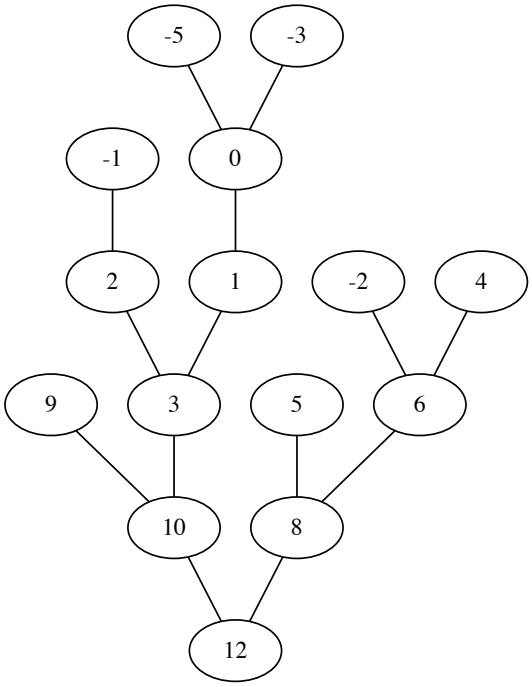
Cartesian Sequence

It is time to show how the Codility challenge codenamed Y (Upsilon) can be solved. You can still give it a try, but no certificate will be granted.

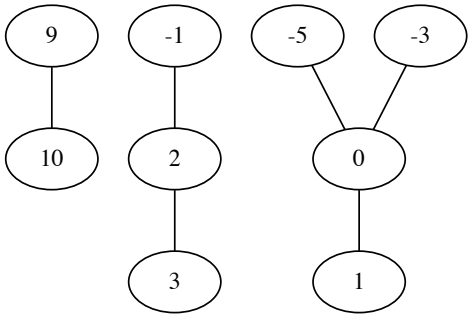
It is easier to describe the solution if we first introduce a notion of a *Cartesian tree*. We are given an array of N different integers. Cartesian tree is a binary tree whose nodes are elements of the array. The root is the maximal element in the array. The left and right subtrees are constructed recursively. The left subtree is a Cartesian tree build of all the elements to the left from the root, and the right subtree is a Cartesian tree build of all the elements to the right from the root. For the example array $[9, 10, 2, -1, 3, -5, 0, -3, 1, 12, 5, 8, -2, 6, 4]$ we obtain the following tree:

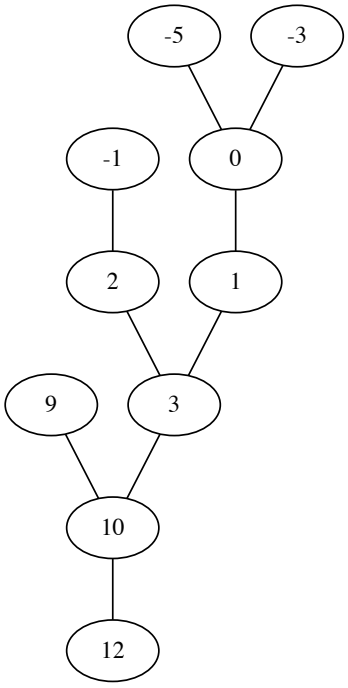


The task can be seen as looking for the height of the Cartesian tree. The solution resembles the algorithm building the Cartesian tree, but we don't have to store that much information. We scan the given array from left to right. In the algorithm building the Cartesian tree, scanned elements are kept on the stack of Cartesian (sub-)trees. Each element of the stack is

a node of the resulting Cartesian tree together with its left subtree, but with the right subtree missing. Trees that are higher on the stack have smaller values in their roots. For each root of a tree on the stack, its right sub tree will be constructed by merging all trees that are above on the stack and possibly some elements of the array, that have not been scanned yet.

How should we update the stack, when another element of the array is scanned. If the scanned integer is smaller than the root of the topmost tree on the stack, we can simply push it onto the stack as a single-node tree. If it is larger than the roots of some trees on top of the stack, we have to pop all these trees and merge them to form the left subtree of the scanned element. The following figure shows the situation on the stack just before and after scanning 12 in the example sequence.





What changes if we do not need the Cartesian tree itself, but just its height? Instead of a tree it suffices to store its height and its root. Below is a solution in Python. Heights of trees on the stack are kept in `h_stack` and roots of trees are kept in `v_stack`. The code can be simplified by adding two guards: Initially, if the stack is not empty, but contains ∞ instead, it will never be popped and we don't have to check if the stack is empty. By adding "$\infty - 1$"

at the end of the given sequence, we have a guarantee that in the last step all the trees on the stack (except the other guard) are merged into one Cartesian tree.

1: Model solution — $O(N^4)$

```
INFTY = 2000000000

def sequence(A):
    N = len(A)
    if N == 0: return 0
    A = A + [INFTY - 1]

    v_stack = [INFTY] * (N + 2)
    h_stack = [0] * (N + 2)
    sp = 1
    for X in A:
        if X > v_stack[sp]:
            # Elements smaller than X on the stack will form its
                left sub-tree
            while X > v_stack[sp - 1]:
                # merge two elements on top of the stack
                h_stack[sp - 1] = max(h_stack[sp - 1], h_stack[
                    sp] + 1)
                sp = sp-1
            # Element on top of the stack represents left sub-
                tree of x
            v_stack[sp] = X
            h_stack[sp] = h_stack[sp] + 1
        else:
            # X is a leaf
            sp = sp + 1
            v_stack[sp] = X
            h_stack[sp] = 0
    assert sp == 2
    return h_stack[sp]
```