (int i = 0; i < k; i++)
                {
                    tmp = nums[start + n - 2*k + i];
                    nums[start + n - 2*k + i] = nums[start + n - k + i];
                    nums[start + n - k + i] = tmp;
                }

                // nums[n - 2*k,...,n - k - 1] are in their correct positions now
.
                // Need to rotate the elements of nums[0,...,n - k - 1] to the ri
ght
                // by k%n steps.
                n = n - k;
                k = k%n;
            }
            else
            {
                // The left subarray with size n - k is shorter than
                // the right subarray with size k. Exchange
                // nums[0,...,n - k - 1] with nums[n - k,...,2*(n - k) - 1].
                for (int i = 0; i < n - k; i++)
                {
                    tmp = nums[start + i];
                    nums[start + i] = nums[start + n - k + i];
                    nums[start + n - k + i] = tmp;
                }

                // nums[n - k,...,2*(n - k) - 1] are in their correct positions n
ow.
                // Need to rotate the elements of nums[n - k,...,n - 1] to the ri
ght
                // by k - (n - k) steps.
                tmp = n - k;
                n = k;
                k -= tmp;
                start += tmp;
            }
        }
    }
```

```
};
```

written by zhukov original link here

Solution 2

```
void rotate(int nums[], int n, int k) {
    reverse(nums,nums+n);
    reverse(nums,nums+k%n);
    reverse(nums+k%n,nums+n);
}
```

written by monaziyi original link here

Solution 3

I really don't like those *something little* line solutions as they are incredibly hard to read. Below is my 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--;
    }
}
```

written by danny6514 original link here