Maximum Product Subarray – Full Explanation

Problem:

Given an integer array nums, find a continuous subarray that has the largest product.

BRUTE FORCE APPROACH ($O(n^2)$)

Idea:

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

Dry Run Example:

nums = [2,3,-2,4]

Subarrays:

[2] = 2

[2,3] = 6

[2,3,-2] = -12

[2,3,-2,4] = -48

[3] = 3

[3,-2] = -6

[3,-2,4] = -24

[-2] = -2

[-2,4] = -8

[4] = 4

Maximum = 6

Java Brute Force:

```
class Solution {
public int maxProduct(int[] nums) {
int max = Integer.MIN_VALUE;
for (int i = 0; i < nums.length; i++) {
int prod = 1;
for (int j = i; j < nums.length; j++) {
prod *= nums[j];
max = Math.max(max, prod);
}
}
return max;
}
}
```

C++ Brute Force:

```
class Solution {
public:
int maxProduct(vector& nums) {
int maxVal = INT_MIN;
for (int i = 0; i < nums.size(); i++) {
int prod = 1;
for (int j = i; j < nums.size(); j++) {
prod *= nums[j];
maxVal = max(maxVal, prod);
}
}
```

return maxVal;

}

};

Python Brute Force:

class Solution:

def maxProduct(self, nums):

max_val = -10**9

for i in range(len(nums)):

prod = 1

for j in range(i, len(nums)):

prod *= nums[j]

max_val = max(max_val, prod)

return max_val

OPTIMAL APPROACH (Prefix + Suffix) O(n)

Idea:

Maintain prefix and suffix products.

Reset when hitting zero.

This captures max product subarrays even when negatives flip signs.

Dry Run:

nums = [2,3,-2,4]

Prefix:

2 → 6 → -12 → -48 → max = 6

Suffix:

4 → -8 → -24 → -48 → max = 6

Final Answer = 6

Java Efficient:

```
class Solution {
public int maxProduct(int[] nums) {
int n = nums.length;
int prefix = 1, suffix = 1;
int max = Integer.MIN_VALUE;
for (int i = 0; i < n; i++) {
prefix = (prefix == 0 ? 1 : prefix) * nums[i];
suffix = (suffix == 0 ? 1 : suffix) * nums[n - 1 - i];
max = Math.max(max, Math.max(prefix, suffix));
}
return max;
}
}
```

C++ Efficient:

```
class Solution {
public:
int maxProduct(vector& nums) {
int n = nums.size();
int prefix = 1, suffix = 1;
int maxVal = INT_MIN;
for (int i = 0; i < n; i++) {
```

```
prefix = (prefix == 0 ? 1 : prefix) * nums[i];

suffix = (suffix == 0 ? 1 : suffix) * nums[n - 1 - i];

maxVal = max(maxVal, max(prefix, suffix));

}

return maxVal;

}

};
```

Python Efficient:

```
class Solution:

def maxProduct(self, nums):

prefix = 1

suffix = 1

max_val = -10**9

n = len(nums)

for i in range(n):

prefix = (prefix or 1) * nums[i]

suffix = (suffix or 1) * nums[n - 1 - i]

max_val = max(max_val, prefix, suffix)

return max_val
```