Sélection internationale Session 2013
École Normale Supérieure Paris
Épreuve de culture scientifique - **Informatique** Durée : 3 heures

Pour les candidats ayant choisi **l'informatique** comme **spécialité principale**

Si vous ne parvenez pas à répondre à une question, vous pouvez cependant l'utiliser comme hypothèse pour les questions suivantes.

L'usage de la calculatrice n'est pas autorisé.

Exercice 1.

Soit L une liste de n entiers positifs et différents, que nous comparons en utilisant l'ordre $\leq$ usuel dans les entiers. L'élément le plus grand dans L sera appelé *max*, et son prédécesseur dans l'ordre sera appelé *sec-max*. Autrement dit, *max* est l'élément maximum de la liste L, et *sec-max* est le maximum de la liste L privée de *max*.

1. En utilisant le langage de programmation de votre choix, programmez un algorithme pour calculer *max* et *sec-max* sans trier L. Quelle est la complexité (nombre de comparaisons) de votre algorithme ?

2. Montrez que tout algorithme qui calcule *max* en comparant les éléments de L doit faire au moins $n-1$ comparaisons, et en déduire que l'on peut calculer *max* et *sec-max* avec $2n-3$ comparaisons.

3. La borne calculée dans la question précédante peut être améliorée : au lieu de calculer les éléments *max* et *sec-max* de manière indépendante, on peut utiliser un algorithme du style " tournement ". Donnez un algorithme pour résoudre ce problème en effectuant un total de $n + \lceil log_2 n \rceil - 2$ comparaisons dans le pire des cas.

4. Montrez que tout algorithme qui calcule *max* et *sec-max* en comparant les éléments de L doit faire au moins
$$n + \lceil log_2 n \rceil - 2$$
comparaisons dans le pire des cas.

Exercice 2. Le problème SAT est un problème de logique propositionnelle. Un prédicat est défini sur un ensemble de variables logiques, à l'aide des trois opérations élémentaires que sont la négation NON ($\neg x$ que nous noterons aussi $\overline{x}$), la conjonction ET ($x \wedge y$) et la disjonction OU ($x \vee y$). Un littéral est un prédicat formé d'une seule variable (x) ou de sa négation ($\overline{x}$).

Une clause est un prédicat particulier, formée uniquement de la disjonction de littéraux, par exemple $C = x \vee \overline{y} \vee z$. Une formule est sous forme normale conjonctive si elle s'écrit comme la conjonction de clauses. Le problème SAT consiste à décider si une formule en forme normale conjonctive est satisfaisable, c'est-à-dire si il existe une assignation des valeurs de vérité vrai ou faux aux variables telle que toutes les clauses sont vraies.

Par exemple la formule $(x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z})$ est satisfaite avec l'assignation $\tau(x) = \text{vrai}$, $\tau(y) = \text{faux}$ et $\tau(z) = \text{faux}$.

Nous nous intéressons au problème 3-SAT, qui consiste à déterminer si une formule logique en forme normale conjonctive avec au plus trois littéraux par clause est satisfaisable. La complexité des algorithmes sera exprimée en fonction de n le nombre de variables apparaissant dans la formule.

1. Démontrer que si $\psi \wedge x$ est insatisfaisable, alors toute assignation des variables satisfaisant ψ contiendrait $\tau(x) = \text{faux}$.
2. Démontrer si x est un littéral, alors les formules ψ et $(\psi \wedge x) \vee (\psi \wedge \overline{x})$ sont équivalentes (*i.e.* ont les mêmes modèles). En déduire un algorithme récursif pour 3-SAT de complexité $O(P(n) \cdot 2^n)$ où P est un polynôme (dont la forme explicite n'est pas demandée).

Nous allons maintenant chercher à obtenir un algorithme asymptotiquement plus rapide. Si ψ n'est pas vide, alors elle s'écrit $\psi = (x \vee y \vee z) \wedge \psi'$, où x, y et z sont des littéraux.

