Maximum Product Subarray in an Array

Problem Statement: Given an array that contains both negative and positive integers, find the maximum product subarray.

Example 1:

Input:

```
Nums = [1,2,3,4,5,0]
```

Output:

```
120
```

Explanation:

```
In the given array, we can see 1×2×3×4×5 gives maximum product value.
```

Example 2:

Input:

```
Nums = [1,2,-3,0,-4,-5]
```

Output:

```
20
```

Explanation:

```
In the given array, we can see (-4)×(-5) gives maximum product value.
```

Brute Force Approach

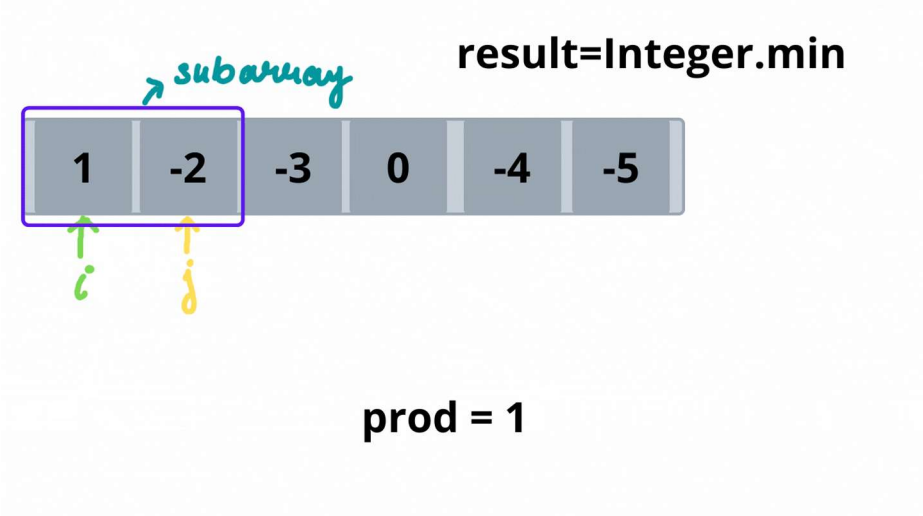
Approach:

Find all possible subarrays of the given array. Find the product of each subarray. Return the maximum of all them.

Following are the steps for the approach:-

- Run a loop on the array to choose the start point for each subarray.
- Run a nested loop to get the end point for each subarray.
- Multiply elements present in the chosen range.

Dry Run:




```
#include<bits/stdc++.h>
using namespace std;

int maxProductSubArray(vector<int>& nums) {
    int result = INT_MIN;
    for(int i=0;i<nums.size()-1;i++) {
        for(int j=i+1;j<nums.size();j++) {
            int prod = 1;
            for(int k=i;k<=j;k++)
                prod *= nums[k];
            result = max(result,prod);
        }
```

```
    }
    return result;
}

int main() {
    vector<int> nums = {1,2,-3,0,-4,-5};
    cout<<"The maximum product subarray: "<<maxProductSubArray(nums);
    return 0;
}
```

Output: The maximum product subarray: 20

Time Complexity: O(N^3)

Reason: We are using 3 nested loops for finding all possible subarrays and their product.

Space Complexity: O(1)

Reason: No extra data structure was used

Better Approach:

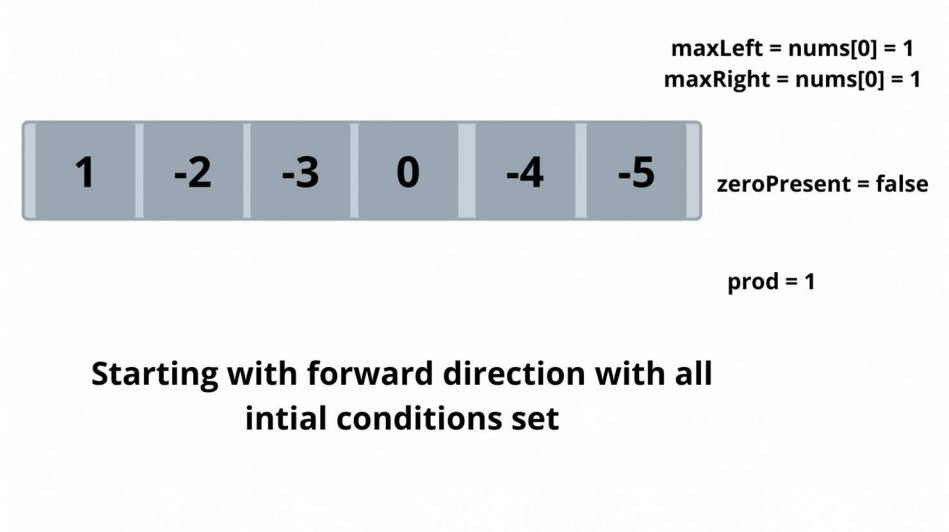
Approach:

We can optimize the brute force by making 3 nested iterations to 2 nested iterations

Following are the steps for the approach:

- Run a loop to find the start of the subarrays.
- Run another nested loop
- Multiply each element and store the maximum value of all the subarray.

Dry Run:




```
#include<bits/stdc++.h>
using namespace std;

int maxProductSubArray(vector<int>& nums) {
    int result = nums[0];
    for(int i=0;i<nums.size()-1;i++) {
        int p = nums[i];
        for(int j=i+1;j<nums.size();j++) {
          result = max(result,p);
          p *= nums[j];
        }
        result = max(result,p);//manages (n-1)th term
    }
    return result;
}
```

```
int main() {
   vector<int> nums = {1,2,-3,0,-4,-5};
   cout<<"The maximum product subarray: "<<maxProductSubArray(nums);
   return 0;
}
```

Complexity Analysis

Time Complexity: O(N^2)

Reason: We are using two nested loops

Space Complexity: O(1)

Reason: No extra data structures are used for computation

Algorithm / Intuition

Approach:

The following approach is motivated by Kandane's algorithm. To know Kadane's Algorithm follow **Kadane's Algorithm**

The pick point for this problem is that we can get the maximum product from the product of two negative numbers too.

Following are the steps for the approach:

- Initially store 0th index value in prod1, prod2 and result.
- Traverse the array from 1st index.
- For each element, update prod1 and prod2.
- Prod1 is maximum of current element, product of current element and prod1, product of current element and prod2
- Prod2 is minimum of current element, product of current element and prod1, product of current element and prod2
- Return maximum of result and prod1

```
#include<bits/stdc++.h>
using namespace std;

int maxProductSubArray(vector<int>& nums) {
   int prod1 = nums[0],prod2 = nums[0],result = nums[0];

   for(int i=1;i<nums.size();i++) {
      int temp = max({nums[i],prod1*nums[i],prod2*nums[i]});
      prod2 = min({nums[i],prod1*nums[i],prod2*nums[i]});
      prod1 = temp;

      result = max(result,prod1);
   }

   return result;
}

int main() {
   vector<int> nums = {1,2,-3,0,-4,-5};
   cout<<"The maximum product subarray: "<<maxProductSubArray(nums);
   return 0;
}
```

Complexity Analysis

Time Complexity: O(N)

Reason: A single iteration is used.

Space Complexity: O(1)

Reason: No extra data structure is used for computation