Maximum Product Subarray

Given an array that contains both positive and negative integers, find the product of the maximum product subarray. Expected Time complexity is O(n) and only O(1) extra space can be used.

Examples:

```
Input: arr[] = {6, -3, -10, 0, 2}
Output:   180  // The subarray is {6, -3, -10}

Input: arr[] = {-1, -3, -10, 0, 60}
Output:   60  // The subarray is {60}

Input: arr[] = {-2, -3, 0, -2, -40}
Output:   80  // The subarray is {-2, -40}
```

Recommended: Please solve it on "*PRACTICE*" first, before moving on to the solution.

The following solution assumes that the given input array always has a positive output. The solution works for all cases mentioned above. It doesn't work for arrays like {0, 0, -20, 0}, {0, 0, 0}.. etc. The solution can be easily modified to handle this case.
It is similar to Largest Sum Contiguous Subarray problem. The only thing to note here is, maximum product can also be obtained by minimum (negative) product ending with the previous element multiplied by this element. For example, in array {12, 2, -3, -5, -6, -2}, when we are at element -2, the maximum product is multiplication of, minimum product ending with -6 and -2.

C/C++

```
// C program to find Maximum Product Subarray
#include <stdio.h>

// Utility functions to get minimum of two integers
int min (int x, int y) {return x < y? x : y; }

// Utility functions to get maximum of two integers
int max (int x, int y) {return x > y? x : y; }

/* Returns the product of max product subarray.
   Assumes that the given array always has a subarray
   with product more than 1 */
int maxSubarrayProduct(int arr[], int n)
{
    // max positive product ending at the current position
    int max_ending_here = 1;

    // min negative product ending at the current position
    int min_ending_here = 1;
```

```
    // Initialize overall max product
    int max_so_far = 1;

    /* Traverse through the array. Following values are
       maintained after the i'th iteration:
       max_ending_here is always 1 or some positive product
                       ending with arr[i]
       min_ending_here is always 1 or some negative product
                       ending with arr[i] */
    for (int i = 0; i < n; i++)
    {
        /* If this element is positive, update max_ending_here.
           Update min_ending_here only if min_ending_here is
           negative */
        if (arr[i] > 0)
        {
            max_ending_here = max_ending_here*arr[i];
            min_ending_here = min (min_ending_here * arr[i], 1);
        }

        /* If this element is 0, then the maximum product
           cannot end here, make both max_ending_here and
           min_ending_here 0
           Assumption: Output is alway greater than or equal
                       to 1. */
        else if (arr[i] == 0)
        {
            max_ending_here = 1;
            min_ending_here = 1;
        }

        /* If element is negative. This is tricky
           max_ending_here can either be 1 or positive.
           min_ending_here can either be 1 or negative.
           next min_ending_here will always be prev.
           max_ending_here * arr[i] next max_ending_here
           will be 1 if prev min_ending_here is 1, otherwise
           next max_ending_here will be prev min_ending_here *
           arr[i] */
        else
        {
            int temp = max_ending_here;
            max_ending_here = max (min_ending_here * arr[i], 1);
            min_ending_here = temp * arr[i];
        }

        // update max_so_far, if needed
        if (max_so_far <  max_ending_here)
          max_so_far  =  max_ending_here;
    }

    return max_so_far;
}

// Driver Program to test above function
int main()
{
    int arr[] = {1, -2, -3, 0, 7, -8, -2};
    int n = sizeof(arr)/sizeof(arr[0]);
    printf("Maximum Sub array product is %d",
           maxSubarrayProduct(arr, n));
    return 0;
}
```

