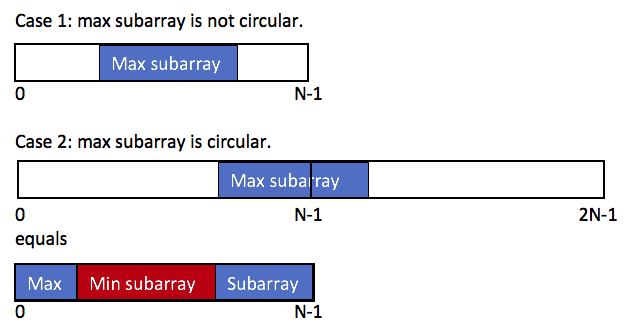
# [**https://leetcode.com/problems/maximum-sum-circular-subarray/discuss/178422/One-Pass**](https://leetcode.com/problems/maximum-sum-circular-subarray/discuss/178422/One-Pass)

# **Intuition**

I guess you know how to solve max subarray sum (without circular).  
If not, you can have a reference here: 53. Maximum Subarray

# **Explanation**

So there are two case.  
Case 1. The first is that the subarray take only a middle part, and we know how to find the max subarray sum.  
Case2. The second is that the subarray take a part of head array and a part of tail array.  
We can transfer this case to the first one.  
The maximum result equals to the total sum minus the minimum subarray sum.

Here is a diagram by @motorix:  


So the max subarray circular sum equals to  
max(the max subarray sum, the total sum - the min subarray sum)

# **Prove of the second case**

max(prefix+suffix)  
= max(total sum - subarray)  
= total sum + max(-subarray)  
= total sum - min(subarray)

# **Corner case**

Just one to pay attention:  
If all numbers are negative, maxSum = max(A) and minSum = sum(A).  
In this case, max(maxSum, total - minSum) = 0, which means the sum of an empty subarray.  
According to the deacription, We need to return the max(A), instead of sum of am empty subarray.  
So we return the maxSum to handle this corner case.

# **Complexity**

One pass, time O(N)  
No extra space, space O(1)

**C++:**

int maxSubarraySumCircular(vector<int>& A) {

int total = 0, maxSum = A[0], curMax = 0, minSum = A[0], curMin = 0;

for (int& a : A) {

curMax = max(curMax + a, a);

maxSum = max(maxSum, curMax);

curMin = min(curMin + a, a);

minSum = min(minSum, curMin);

total += a;

}

return maxSum > 0 ? max(maxSum, total - minSum) : maxSum;

}

**Java:**

public int maxSubarraySumCircular(int[] A) {

int total = 0, maxSum = A[0], curMax = 0, minSum = A[0], curMin = 0;

for (int a : A) {

curMax = Math.max(curMax + a, a);

maxSum = Math.max(maxSum, curMax);

curMin = Math.min(curMin + a, a);

minSum = Math.min(minSum, curMin);

total += a;

}

return maxSum > 0 ? Math.max(maxSum, total - minSum) : maxSum;

}

**Python:**

def maxSubarraySumCircular(self, A):

total, maxSum, curMax, minSum, curMin = 0, A[0], 0, A[0], 0

for a in A:

curMax = max(curMax + a, a)

maxSum = max(maxSum, curMax)

curMin = min(curMin + a, a)

minSum = min(minSum, curMin)

total += a

return max(maxSum, total - minSum) if maxSum > 0 else maxSum