

Maximum Product Subarray - GeeksforGeeks

Source: <https://www.geeksforgeeks.org/maximum-product-subarray/>

Courses Tutorials Practice Jobs DSA Tutorial Interview Questions Quizzes Must Do Advanced DSA System Design Aptitude Puzzles Interview Corner DSA Python Technical Scripter 2026 Explore DSA Fundamentals Logic Building Problems Analysis of Algorithms Data Structures Array Data Structure String in Data Structure Hashing in Data Structure Linked List Data Structure Stack Data Structure Queue Data Structure Tree Data Structure Graph Data Structure Trie Data Structure Algorithms Searching Algorithms Sorting Algorithms Introduction to Recursion Greedy Algorithms Tutorial Graph Algorithms Dynamic Programming or DP Bitwise Algorithms Advanced Segment Tree Binary Indexed Tree or Fenwick Tree Square Root (Sqrt) Decomposition Algorithm Binary Lifting Geometry Interview Preparation Interview Corner GfG160 Practice Problem GeeksforGeeks Practice - Leading Online Coding Platform Problem of The Day - Develop the Habit of Coding DSA Course 90% Refund Maximum Product Subarray Last Updated : 22 Jul, 2025 Given an array `arr[]` consisting of positive, negative, and zero values, find the maximum product that can be obtained from any contiguous subarray of `arr[]`. Examples: Input: `arr[] = [-2, 6, -3, -10, 0, 2]` Output: 180 Explanation: The subarray with maximum product is `[6, -3, -10]` with product = $6 * (-3) * (-10) = 180$. Input: `arr[] = [-1, -3, -10, 0, 6]` Output: 30 Explanation: The subarray with maximum product is `[-3, -10]` with product = $(-3) * (-10) = 30$. Input: `arr[] = [2, 3, 4]` Output: 24 Explanation: For an array with all positive elements, the result is product of all elements. Try it on GfG Practice Table of Content [Naive Approach] Using Two Nested Loops – $O(n^2)$ Time and $O(1)$ Space [Expected Approach - 1] Greedy Min-Max Product - $O(n)$ Time and $O(1)$ Space [Expected Approach - 2] By Traversing in Both Directions - $O(n)$ Time and $O(1)$ Space [Naive Approach] Using Two Nested Loops – $O(n^2)$ Time and $O(1)$ Space The idea is to traverse over every contiguous subarray, find the product of each of these subarrays and return the maximum product among all the subarrays. C++ #include <iostream> #include <vector> using namespace std; int maxProduct (vector < int > & arr) { int n = arr . size (); // Initializing result int maxProd = arr [0]; for (int i = 0 ; i < n ; i ++) { int mul = 1 ; // traversing in current subarray for (int j = i ; j < n ; j ++) { mul *= arr [j]; // updating result every time // to keep track of the maximum product maxProd = max (maxProd , mul); } } return maxProd ; } int main () { vector < int > arr = { -2 , 6 , -3 , -10 , 0 , 2 }; cout << maxProduct (arr); return 0 ; } C #include <stdio.h> int maxProduct (int arr [], int n) { // Initializing result int maxProd = arr [0]; for (int i = 0 ; i < n ; i ++) { int mul = 1 ; // traversing in current subarray for (int j = i ; j < n ; j ++) { mul *= arr [j]; // updating result every time // to keep track of the maximum product maxProd = (maxProd > mul) ? maxProd : mul ; } } return maxProd ; } int main () { int arr [] = { -2 , 6 , -3 , -10 , 0 , 2 }; int n = sizeof (arr) / sizeof (arr [0]); printf ("%lld \n " , maxProduct (arr , n)); return 0 ; } Java class GfG { static int maxProduct (int arr []) { int n = arr . length ; // Initializing