Array Closest Ascenders

codility
WE TEST CODERS

It's time to show you how the Codility challenge code-named π (Pi) can be solved. You can still give it a try, but no certificate will be granted.

In the π certificate problem, we are given an array of integers. For each element of the array we have to find the distance to the closest larger element in the array, called *the closest ascender*. (If there is no ascender, we assume that the distance equals zero.) The result is an array of distances to the closest ascenders.

Is this difficult? It depends on the desired time complexity. Let us focus for a while on a single element of the array, $A[I]$. One can easily find the distance to its closest ascender by increasing the distance D until $A[I-D]$ or $A[I+D]$ is larger than $A[I]$. This way, for a given element of the array, the distance can be computed in $O(N)$ time, and the whole result can be computed in $O(N^2)$ time. Such a solution is correct and simple, although it is not optimal. Here is its implementation in Python:

1: Array Closest Ascenders — $O(N^2)$

```
def array_closest_ascenders(A):
    N = len(A)
    ret = [0] * N
    for I in xrange(N):
        for D in xrange(1,N):
            if (I-D >= 0) and (A[I-D] > A[I]):
                ret[I] = D
                break
            if (I+D < N) and (A[I+D] > A[I]):
                ret[I] = D
                break
    return ret
```

Let us simplify the problem. Instead of looking for the distance to the closest ascender, we can look for two distances: that to the closest ascender lying to the left, and that to the closest ascender lying to the right, called respectively the closest left ascender and the closest right ascender. These problems are symmetrical, so let us focus just on finding the distances to the closest left ascenders.

It turns out that it is possible to find all the closest left ascenders in $O(N)$ time. The main observation is that not all elements of the array can possibly be closest left ascenders. Let $0 \leqslant K < J < I < N$. If $A[K] < A[J]$ then $A[K]$ cannot be the closest left ascender of $A[I]$. Let us filter the list $[0, 1, \ldots, I-1]$ and narrow it to include only the indices of those elements that can possibly be the closest left ascenders of $A[I]$, regardless of its actual value. Such a list of indices is increasing, and the elements to which these indices point form

a decreasing sequence. We process the elements of the array from left to right, updating the list of indices of possible closest left ascenders. When processing $A[I]$ we can remove from the list all indices of the elements not larger than $A[I]$. Since the sequence of indexed elements is decreasing, we remove some trailing part of the list of indices. The last element left is the closest left ascender of $A[I]$. Then, by appending I to the list of indices, we obtain a list of indices of possible closest left ascenders of $A[I+1]$. Below is a Python function calculating the array of distances to the closest left ascenders.

2: Left Closest Ascenders — $O(N)$

```
INF = 1000000123

def left_closest_ascenders(A):
    N = len(A)
    if (N == 0): return A
    ret = [0] * N
    stack = [0] * N
    stack_size = 0
    for I in xrange(N):
        while (stack_size > 0) and (A[stack[stack_size-1]] <= A[I]):
            stack_size -= 1
        if (stack_size == 0):
            ret[I] = INF
        else:
            ret[I] = I - stack[stack_size-1]
        stack[stack_size] = I
        stack_size += 1
    return ret
```

What is the time complexity of this function? The outer `for` loop performs N steps. The inner `while` loop can perform up to $O(N)$ steps. However, if we use amortized cost analysis, it turns out that the overall time complexity is smaller than $O(N^2)$. With each step of the `while` loop, `stack_size` decreases by 1. Its initial value is 0, it is increased by 1 with each step of the outer `for` loop, and it never takes negative values. Hence, the overall time complexity of this function is $O(N)$.

In other words, each element of the array is pushed onto the stack once and can be popped only once. Hence, the `while` loop does not influence the time complexity.

The array of distances to the closest right ascenders is just a reversed array of distances to the closest left ascenders, computed for the reversed array of elements. By combining the arrays of distances to the closest left and right ascenders, we instantly obtain the result. Below is a Python implementation of the solution's main function:

3: Array Closest Ascenders — $O(N)$

```
def array_closest_ascenders(A):
    N = len(A)
    left = left_closest_ascenders(A)
    A.reverse()
    right = left_closest_ascenders(A)
    right.reverse()
    ret = [0]*N
    for I in xrange(N):
        ret[I] = min(left[I], right[I])
        if ret[I] == INF:
            ret[I] = 0
    return ret
```