Run on IDE

Java

```
// Java program to find maximum product subarray
import java.io.*;

class ProductSubarray {

    // Utility functions to get minimum of two integers
    static int min (int x, int y) {return x < y? x : y; }

    // Utility functions to get maximum of two integers
    static int max (int x, int y) {return x > y? x : y; }

    /* Returns the product of max product subarray.
       Assumes that the given array always has a subarray
```

```
       with product more than 1 */
    static int maxSubarrayProduct(int arr[])
    {
        int n = arr.length;
        // max positive product ending at the current position
        int max_ending_here = 1;

        // min negative product ending at the current position
        int min_ending_here = 1;

        // Initialize overall max product
        int max_so_far = 1;

        /* Traverse through the array. Following
           values are maintained after the ith iteration:
           max_ending_here is always 1 or some positive product
                           ending with arr[i]
           min_ending_here is always 1 or some negative product
                           ending with arr[i] */
        for (int i = 0; i < n; i++)
        {
            /* If this element is positive, update max_ending_here.
                Update min_ending_here only if min_ending_here is
                negative */
            if (arr[i] > 0)
            {
                max_ending_here = max_ending_here*arr[i];
                min_ending_here = min (min_ending_here * arr[i], 1);
            }

            /* If this element is 0, then the maximum product cannot
               end here, make both max_ending_here and min_ending
              _here 0
               Assumption: Output is alway greater than or equal to 1. */
            else if (arr[i] == 0)
            {
                max_ending_here = 1;
                min_ending_here = 1;
            }

            /* If element is negative. This is tricky
               max_ending_here can either be 1 or positive.
               min_ending_here can either be 1 or negative.
               next min_ending_here will always be prev.
               max_ending_here * arr[i]
               next max_ending_here will be 1 if prev
               min_ending_here is 1, otherwise
               next max_ending_here will be
                           prev min_ending_here * arr[i] */
            else
            {
                int temp = max_ending_here;
                max_ending_here = max (min_ending_here * arr[i], 1);
                min_ending_here = temp * arr[i];
            }

            // update max_so_far, if needed
            if (max_so_far <  max_ending_here)
                max_so_far  =  max_ending_here;
        }

        return max_so_far;
    }

    public static void main (String[] args) {

        int arr[] = {1, -2, -3, 0, 7, -8, -2};
        System.out.println("Maximum Sub array product is "+
                           maxSubarrayProduct(arr));
    }
}/*This code is contributed by Devesh Agrawal*/
```

Run on IDE

Python

```
# Python program to find maximum product subarray

# Returns the product of max product subarray.
# Assumes that the given array always has a subarray
# with product more than 1
```

```
def maxsubarrayproduct(arr):

    n = len(arr)

    # max positive product ending at the current position
    max_ending_here = 1

    # min positive product ending at the current position
    min_ending_here = 1

    # Initialize maximum so far
    max_so_far = 1

    # Traverse throughout the array. Following values
    # are maintained after the ith iteration:
    # max_ending_here is always 1 or some positive product
    # ending with arr[i]
    # min_ending_here is always 1 or some negative product
    # ending with arr[i]
    for i in range(0,n):

        # If this element is positive, update max_ending_here.
        # Update min_ending_here only if min_ending_here is
        # negative
        if arr[i] > 0:
            max_ending_here = max_ending_here*arr[i]
            min_ending_here = min (min_ending_here * arr[i], 1)

        # If this element is 0, then the maximum product cannot
        # end here, make both max_ending_here and min_ending_here 0
        # Assumption: Output is alway greater than or equal to 1.
        elif arr[i] == 0:
            max_ending_here = 1
            min_ending_here = 1

        # If element is negative. This is tricky
        # max_ending_here can either be 1 or positive.
        # min_ending_here can either be 1 or negative.
        # next min_ending_here will always be prev.
        # max_ending_here * arr[i]
        # next max_ending_here will be 1 if prev
        # min_ending_here is 1, otherwise
        # next max_ending_here will be prev min_ending_here * arr[i]
        else:
            temp = max_ending_here
            max_ending_here = max (min_ending_here * arr[i], 1)
            min_ending_here = temp * arr[i]
        if (max_so_far <  max_ending_here):
            max_so_far  =  max_ending_here
    return max_so_far

# Driver function to test above function
arr = [1, -2, -3, 0, 7, -8, -2]
print "Maximum product subarray is",maxsubarrayproduct(arr)

# This code is contributed by Devesh Agrawal
```

Run on IDE

Output:

```
Maximum Sub array product is 112
```

Time Complexity: O(n)

Auxiliary Space: O(1)

Asked in: Amazon, Microsoft, Morgan-Stanley

This article is compiled by **Dheeraj Jain** and reviewed by GeeksforGeeks team. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

GATE CS Corner Company Wise Coding Practice

Arrays

Recommended Posts:

Find four elements that sum to a given value | Set 1 (n^3 solution)

(Login to Rate and Mark)

3.3 Average Difficulty : **3.3/5.0**
Based on **142** vote(s)

Add to TODO List

Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!