result int maxProd = arr [0]; for (int i = 0 ; i < n ; i ++) { int mul = 1 ; // traversing in current subarray for (int j = i ; j < n ; j ++) { mul *= arr [j]; // updating result every time // to keep track of the maximum product maxProd = Math . max (maxProd , mul); } } return maxProd ; } public static void main (String [] args) { int arr [] = { - 2 , 6 , - 3 , - 10 , 0 , 2 }; System . out . println (maxProduct (arr)); } } Python def maxProduct (arr): n = len (arr) # Initializing result maxProd = arr [0] for i in range (n): mul = 1 # traversing in current subarray for j in range (i , n): mul *= arr [j] # updating result every time # to keep track of the maximum product maxProd = max (maxProd , mul) return maxProd if __name__ == "__main__": arr = [- 2 , 6 , - 3 , - 10 , 0 , 2] print (maxProduct (arr)) C# using System ; class GfG { static int maxProduct (int [] arr) { int n = arr . Length ; // Initializing result int maxProd = arr [0]; for (int i = 0 ; i < n ; i ++) { int mul = 1 ; // traversing in current subarray for (int j = i ; j < n ; j ++) { mul *= arr [j]; // updating result every time // to keep track of the maximum product maxProd = Math . Max (maxProd , mul); } } return maxProd ; } public static void Main (String [] args) { int [] arr = { - 2 , 6 , - 3 , - 10 , 0 , 2 }; Console . Write (maxProduct (arr)); } } JavaScript function maxProduct (arr) { const n = arr . length ; // Initializing result let maxProd = arr [0]; for (let i = 0 ; i < n ; i ++) { let mul = 1 ; // Traversing in current subarray for (let j = i ; j < n ; j ++) { mul *= arr [j]; // Update result to keep track of the maximum product if (mul > maxProd) maxProd = mul ; } } return maxProd ; } // Driver Code const arr = [- 2 , 6 , - 3 , - 10 , 0 , 2]; console . log (maxProduct (arr) . toString ()); Output 180 [Expected Approach - 1] Greedy Min-Max Product - $O(n)$ Time and $O(1)$ Space Let's assume that the input array has only positive elements. Then, we can simply iterate from left to right keeping track of the maximum running product ending at

any index. The maximum product would be the product ending at the last index. The problem arises when we encounter zero or a negative element. If we encounter zero, then all the subarrays containing this zero will have product = 0, so zero simply resets the product of the subarray. If we encounter a negative number, we need to keep track of the minimum product as well as the maximum product ending at the previous index. This is because when we multiply the minimum product with a negative number, it can give us the maximum product. So, keeping track of minimum product ending at any index is important as it can lead to the maximum product on encountering a negative number.

Step By Step Approach: Initialize three variables: currMax, currMin, and maxProd with the first element of the array. Loop through the array from index 1 to end to evaluate every position's contribution. For each index, calculate the temporary maximum using max of current value, current \times currMax, and current \times currMin. Update currMin using min of current value, current \times currMax, and current \times currMin. Assign currMax to the previously computed temporary maximum value. Update maxProd by comparing it with the updated currMax value. Finally, return maxProd which stores the maximum product of any subarray.

