Daily Coding Problem #126

Problem

This problem was asked by Facebook.

Write a function that rotates a list by k elements. For example, `[1, 2, 3, 4, 5, 6]` rotated by two becomes `[3, 4, 5, 6, 1, 2]`. Try solving this without creating a copy of the list. How many swap or move operations do you need?

Solution

We can naively rotate a list without creating a copy by simply moving each element down by one, k times. Don't forget to wrap around:

```
def rotate(lst, k):
    for _ in range(k):
        # Move each element down by one.
        first_element = lst[0]
        for i in range(len(lst) - 1):
            lst[i] = lst[i + 1]
        lst[len(lst) - 1] = first_element
    return lst
```

Although this takes constant space, this will take O(nk) time. Can we do this any faster?

We can view this problem as transforming the list into `lst[k:] + lst[:k]`. By reversing these subarrays and then reversing the whole array we can effectively rotate the array in linear time and without copying.

Take our example, `[1, 2, 3, 4, 5, 6]` and k = 2.

- First reverse from 0 to k: `[2, 1, 3, 4, 5, 6]`
- Then reverse from k to n: `[2, 1, 6, 5, 4, 3]`
- Then reverse from 0 to n: `[3, 4, 5, 6, 1, 2]`

```python
def rotate(lst, k):
    reverse(lst, 0, k - 1)
    reverse(lst, k, len(lst) - 1)
    reverse(lst, 0, len(lst) - 1)

def reverse(lst, i, j):
    while i < j:
        lst[i], lst[j] = lst[j], lst[i]
        i += 1
        j -= 1
```

Since reversing a list takes O(n) and we do a constant number of reversals (3), this algorithm takes O(n) time.