3. Montrer que ψ est équivalente à

$$(x \wedge \psi') \vee (y \wedge \psi') \vee (z \wedge \psi').$$

   En déduire un algorithme récursif de complexité $O(1.8392^n)$.
   Indication : On pourra exprimer la complexité de l'algorithme obtenu sous forme d'une suite récurrente linéaire et utiliser sans démonstration que $\rho_1 = 1.839286755\ldots$ est l'unique racine réelle de l'équation $X^3 - X^2 - X - 1 = 0$.
4. Un littéral x est dit pur si $\overline{x}$ n'apparaît pas dans ψ. Montrer que si ψ est satisfaisable, alors il existe une assignation des variables dans laquelle tous les littéraux purs sont vrais. Cela permet de se ramener au cas où aucun littéral n'est pur. Cela signifie que si ψ est non-triviale, on peut l'écrire :

$$(x \vee y \vee z) \wedge (\overline{x} \vee u \vee v) \wedge \psi'$$

   où u et v sont des littéraux arbitraires (qui peuvent très bien être y, $\overline{y}$, z et $\overline{z}$). En déduire un algorithme récursif de complexité $O(1.7692^n)$
   Indication : On pourra utiliser sans démonstration que $\rho_2 = 1.769292354\ldots$ est l'unique racine réelle de l'équation $X^3 - 2X - 2 = 0$.
5. ($\star$) Justifier que dans chaque appel récursif de l'algorithme précédent, ou bien on a fait apparaître un littéral pur (qu'on peut donc éliminer), ou bien on fait apparaître une clause à deux littéraux. En déduire un algorithme de complexité $O(1.6180^n)$.

Sélection internationale Session 2013
École Normale Supérieure Paris
Épreuve de culture scientifique - **Informatique** Durée : 2 heures

Pour les candidats ayant choisi **l'informatique** comme **spécialité secondaire**

Si vous ne parvenez pas à répondre à une question, vous pouvez cependant l'utiliser comme hypothèse pour les questions suivantes.

L'usage de la calculatrice n'est pas autorisé.

Exercice 1.
L'algorithme d'Euclide calcule le plus grand commun diviseur (pgcd) de deux entiers naturels. Dans cet exercice nous considérons l'algorithme récursif suivant, appelé algorithme d'Euclide binaire.

Entrée : $a, b \in N$ tels que $a > 0$ et $b > 0$.

Sortie : $pgcd(a, b) \in N$.

```
fonction pgcd(a,b) =
   Si a = b alors a sinon
    Si a et b sont pairs alors 2*pgcd(a/2,b/2);
    Si seulement un des deux nombres, disons a, est pair,
           alors pgcd(a/2,b);
    Si a et b sont impairs et a>b (le cas b>a est similaire)
           alors pgcd((a-b)/2,b).
```

1. Trouver le pgcd de 34 et 21, et de 136 et 51, en utilisant cet algorithme.

2. Écrire une implémentation du pseudo-code de l'algorithme d'Euclide binaire dans le langage de programmation de votre choix.

3. Prouver que l'algorithme termine pour toute entrée $a, b \in N$ telle que $a > 0$ et $b > 0$.

4. La profondeur récursive d'un algorithme est le nombre maximal d'appels récursifs réalisés pour une entrée donnée. Trouver une borne supérieure à la profondeur récursive de l'algorithme d'Euclide binaire.

5. Donner une famille infinie de couples (a, b) pour lesquels la borne supérieure sur la profondeur récursive de la question précédente est atteinte.

6. Prouver que l'algorithme est correct, c'est-à-dire, que quelques soient a et $b \in N$, $pgcd(a, b)$ est leur plus grand diviseur commun.

7. Modifier l'algorithme pour calculer aussi $s, t \in Z$ tels que

$$pgcd(a, b) = sa + tb.$$

Sélection internationale Session 2013
École Normale Supérieure Paris
Scientific Test - **Informatics** Duration : 3 hours

For candidates who chose **Informatics** as **principal discipline**

If you cannot answer a question you can still use it as an assumption in the next questions.

The use of calculators is not allowed.

Exercise 1.

Assume L is a list of n different natural numbers, which we will compare using the standard ordering on integers, written $\leq$. The greatest element in L will be called *max*, and its predecessor in the order will be called *sec-max*. In other words, *max* is the maximum element in L, and *sec-max* is the maximum in the list obtained by erasing *max* from L.

1. Using your favourite programming language, write an algorithm to compute *max* and *sec-max* without sorting L. What is the complexity (i.e., number of comparisons) of your algorithm ?

2. Show that any algorithm that computes *max* by comparing the elements in L must make at least $n-1$ comparisons. Deduce that it is possible to compute *max* and *sec-max* with $2n-3$ comparisons.

3. The bound computed in the previous questions can be improved. Instead of computing *max* and *sec-max* independently, we can use a "championship-style" algorithm. Write an algorithm to solve this problem with $n + \lceil log_2 n \rceil - 2$ comparisons in the worst case.

4. Show that any algorithm that computes *max* and *sec-max* by comparing the elements of L must make at least
$$n + \lceil log_2 n \rceil - 2$$
comparisons in the worst case.