Working:

```
C++ #include <iostream> #include <vector> #include <algorithm> using namespace std;
int maxProduct ( vector < int > & arr ) { int n = arr . size (); // max product ending at the current index
int currMax = arr [ 0 ]; // min product ending at the current index
int currMin = arr [ 0 ]; // Initialize overall max product
int maxProd = arr [ 0 ]; // Iterate through the array for ( int i = 1 ; i < n ; i ++ ) { // Temporary variable to store the maximum product ending // at the current index
int temp = max ( { arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin } ); // Update the minimum product ending at the current index
currMin = min ( { arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin } ); // Update the maximum product ending at the current index
currMax = temp ; // Update the overall maximum product
maxProd = max ( maxProd , currMax ); } return maxProd ; }

int main () { vector < int > arr = { -2 , 6 , -3 , -10 , 0 , 2 }; cout << maxProduct ( arr ); return 0 ; }

C #include <stdio.h> #include <limits.h> int max ( int a , int b , int c ) { if ( a >= b && a >= c ) return a ; if ( b >= a && b >= c ) return b ; return c ; }
int min ( int a , int b , int c ) { if ( a <= b && a <= c ) return a ; if ( b <= a && b <= c ) return b ; return c ; }
int maxProduct ( int arr [], int n ) { // max product ending at the current index
int currMax = arr [ 0 ]; // min product ending at the current index
int currMin = arr [ 0 ]; // Initialize overall max product
int maxProd = arr [ 0 ]; // Iterate through the array for ( int i = 1 ; i < n ; i ++ ) { // Temporary variable to store the maximum product ending at the // current index
int temp = max ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ); // Update the minimum product ending at the current index
currMin = min ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ); // Update the maximum product ending at the current index
currMax = temp ; // Update the overall maximum product
maxProd = max ( maxProd , currMax , maxProd ); } return maxProd ; }

int main () { int arr [] = { -2 , 6 , -3 , -10 , 0 , 2 }; int n = sizeof ( arr ) / sizeof ( arr [ 0 ] ); printf ( "%lld \n" , maxProduct ( arr , n ) ); return 0 ; }

Java class GfG { static int max ( int a , int b , int c ) { return Math . max ( a , Math . max ( b , c ) ); } static int min ( int a , int b , int c ) { return Math . min ( a , Math . min ( b , c ) ); } static int maxProduct ( int [] arr ) { int n = arr . length ; // max product ending at the current index
int currMax = arr [ 0 ]; // min product ending at the current index
int currMin = arr [ 0 ]; // Initialize overall max product
int maxProd = arr [ 0 ]; // Iterate through the array for ( int i = 1 ; i < n ; i ++ ) { // Temporary variable to store the maximum product ending // at the current index
int temp = max ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ); // Update the minimum product ending at the current index
currMin = min ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ); // Update the maximum product ending at the current index
currMax = temp ; // Update the overall maximum product
maxProd = Math . max ( maxProd , currMax ); } return maxProd ; } public static void main ( String [] args ) { int [] arr = { -2 , 6 , -3 , -10 , 0 , 2 }; System . out . println ( maxProduct ( arr ) ); } }

Python def maxProduct ( arr ): n = len ( arr ) # max product ending at the current index
currMax = arr [ 0 ] # min product ending at the current index
currMin = arr [ 0 ] # Initialize overall max product
maxProd = arr [ 0 ] # Iterate through the array for i in range ( 1 , n ): # Temporary variable to store the maximum product ending # at the current index
temp = max ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ) # Update the minimum product ending at the current index
currMin = min ( arr [ i ], arr [ i ] * currMax , arr [ i ] * currMin ) # Update the maximum product ending at the current index
currMax = temp # Update the overall maximum product
