Sélection internationale Session 2015
École Normale Supérieure Paris
ÃÉpreuve 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.

Calculatrices interdites.

Exercice 1. Supposons que nous avons en notre possession n pièces de monnaie dont certaines sont contrefaites. Les pièces réelles ont toutes le même poids w_0 et les pièces contrefaites ont toutes le même poids $w_1 < w_0$. Nous disposons d'une balance de type Roberval très précise qui étant donnés deux ensembles arbitraires de pièces (parmi ces n pièces) nous permet de décider si les ensembles sont de même poids ou quel ensemble est le plus léger. Sans l'aide de cette balance, les pièces sont cependant indistinguables. Nous supposons qu'il y a au moins une pièce contrefaite dans l'ensemble des n pièces.

1. Proposer un algorithme qui permet de trouver (au moins) une pièce contrefaite lorsqu'il y a $n = 8$ pièces et effectue le nombre minimum de pesées (on ne vous demande pas de demander que l'algorithme minimise le nombre de pesées). Combien votre algorithme effectue-t-il de pesées au pire ?

2. Proposer un algorithme qui permet de trouver (au moins) une pièce contrefaite en $\log_2(n)+O(1)$ pesées.

3. Proposer un algorithme en $\log_2(n)+O(1)$ pesées. qui permet de diviser les pièces en trois ensembles S_1, S_2 et S_3 tels que

   - S_1 et S_2 contiennent autant de pièces réelles ;
   - S_1 et S_2 contiennent autant de pièces contrefaites ;
   - S_3 contient au plus deux pièces.

4. En déduire un algorithme en $\log_2(n) + O(1)$ pesées qui retourne la parité du nombre de pièces contrefaites parmi les n pièces.

5. Proposer un algorithme en $\log_2^2(n) + O(\log_2(n))$ pesées qui retourne le nombre de pièces contrefaites parmi les n pièces.

6. Montrer (par un argument de combinatoire par exemple) qu'un algorithme qui retourne le nombre de pièces contrefaites parmi les n pièces doit effectuer au moins $\log_3(n)$ pesées.

Exercice 2. Supposons que nous avons en notre possession n écrous et n vis de tailles différentes. Chaque vis correspond à un écrou et un seul (et réciproquement). Les écrous et les vis sont presque de la même taille et il est impossible de dire si un écrou est plus grand qu'un autre ou si une vis est plus grand qu'une autre. Cependant, si on essaie de faire correspondre un écrou à une vis, la vis sera soit trop grande, soit trop petite, soit de la bonne taille exactement. On peut donc faire des comparaisons écrou-vis, et chaque comparaison donne un résultat parmi les trois résultats possibles.

1. Proposer un algorithme qui, étant donné un écrou, retrouve la vis correspondante. Donner sa complexité en nombre de comparaisons écrou-vis (en fonction de n).

2. Proposer un algorithme probabiliste inspiré du tri rapide pour associer correctement chaque écrou à sa vis.

3. Soit $X_{i,j}$ la variable aléatoire qui vaut 1 si à un moment lors de l'exécution le écrou i est comparé à la vis j, et $X_{i,j} = 0$ sinon. Exprimer en fonction des $X_{i,j}$ le nombre $T(n)$ de comparaisons écrou-vis effectuées par l'algorithme.

4. Donner une formule pour $\mathbb{E}(X_{i,j})$ l'espérance de $X_{i,j}$.

5. Montrer que $\mathbb{E}(T(n)) = O(n \log n)$ (où $\mathbb{E}(T(n))$ est l'espérance de $T(n)$).

Exercice 3. Vous avez n devoirs à faire mais avez pris du retard. Le i-ième devoir vous réclamera h_i heures de travail et doit être terminé avant le temps t_i, sinon vous sera pénalisé : si vous le terminez au temps t'_i, votre pénalité sera 0 pour $t'_i \leq t_i$ et $t'_i - t_i$ pour $t'_i > t_i$. Le but est de déterminer pour chaque i l'heure s_i de démarrage du devoir i, de façon à minimiser la pénalité totale que vous devrez payer. Bien évidemment, vous ne pouvez travailler que sur un devoir à la fois.

1. Donner un algorithme polynomial qui, étant donné $h_1, h_2, \ldots, h_n$, calcule $s_1, s_2, \ldots, s_n$.

2. Démontrer que votre algorithme est correct, c'est-à-dire qu'il minimise la pénalité que vous devrez payer.

3. Quel est la complexité de l'algorithme en temps ?

4. Finalement, vous n'allez pas y arriver et devez laisser tomber certains des devoirs. Choisir de ne pas rendre le devoir i vous fait encourir une pénalité de p_i. Donner un algorithme efficace qui, étant donné $(h_1, p_1), (h_2, p_2), \ldots, (h_n, p_n)$, calcule $(b_1, s_1), (b_2, s_2), \ldots, (b_n, s_n)$, où b_i est une variable booléenne, vraie si vous terminez le devoir i et faux sinon, et s_i n'est définie que si b_i est vrai et est dans ce cas égal à l'heure de démarrage de la tâche i.

Exercice 4. Le but de l'exercice est de concevoir une structure de données pour stocker des mesures de températures. Une mesure est de la forme $(t; d)$ où t, un nombre réel qui peut être négatif, est une température en Celsius, et où d, un entier naturel, est le nombre de jours écoulés entre le 31 décembre 1999 et le jour de la mesure (d est toujours positif). