Maximum Sum Circular Subarray

This is the classic maximum subarray sum on a circular array. The key is to combine two cases.

Core Idea (Kadane twice):

For any array, the maximum circular subarray sum is either:

1. Non-wrapping: entirely inside the array (no wrap) → just the standard Kadane maximum subarray.
2. Wrapping: it wraps around the end to the start. If a subarray wraps, it's equivalent to total sum − (minimum subarray sum), because the wrap includes "everything except" some middle chunk; that excluded chunk must be the minimum subarray.

So: answer = max(max_subarray_sum, total_sum − min_subarray_sum).

Important edge case:

If all numbers are negative, then:

- max_subarray_sum is the largest (least negative) element,
- min_subarray_sum equals total_sum,
- total_sum − min_subarray_sum becomes 0, which is invalid because the subarray must be non-empty.

So in that case, return max_subarray_sum only.

How to compute the pieces:

- Kadane's max sum in $O(n)$:
  - Track curMax = max(num, curMax + num) and bestMax = max(bestMax, curMax).
- Kadane's min sum (same trick but flipped):
  - Track curMin = min(num, curMin + num) and bestMin = min(bestMin, curMin).
- total_sum: accumulate in the same pass.

Do it all in one pass:

- initialize curMax = curMin = 0, bestMax = $-\infty$, bestMin = $+\infty$,

total = 0.

- For each x:
  - curMax = max(x, curMax + x), bestMax = max(bestMax, curMax).
  - curMin = min(x, curMin + x), bestMin = min(bestMin, curMin).
  - total += x
- If bestMax < 0 (all negative), return bestMax. Else return max(bestMax, total - bestMin).

Why this works (intuition):

- A wrapping subarray looks like: suffix + prefix. Its sum equals the total array minus a contiguous middle block that's not taken; to maximize the wrap, minimize the middle block → total - minSubarray