maxProd = max ( maxProd , currMax ) return maxProd

if __name__ == "__main__" : arr = [ -2 , 6 , -3 , -10 , 0 , 2 ] print ( maxProduct ( arr ) )

C# using System ; class GfG { static int maxProduct ( int [] arr ) { int n = arr . Length ; // max product ending at the current index
int currMax = arr [ 0 ]; // min product ending at the current index
int currMin = arr [ 0 ]; // Initialize overall max product
int maxProd = arr [ 0 ]; // Iterate through the array for ( int i = 1 ; i < n ; i ++ ) { // Temporary variable to store the maximum product ending // at the current index
int temp = Math . Max ( arr [ i ], Math . Max ( arr [ i ] * currMax , arr [ i ] * currMin ) ); // Update the minimum product ending at the current index
currMin = Math . Min ( arr [ i ], Math . Min ( arr [ i ] * currMax , arr [ i ] * 
```

```

currMin )); // Update the maximum product ending at the current index currMax = temp ; // Update the
overall maximum product maxProd = Math . Max ( maxProd , currMax ); } return maxProd ; } static void
Main () { int [] arr = { - 2 , 6 , - 3 , - 10 , 0 , 2 }; Console . WriteLine ( maxProduct ( arr )); } } JavaScript
function max ( a , b , c ) { return a > b ? ( a > c ? a : c ) : ( b > c ? b : c ); } function min ( a , b , c ) { return
a < b ? ( a < c ? a : c ) : ( b < c ? b : c ); } function maxProduct ( arr ) { // Initialize the maximum and
minimum products ending at // the current index let currMax = arr [ 0 ]; let currMin = arr [ 0 ]; // Initialize
the overall maximum product let maxProd = arr [ 0 ]; // Iterate through the array starting from the
second element for ( let i = 1 ; i < arr . length ; i ++ ) { // Calculate potential maximum product at this
index const temp = max ( arr [ i ] * currMax , arr [ i ] * currMin , arr [ i ] ); // Update the minimum product
ending at the current index currMin = min ( arr [ i ] * currMax , arr [ i ] * currMin , arr [ i ] ); // Update the
maximum product ending at the current index currMax = temp ; // Update the overall maximum product
maxProd = max ( maxProd , maxProd , currMax ); } return maxProd ; } // Driver Code const arr = [ - 2 , 6
, - 3 , - 10 , 0 , 2 ]; console . log ( maxProduct ( arr ) . toString ( )); Output 180 [Expected Approach - 2] By
Traversing in Both Directions - O(n) Time and O(1) Space We will follow a simple approach that is to
traverse from the start and keep track of the running product and if the running product is greater than
the max product, then we update the max product. Also, if we encounter '0' then make product of all
elements till now equal to 1 because from the next element, we will start a new subarray. But what is
the problem with this approach? Problem will occur when our array will contain odd no. of negative
elements. In that case, we have to reject one negative element so that we can even no. of negative
elements and their product can be positive . Now, since subarray should be contiguous so we can't
simply reject any one negative element. We have to either reject the first negative element or the last
negative element. Now, if we traverse from start then only the last negative element can be rejected
and if we traverse from the last then the first negative element can be rejected. So we will traverse from
both ends and find the maximum product subarray. Illustration: C++ #include <iostream> #include
<vector> #include <climits> #include <algorithm> using namespace std ; int maxProduct ( vector < int >
& arr ) { int n = arr . size ( ); int maxProd = INT_MIN ; // leftToRight to store product from left to Right int
leftToRight = 1 ; // rightToLeft to store product from right to left int rightToLeft = 1 ; for ( int i = 0 ; i < n ; i
++ ) { if ( leftToRight == 0 ) leftToRight = 1 ; if ( rightToLeft == 0 ) rightToLeft = 1 ; // calculate product
from index left to right leftToRight *= arr [ i ] ; // calculate product from index right to left int j = n - i - 1 ;
rightToLeft *= arr [ j ] ; maxProd = max ( { leftToRight , rightToLeft , maxProd } ); } return maxProd ; } int