Exercise 2. The SAT problem is a problem in propositional logic. A predicate is defined on a set of logical variables using three basic operations : the negation NOT ($\neg x$ that we will also denote $\overline{x}$), the conjunction AND ($x \wedge y$) and the disjunction OR ($x \vee y$). A literal is a predicate formed with only one variable (x) or its negation ($\overline{x}$).

A clause is a specific predicate formed only as a disjunction of literals, for instance $C = x \vee \overline{y} \vee z$. A formula is a conjunctive normal form if it's written as a conjunction of clauses. The SAT problem consists in deciding whether a formula in conjunctive normal form is satisfiable, namely if there exists an assignation of the truth value true or false to the variables such that all clauses are true.

For instance, the formula $(x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y}) \wedge (\overline{x} \vee \overline{z})$ is satisfied with the assignation $\tau(x) = \mathsf{true}$, $\tau(y) = \mathsf{false}$ and $\tau(z) = \mathsf{false}$.

We study the 3-SAT problem which consists in deciding whether a formula in conjunctive normal form with at most three literals per clause is satisfiable. The complexity of the algorithms will be given as a function of the number n of variables in the formula.

1. Prove that if $\psi \wedge x$ is not satisfiable then all assignation of variables satisfying ψ verifies $\tau(x) = \mathsf{faux}$.
2. Prove that if x is a literal then the formulas ψ and $(\psi \wedge x) \vee (\psi \wedge \overline{x})$ are equivalent (*i.e.* have the same models). Deduce a recursive algorithm for 3-SAT of complexity $O(P(n) \cdot 2^n)$ where P is a polynomial (whose explicit form is not asked).

We are now looking for an algorithm asymptotically faster. If ψ is not empty, it can be written as $\psi = (x \vee y \vee z) \wedge \psi'$, where x, y and z are literals.

3. Show that ψ is equivalent to

$$(x \wedge \psi') \vee (y \wedge \psi') \vee (z \wedge \psi').$$

   Deduce a recursive algorithm of complexity $O(1.8392^n)$.
   Hint : One may express the complexity of the algorithm as a linear recurrent sequence and use without proof that $\rho_1 = 1.839286755\ldots$ is the only real root of the equation $X^3 - X^2 - X - 1 = 0$.
4. A literal x is deemed pure if $\overline{x}$ does not appear in ψ. Show that if ψ is satisfiable then there exist an assignation of variables in which all pure literals are true. In particular, one can consider only formulas where no literal is pure and if ψ is not empty, it can be written as :

$$(x \vee y \vee z) \wedge (\overline{x} \vee u \vee v) \wedge \psi'$$

   where u and v are arbitrary literals (that can be y, $\overline{y}$, z or $\overline{z}$). Deduce a recursive algorithm of complexity $O(1.7692^n)$
   Hint : One may use without proof that $\rho_2 = 1.769292354\ldots$ is the only real root of the equation $X^3 - 2X - 2 = 0$.
5. ($\star$) Prove that in each recursive call of the previous algorithm, we create a pure literal (that we can eliminate) or a clause with two literals. Deduce an algorithm with complexity $O(1.6180^n)$.

Sélection internationale Session 2013
École Normale Supérieure Paris
Scientific Test - **Informatics** Duration : 2 hours

For candidates who chose **Informatics** as **secondary discipline**

If you cannot answer a question you can still use it as an assumption in the next questions.

The use of calculators is not allowed.

Exercise 1.

Euclide's algorithm computes the greatest common divisor (gcd) of two natural numbers. In this exercise, we will consider the following recursive algorithm, called "binary gcd algorithm".

Input : $a, b \in N$ such that $a > 0$ and $b > 0$.

Output : $gcd(a, b) \in N$.

```
function gcd(a,b) =
   If a = b then a else
    If a and b are even then 2*gcd(a/2,b/2);
    If only one of the two numbers, say a, is even,
           then gcd(a/2,b);
    If a and b are odd and a>b (the case b>a is similar)
           then gcd((a-b)/2,b).
```

1. Find the gcd of 34 and 21, and the gcd of 136 abd 51, using this algorithm.

2. Implement, in your favourite programming language, the algorithm given above in pseudo-code.

3. Prove that the algorithm terminates for all inputs $a, b \in N$ such that $a > 0$ and $b > 0$.

4. The recursive depth of an algorithm is the maximal number of recursive calls performed for an input. Give an upper bound for the recursive depth of the binary gcd algorithm.

5. Give an infinite family of pairs (a, b) for which the upper bound mentioned in the previous question is reached.

6. Show that the algorithm is correct, that is, for any a and $b \in N$, $gcd(a, b)$ is their greatest common divisor.

7. Modify the algorithm in order to compute also integers s, t such that

$$gcd(a, b) = sa + tb.$$