Sélection internationale Session 2015
École Normale Supérieure Paris
Épreuve de culture scientifique - **Informatique** Duration : 2 hours

For candidates who chose **computer science** as **secondary specialisation**

If you cannot answer a question, you may use it as hypothesis to later questions.
Calculators are not allowed.

Exercice 1.
A basic arithmetic expression is composed of characters from the set `1`, `+` and parentheses. Almost every integer can be represented by more than one basic arithmetic expression. For example, all of the following basic arithmetic expressions represent the integer 14:

```
1+1+1+1+1+1+1+1+1+1+1+1+1+1
((1+1) (1+1+1+1+1))+((1+1) (1+1))
(1+1) (1+1+1+1+1+1+1)
(1+1) (((1+1+1) (1+1))+1)
```

Describe an algorithm to compute, given an integer n as input, the minimum number of 1's in a basic arithmetic expression whose value is n. The number of parentheses doesn't matter, just the number of 1's. For example, when $n = 14$, your algorithm should return 8, for the final expression above.

Your algorithm should run in time polynomial in n.

Exercice 2.
For this exercise, all arrays are arrays of integers and a *sorted* array is an array where elements are in non-decreasing order.

1. Describe an $O(n)$ algorithm which

   - takes as input a size n array A and an integer i where $A[1..i]$ is already sorted and $A[i+1..n]$ is already sorted, and
   - modifies A so the entire array becomes sorted.

   Your algorithm must not create any new arrays. (In other words, your algorithm should be *in-place*.)

Merge sort is a sorting algorithm which starts from a completely unsorted array and recursively applies your algorithm from 1.

In this question, we will look at a variant of merge sort where before merging, instead of having two sorted subarrays, the array may be split into many sorted subarrays. A sequence represents the length of the sorted subarrays.

We wish to update this sequence to reflect what happens when a new subarray is added and merge sort is applied to some pairs of subarrays. For the remainder of this question, we will only be concerned with this sequence of lengths (instead of the details of sorting the array).

Consider the following update function on a sequence L of positive integers (representing length of subarrays).

```
function UPDATE(L, value)
  ADD(L, value)
  repeat indefinitely
    n ← LENGTH(L)
    if n > 2 and L[n − 2] ≤ L[n − 1] + L[n] and L[n − 2] < L[n] then
      MERGE(L, n − 2, n − 1)
    else if n > 2 and L[n − 2] ≤ L[n − 1] + L[n] and L[n − 2] ≥ L[n] then
      MERGE(L, n − 1, n)
    else if n ≥ 2 and L[n − 1] ≤ L[n] then
      MERGE(L, n − 1, n)
    else
      return
```

The above functions uses the following subroutines.

LENGTH(L) returns the number of elements currently in L.

ADD(L, value) adds value to the end of the sequence L.

MERGE($L, i, i+1$) replaces elements at index i and $i+1$ by a single element (at index i) that is their sum. For example MERGE($L, 3, 4$) when L is

$$0, 2, 4, 6, 8$$

gives

$$0, 2, 10, 8$$

since $4 + 6 = 10$.

$L[i]$ returns the ith element in the sequence. $L[n]$ is the last element in the sequence and $L[1]$ is the first element.

2. Describe a data structure that supports

   - the LENGTH and ADD operations in $O(1)$ and
   - the MERGE operation where we merge the last two elements or the two elements before the last element in $O(1)$. I.e., the only indices allowed as inputs to MERGE($L, i, i+1$) are $i =$LENGTH$(L) - 1$ and $i =$LENGTH$(L) - 2$.

3. Show that if we start with an empty sequence, using your data structure, the running time for applying UPDATE k times is $O(k)$.

4. When the update function returns, the following is true (if L has enough elements left):

   - $L[n-1] > L[n]$
   - $L[n-2] > L[n-1] + L[n]$

   Show that if both

   - $L[i-1] > L[i]$
   - $L[i-2] > L[i-1] + L[i]$

   for all i then the first element of L has value exponential in the length of the sequence L.

5. Show that starting from an empty sequence, there is a set of calls to UPDATE (and no other function) so that it is not true that

   - $L[i-1] > L[i]$
   - $L[i-2] > L[i-1] + L[i]$

   for all i at the end of these calls.

6. Show that if UPDATE is changed to MODIFIEDUPDATE so that

   - $L[i-1] > L[i]$
   - $L[i-2] > L[i-1] + L[i]$

   is true for both $i = n$ and $i = n-1$ at the end of every MODIFIEDUPDATE operation then starting from an empty sequence, after any number of calls to MODIFIEDUPDATE (and no other functions),

   - $L[i-1] > L[i]$
   - $L[i-2] > L[i-1] + L[i]$

   is true for all i.