Problem 1: Rumour spreading

Rumour spreading is one of the basic mechanisms for information dissemination in networks. For simplicity, assume that rumour can be spread by either male or female person. Female person can spread the rumour by disseminating the information to other two female people and one male person, where as when male person can spread the rumour by disseminating the information to one female person and one male person. Assume that the spreading of rumour is clocked, that is, each spreading happens at the beginning of a microsecond (our unit of time in this exercise). At t = 0, we start the rumour spreading just by staring with a single female person.

Let H_n denote the number of female person and L_n the number of male person at time t = n. We have:

$H_0 = 1$,

$L_0 = 0$,

$H_n = 2H_{n-1} + L_{n-1}$ for $n > 1$,

$L_n = H_{n-1} + L_{n-1}$ for $n > 1$

Use a floating-point representation (preferably double) for H_n and L_n. These counts are integers but grow so rapidly that except for small values of n, **int** variables encounter overflow for storing them. Given an integer $n \geq 0$, we want to compute H_n and L_n. Three ways of doing this are explained below.

Method 0: Individual computation of H_n and L_n

Write a function *hirec(n)* that returns H_n (and nothing else). Likewise, write a function *lorec(n)* to return L_n (and nothing else). The only parameter that may be passed to each of these functions is n.

Method 1: Simultaneous computation of H_n and L_n

Write a function *hilorec(n)* to return the pair (H_n,L_n) (in a structure or a two-element array). Again, the only parameter allowed to be passed to the function is n.

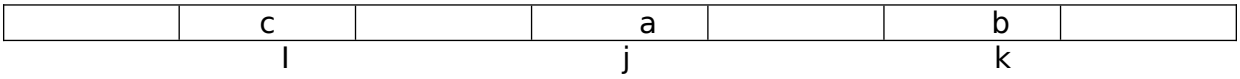
Check for algolicious sequences:

The international chemistry Prof Heisenberg is in the making of a complex chemist compound. He knows he has to add **n** elements to make this complex chemist compound. These complex chemist compound are numbered 1,2,3,.....n. The complex chemist compound becomes possible if the sequence of adding the elements is proper. Any improper sequence ruins the complex chemist compound altogether, so Prof Heisenberg asks for your help to ensure that his sequence is proper. A proper sequence is defined as follows.

Let **S** be the array storing the sequence. Assume that each compound (an integer in the range [1, n]) appears once and only once in S. In particular, the size of S is n. Take any three different compound *a, b, c* with

1 ≤ a < b < c ≤ n

An improper placement of a,b, c in S is described by a situation like this:



Here*, c, a, b* need not appear consecutively, but their relative position in S of this form is improper. The sequence S is called *algolicious* (and the complex chemist compound) if S does not contain any improper placement of *a,b, c* for all possible choices of the compounds. For example, for n = 10, the sequence *3,4,5,2,6,7,9,8,1,10* is *algolicious*, whereas the sequence *3,4,5,1,6,7,9,8,2,10* is not (see the improper placement of *1,2,5*). Notice that the only improper order is largest-smallest-intermediate. All other combinations (like smallest largest-intermediate) are proper.

Method 0: Brute force – O(n^4)

A direct translation of the above definition is to choose each combination a,b, c of compounds, find their positions j, k, i (as in the picture), check whether i < j < k, and if so, discard *S* as unalgolicious. If all checks discover proper placements, declare *S* as algolicious. There are nC3 ≈ (n^3)/6 compound combinations. Since *S* is unsorted, you have to make linear search in *S* to locate the positions *i, j, k.* Therefore the running time is O(n^4).

Method 1: Brute force, but better – O(n^3)

Instead of running the search on a,b, c, let us run the search on *i, j, k.* For each choice of the indices satisfying 0 ≤ i < j < k ≤ n − 1, take c = S[i] , a = S[j] , and b = S[k]. Then, check whether a < b < c. Since every compound appears at some position in *S*, this exhausts all compound combinations. Still, a factor of *n* is gained.

Method 2: Brute force, more refinement – $O(n^2)$

For each i, set c = S[i]. Look at the subsequence of S[i+1 , n−1] consisting of numbers < c . S is algolicious if and only if for each i, this subsequence is (strictly) decreasing. Therefore for each i, you need to make a single pass through the rest of the sequence, and another factor of n is gained.

Method 3: Good bye brute force – O(n)

Welcome to the world of algorithms. Design an O(n)-time algorithm yourself.