918. Maximum Sum Circular Subarray

Problem Statement

The task is to find the maximum sum of a non-empty subarray within a given circular integer array, nums. A circular array means that the array is conceptually connected end-to-end, so the elements wrap around.

Intuition

To solve the maximum subarray problem for a circular array, we must consider two cases:

1. The maximum sum subarray is similar to the one found in a non-circular array (Kadane's algorithm is useful here).
2. The maximum sum subarray is the result of wrapping around the circular array.

For the first case, we use Kadane's algorithm to find the maximum sum subarray that does not wrap around.

For the second case, to handle subarrays that wrap, consider that the answer could be the total sum of the array minus the minimum sum subarray. This is akin to "selecting" the rest of the array that doesn't include the minimum sum subarray. To find the minimum sum subarray, we modify Kadane's algorithm. At the end, the maximum possible sum for the case when the subarray wraps around is computed as the total sum of the array minus the minimum sum subarray (sum(nums) - s2).

The edge case to consider is when all numbers in the array are negative. In this situation, Kadane's algorithm yields a subarray sum which is the maximum negative number (i.e., the least negative), and thus is the maximum sum we can achieve without wrapping around. Subtracting any subarray would result in a smaller sum, so we return the maximum subarray sum found without wrapping in this case.

Putting it all together, the final answer is the larger of the two possibilities: the maximum subarray sum found without wrapping (using Kadane's algorithm) or the total array sum minus the minimum subarray sum, unless all numbers are negative, in which case we return the maximum subarray sum without wrapping.

Implementation

```python
from typing import List

class Solution:
    def max_subarray_sum_circular(self, nums: List[int]) -> int:
        max_sum_end_here = min_sum_end_here = max_subarray_sum = min_subarray_sum = nums[0]

        # Iterate through the given nums list starting from the second element
        for num in nums[1:]:
            # Update max_sum_end_here to be the maximum of the current number or the current number
            max_sum_end_here = num + max(max_sum_end_here, 0)
            # Update min_sum_end_here to be the minimum of the current number or the current number
            min_sum_end_here = num + min(min_sum_end_here, 0)

            # Update the max_subarray_sum if the newly computed max_sum_end_here is larger
            max_subarray_sum = max(max_subarray_sum, max_sum_end_here)
            # Update the min_subarray_sum if the newly computed min_sum_end_here is smaller
            min_subarray_sum = min(min_subarray_sum, min_sum_end_here)

        # If the max_subarray_sum is non-positive, the whole array could be non-positive
        # Thus, the max subarray sum is the max_subarray_sum itself
        if max_subarray_sum <= 0:
            return max_subarray_sum
        else:
            # Otherwise, we compare the max_subarray_sum vs. total_sum minus min_subarray_sum
            # The latter represents the maximum sum obtained by considering the circular nature of
            # We subtract min_subarray_sum from the total sum to get the maximum sum subarray which
            total_sum = sum(nums)
            return max(max_subarray_sum, total_sum - min_subarray_sum)

# Usage example:
# solution = Solution()
# print(solution.max_subarray_sum_circular([5, -3, 5])) # Output: 10
```