Maximum Subarray Product

Problem

Given an array of integers, find the contiguous subarray (containing at least one number) which has the maximum product and return its value.

This problem is a variation of the classic Kadane's Algorithm but for products, not sums - and that brings special challenges:

- Negative numbers can flip the product.
- Zeros reset the subarray product.
- Tracking both minimum and maximum becomes necessary.

Approach 1: Brute Force

Approach 1: Brute Force

Idea:

Try every possible subarray. For each, compute the product and track the maximum.

Steps:

- Loop through all possible subarrays using nested loops.
- For each subarray, calculate its product.
- Update the max product found so far.

Efficiency:

- Time Complexity: $O(n^3)$
- Space Complexity: O(1)

Approach 2: Kadane's-style

Approach 2: Kadane's-style with max/min tracking

Idea:

Maximum Subarray Product

Use a variation of Kadane's algorithm:

- Maintain both maxEndingHere and minEndingHere.
- When a negative number is encountered, the min product might become max (and vice versa).
- Reset on encountering zero, but don't ignore the current number.

Steps:

1. Initialize: maxSoFar = nums[0], maxEndingHere = nums[0], minEndingHere = nums[0]
2. Iterate through the array from index 1:
   - If nums[i] is negative, swap maxEndingHere and minEndingHere
   - Update:
     - maxEndingHere = max(nums[i], nums[i] * maxEndingHere)
     - minEndingHere = min(nums[i], nums[i] * minEndingHere)
   - Update maxSoFar = max(maxSoFar, maxEndingHere)

Efficiency:

- Time Complexity: O(n)
- Space Complexity: O(1)

Approach 3: Prefix and Suffix

Approach 3: Prefix and Suffix Product

Idea:

Zeros can break the subarray chain. So compute prefix and suffix products separately:

- From left to right and right to left, track product.
- Reset to 1 on encountering 0.
- Track max at each step from both directions.

Steps:

1. Initialize prefix = 1, suffix = 1, maxProduct = INT_MIN
2. Loop from left to right:

- prefix *= arr[i], if prefix == 0 reset to 1
- Update maxProduct

3. Loop from right to left:
- suffix *= arr[i], if suffix == 0 reset to 1
- Update maxProduct

Efficiency:

- Time Complexity: O(n)
- Space Complexity: O(1)

Test Cases

Test Cases:

Input: [2, 3, -2, 4]

Output: 6

Explanation: Subarray [2, 3] gives product 6

Input: [-2, 0, -1]

Output: 0

Explanation: Zero splits subarrays, max is 0

Input: [-2, -3, -4]

Output: 12

Explanation: -3 × -4 = 12 is maximum

Input: [0, -2, -3, 0, -4, -5]

Output: 20

Explanation: -4 × -5 = 20 after skipping zeros

Input: [1, -2, -3, 4]

Maximum Subarray Product

Output: 24

Explanation: Entire array gives max product 24