main () { vector < int > arr = { -2 , 6 , -3 , -10 , 0 , 2 }; cout << maxProduct ( arr ); return 0 ; } C #include
<stdio.h> #include <limits.h> int maxProduct ( int arr [], int n ) { int maxProd = LLONG_MIN ; //
leftToRight to store product from left to Right int leftToRight = 1 ; // rightToLeft to store product from
right to left int rightToLeft = 1 ; for ( int i = 0 ; i < n ; i ++ ) { if ( leftToRight == 0 ) leftToRight = 1 ; if (
rightToLeft == 0 ) rightToLeft = 1 ; // calculate product from index left to right leftToRight *= arr [ i ] ; //
calculate product from index right to left int j = n - i - 1 ; rightToLeft *= arr [ j ] ; maxProd = ( leftToRight >
maxProd ? leftToRight : maxProd ); maxProd = ( rightToLeft > maxProd ? rightToLeft : maxProd ); }
return maxProd ; } int main () { int arr [] = { -2 , 6 , -3 , -10 , 0 , 2 }; int n = sizeof ( arr ) / sizeof ( arr [ 0 ] );
printf ( "%lld\n " , maxProduct ( arr , n )); return 0 ; } Java class GfG { static int maxProduct ( int [] arr ) {
int n = arr . length ; int maxProd = Integer . MIN_VALUE ; // leftToRight to store product from left to
Right int leftToRight = 1 ; // rightToLeft to store product from right to left int rightToLeft = 1 ; for ( int i = 0
; i < n ; i ++ ) { if ( leftToRight == 0 ) leftToRight = 1 ; if ( rightToLeft == 0 ) rightToLeft = 1 ; // calculate
product from index left to right leftToRight *= arr [ i ] ; // calculate product from index right to left int j = n
- i - 1 ; rightToLeft *= arr [ j ] ; maxProd = Math . max ( leftToRight , Math . max ( rightToLeft , maxProd
)); } return maxProd ; } public static void main ( String [] args ) { int [] arr = { - 2 , 6 , - 3 , - 10 , 0 , 2 };
System . out . println ( maxProduct ( arr )); } } Python def maxProduct ( arr ): n = len ( arr ) maxProd =
float ( '-inf' ) # leftToRight to store product from left to Right leftToRight = 1 # rightToLeft to store product
from right to left rightToLeft = 1 for i in range ( n ): if leftToRight == 0 : leftToRight = 1 if rightToLeft == 0
: rightToLeft = 1 # calculate product from index left to right leftToRight *= arr [ i ] # calculate product
from index right to left j = n - i - 1 rightToLeft *= arr [ j ] maxProd = max ( leftToRight , rightToLeft ,
maxProd ) return maxProd if __name__ == "__main__" : arr = [ - 2 , 6 , - 3 , - 10 , 0 , 2 ] print (
maxProduct ( arr )) C# using System ; class GfG { static int maxProduct ( int [] arr ) { int n = arr . Length
; int maxProd = int . MinValue ; // leftToRight to store product from left to Right int leftToRight = 1 ; //
rightToLeft to store product from right to left int rightToLeft = 1 ; for ( int i = 0 ; i < n ; i ++ ) { if (
leftToRight == 0 ) leftToRight = 1 ; if ( rightToLeft == 0 ) rightToLeft = 1 ; // calculate product from index
left to right leftToRight *= arr [ i ] ; // calculate product from index right to left int j = n - i - 1 ; rightToLeft *=
arr [ j ] ; maxProd = Math . Max ( leftToRight , Math . Max ( rightToLeft , maxProd )); } return maxProd ; }

```

```
static void Main () { int [] arr = { - 2 , 6 , - 3 , - 10 , 0 , 2 }; Console . WriteLine ( maxProduct ( arr )); } }
JavaScript function maxProduct ( arr ) { let n = arr . length ; let maxProd = Number .
MIN_SAFE_INTEGER ; // leftToRight to store product from left to Right let leftToRight = 1 ; //
rightToLeft to store product from right to left let rightToLeft = 1 ; for ( let i = 0 ; i < n ; i ++ ) { if (
leftToRight === 0 ) leftToRight = 1 ; if ( rightToLeft === 0 ) rightToLeft = 1 ; // calculate product from
index left to right leftToRight *= arr [ i ]; // calculate product from index right to left let j = n - i - 1 ;
rightToLeft *= arr [ j ]; maxProd = Math . max ( leftToRight , rightToLeft , maxProd ); } return maxProd ; }
// Driver Code let arr = [ - 2 , 6 , - 3 , - 10 , 0 , 2 ]; console . log ( maxProduct ( arr )); Output 180
Comment Article Tags: Article Tags: DSA Arrays Microsoft Amazon Morgan Stanley Myntra
Myntra-Question + 3 More
```