# Daily Coding Problem #190

## Problem

This problem was asked by Facebook.

Given a circular array, compute its maximum subarray sum in O(n) time. A subarray can be empty, and in this case the sum is 0.

For example, given `[8, -1, 3, 4]`, return 15 as we choose the numbers 3, 4, and 8 where the 8 is obtained from wrapping around.

Given `[-4, 5, 1, 0]`, return 6 as we choose the numbers 5 and 1.

## Solution

This question is very similar to Daily Coding Problem #49, which asked to find the maximum subarray sum of an array, although in this case it's possible that the subarray can wrap around.

So we can split this problem into two parts. One part is the same as before: find the maximum subarray sum that doesn't wrap around. We also can find the maximum subarray sum that *does* wrap around, and take the maximum of the two.

To find the maximum subarray sum that wraps around, we can find the minimum subarray sum that doesn't wrap around by similar logic as #49, and subtract the total sum of the array by it.

```
def maximum_circular_subarray(arr):
    max_subarray_sum_wraparound = sum(arr) - min_subarray_sum(arr)
    return max(max_subarray_sum(arr), max_subarray_sum_wraparound)


def max_subarray_sum(arr):
     max_ending_here = max_so_far = 0
     for x in arr:
         max_ending_here = max(x, max_ending_here + x)
         max_so_far = max(max_so_far, max_ending_here)
     return max_so_far
```

```
def min_subarray_sum(arr):
    min_ending_here = min_so_far = 0
    for x in arr:
        min_ending_here = min(x, min_ending_here + x)
        min_so_far = min(min_so_far, min_ending_here)
    return min_so_far
```

This takes O(n) time and O(1) space.