

closed

Demo ticket

Session**ID:** demoVF7CFF-RMA**Time limit:** 120 min.**Status:** closed**Started on:** 2014-01-10 15:50 UTC

Score:

100

of 100

★ 1. MaxProductOfThree

Maximize $A[P] * A[Q] * A[R]$ for any triplet (P, Q, R) .**score: 100 of 100****Task description**

A non-empty zero-indexed array A consisting of N integers is given. The *product* of triplet (P, Q, R) equates to $A[P] * A[Q] * A[R]$ ($0 \leq P < Q < R < N$).

For example, array A such that:

```
A[0] = -3
A[1] = 1
A[2] = 2
A[3] = -2
A[4] = 5
A[5] = 6
```

contains the following example triplets:

- $(0, 1, 2)$, product is $-3 * 1 * 2 = -6$
- $(1, 2, 4)$, product is $1 * 2 * 5 = 10$
- $(2, 4, 5)$, product is $2 * 5 * 6 = 60$

Your goal is to find the maximal product of any triplet. Write a function:

```
class Solution { public int solution(int[] A); }
```

that, given a non-empty zero-indexed array A , returns the value of the maximal product of any triplet.

For example, given array A such that:

```
A[0] = -3
A[1] = 1
A[2] = 2
A[3] = -2
A[4] = 5
A[5] = 6
```

the function should return 60, as the product of triplet $(2, 4, 5)$ is maximal.

Assume that:

- N is an integer within the range $[3..100,000]$;
- each element of array A is an integer within the range $[-1,000..1,000]$.

Complexity:

- expected worst-case time complexity is $O(N \log(N))$;
- expected worst-case space complexity is $O(1)$, beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

Copyright 2009–2014 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution**Programming language used:** C#**Total time used:** 1 minutes**Effective time used:** 1 minutes**Notes:** correct functionality and scalability**Task timeline**

15:50:58

15:51:57

Code: 15:51:57 UTC, cs, final, score: 100.00

```
01. using System.Collections.Generic;
02. using System.Linq;
03. class Solution {
04.     public int solution(int[] A)
05.     {
06.         // write your code in C# with
07.         // .NET 2.0
08.         var sorted =
09.             Solution.sort(A.ToList<int>
10.                 ());
11.         var count = sorted.Count;
12.         var rightMaximum = sorted[count
13.             - 3] * sorted[count - 2] *
14.             sorted[count - 1];
15.         var leftMaximum = sorted[0] *
16.             sorted[1] * sorted[count -
17.                 1]; // In the event first
18.                 // 2 elements are negative
19.         return (rightMaximum >=
20.             leftMaximum ? rightMaximum :
21.             leftMaximum);
22.     }
23.
24.     // Merge sort not really required
25.     // as the Sort function provided by
26.     // .NET performs the same
27.     // functionality but just for
28.     // knowledge
29.     public static List<int>
30.         sort(List<int> value)
31.     {
32.         if (value.Count == 1)
33.             return value;
34.         var midIndex = (int)
35.             (value.Count / 2);
36.         var left = value.GetRange(0,
37.             midIndex);
38.         var right =
39.             value.GetRange(midIndex,
40.                 value.Count - midIndex);
41.
42.         left = sort(left);
43.         right = sort(right);
44.
45.         var leftIndex = 0;
```

```
28.         var rightIndex = 0;
29.         var result = new List<int>();
30.         for (var count = 0; count <
            left.Count + right.Count;
            count++)
31.         {
32.             if (leftIndex ==
                left.Count)
33.             {
34.                 // If no more elements
                // in the left index
                // append all sorted
                // elements from the
                // right array to the
                // result then exit
35.                 result.AddRange(right.GetRange(right
                    right.Count -
                    rightIndex));
36.                 break;
37.             }
38.             else if (rightIndex ==
                right.Count)
39.             {
40.                 // If no more elements
                // in the right index
                // append all sorted
                // elements from the
                // left array to the
                // result then exit
41.                 result.AddRange(left.GetRange(leftIr
                    left.Count -
                    leftIndex));
42.                 break;
43.             }
44.             else if (left[leftIndex] >
                right[rightIndex])
45.             {
46.                 result.Add(right[rightIndex]);
47.                 rightIndex++;
48.             }
49.             else
50.             {
51.                 result.Add(left[leftIndex]);
52.                 leftIndex++;
53.             }
54.         }
55.         return result;
56.     }
57. }
```

Analysis

Detected time complexity:
O(N * log(N))

Get account

test	time	result
example example test	0.080 s.	OK
one_triple three elements	0.080 s.	OK
simple1 simple tests	0.080 s.	OK
simple2 simple tests	0.080 s.	OK
small_random random small, length = 100	0.090 s.	OK
medium_range -1000, -999, ... 1000, length = ~1,000	0.090 s.	OK
medium_random random medium, length = ~10,000	0.110 s.	OK
large_random random large, length = ~100,000	0.330 s.	OK
large_range 2000 * (-10..10) + [-1000, 500, -1]	0.200 s.	OK
extreme_large (-2, .., -2, 1, .., 1) and (MAX_INT).. (MAX_INT), length = ~100,000	0.320 s.	OK