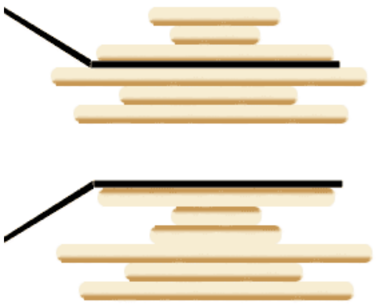
Sélection internationale Session 2017
École normale supérieure Paris
Épreuve de culture scientifique - **Informatique** Duration : 2 hours

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.

Exercise 1. Suppose you are given a stack of n pancakes of different sizes. You want to sort the pancakes so that smaller pancakes are on top of larger pancakes. The only operation you can perform is a flip – insert a spatula under the top k pancakes, for some integer k between 1 and n, and flip them all over.



1. Give a lower bound on the number of flips an algorithm has to perform to sort an arbitrary stack of n pancakes.
2. Describe an efficient algorithm to sort an arbitrary stack of n pancakes. Exactly how many flips does your algorithm perform in the worst case?
3. Now suppose one side of each pancake is burned. Exactly how many flips do you need to sort the pancakes and have the burned side of every pancake on the bottom?

Exercise 2.

1. Suppose you are given two sorted arrays $A[1 \ldots m]$ and $B[1 \ldots n]$ and an integer k. Describe an algorithm to find the kth smallest element in the union of A and B in $O(\log(m+n))$ time.

   For example, given the input

   $$A[1 \ldots 8] = [0, 1, 6, 9, 12, 13, 18, 20],\ B[1 \ldots 5] = [2, 5, 8, 17, 19] \text{ and } k = 6$$

   your algorithm should return 8. You can assume that the arrays contain no duplicates.
2. Suppose you are given two sorted arrays $A[1 \ldots n]$ and $B[1 \ldots n]$. Give an $O(\log^2 n)$ algorithm to find the n-th smallest of the $2n$ elements.
3. Suppose you have k sorted arrays, each with n elements, and you want to combine them into a single sorted array of kn elements.
   (a) Consider the following strategy: Using the merge procedure (from the Mergesort algorithm), merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm, in terms of k and n?

(b) Give a more efficient solution to this problem, using divide-and-conquer. What is its time complexity in terms of k and n?

4. Let $M[1 \dots n][1 \dots n]$ be an $n \times n$ matrix in which every row and every column is sorted. Such an array is called *totally monotone*. No two elements of M are equal.

   (a) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, compute the number of elements of M smaller than $M[i][j]$ and larger than $M[i'][j']$.

   (b) Describe and analyze an algorithm to solve the following problem in $O(n)$ time: Given indices i, j, i', j' as input, return an element of M chosen uniformly at random from the elements smaller than $M[i][j]$ and larger than $M[i'][j']$. Assume the requested range is always non-empty.

   (c) Describe and analyze a randomized algorithm to compute the median element of M in $O(n \log n)$ expected time.

5. Let $A[1 \dots n]$ be an array such that the first $n\sqrt{n}$ elements are already sorted (though we know nothing about the remaining elements. Write an algorithm that will sort A in substantially better than $O(n \log n)$ steps.

Exercise 3. Let $k, n \in \mathbb{N}$. The (k, n)-egg problem is as follows: There is a building with n floors. You are told that there is a floor f such that if an egg is dropped off the fth floor then it will break, but if it is dropped off the $(f-1)$st floor it will not. (It is possible that $f = 1$ so the egg always breaks, or $f = n + 1$ in which case the egg never breaks.)

If an egg breaks when dropped from some floor then it also breaks if dropped from a higher floor. If an egg does not break when dropped from some floor then it also does not break if dropped from a lower floor.

Your goal is, given k eggs, to find f. The only operation you can perform is to drop an egg off some floor and see what happens. If an egg breaks then it cannot be reused. You would like to drop eggs as few times as possible. Let $E(k, n)$ be the minimal number of egg-droppings that will always suffice.

1. Show that $E(1, n) = n$.

2. Show that $E(k, n) = \Theta(n^{1/k})$

3. Find a recurrence for $E(k, n)$. Write a dynamic program to find $E(k, n)$. How fast does it run?

4. Write a dynamic program that can be used to actually yield the optimal strategy for finding the floor f. How fast does it run?