Find the Increasing subsequence of length three with maximum product

Given a sequence of non-negative integers, find the subsequence of length 3 having maximum product with the numbers of the subsequence being in ascending order.

Examples:

```
Input:
arr[] = {6, 7, 8, 1, 2, 3, 9, 10}
Output:
8 9 10

Input:
arr[] = {1, 5, 10, 8, 9}
Output: 5 8 9
```

Since we want to find the maximum product, we need to find following two things for every element in the given sequence:
LSL: The largest smaller element on left of given element
LGR: The largest greater element on right of given element.

Once we find LSL and LGR for an element, we can find the product of element with its LSL and LGR (if they both exist). We calculate this product for every element and return maximum of all products.

A **simple method** is to use nested loops. The outer loop traverses every element in sequence. Inside the outer loop, run two inner loops (one after other) to find LSL and LGR of current element. Time complexity of this method is O(n^2).

We can do this **in O(nLogn) time**. For simplicity, let us first create two arrays LSL[] and LGR[] of size n each where n is number of elements in input array arr[]. The main task is to fill two arrays LSL[] and LGR[]. Once we have these two arrays filled, all we need to find maximum product LSL[i]*arr[i]*LGR[i] where 0 < i < n-1 (Note that LSL[i] doesn't exist for i = 0 and LGR[i] doesn't exist for i = n-1). We can **fill LSL[]** in O(nLogn) time. The idea is to use a Balanced Binary Search Tree like AVL. We start with empty AVL tree, insert the leftmost element in it. Then we traverse the input array starting from the second element to second last element. For every element currently being traversed, we find the floor of it in AVL tree. If floor exists, we store the floor in LSL[], otherwise we store NIL. After storing the floor, we insert the current element in the AVL tree.

We can **fill LGR[]** in O(n) time. The idea is similar to this post. We traverse from right side and keep track of the largest element. If the largest element is greater than current element, we store it in LGR[], otherwise we store NIL.

Finally, we run a O(n) loop and **return maximum of LSL[i]*arr[i]*LGR[i]**

Overall complexity of this approach is O(nLogn) + O(n) + O(n) which is O(nLogn). Auxiliary space required is O(n). Note that we can avoid space required for LSL, we can find and use LSL values in final loop.

This article is contributed by **Amit Jain**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

148 Comments Category: Arrays

Related Posts:

- Longest Span with same Sum in two Binary arrays
- Count Inversions of size three in a give array
- Find the subarray with least average
- Count triplets with sum smaller than a given value
- Find zeroes to be flipped so that number of consecutive 1's is maximized
- Reorder an array according to given indexes
- Find maximum value of Sum(i*arr[i]) with only rotations on given array allowed
- Find maximum average subarray of k length

(Login to Rate and Mark)

5 Average Difficulty : **5/5.0**
Based on **4** vote(s)

Add to TODO List

Mark as DONE

Like Share 35 people like this. Be the first of your friends.

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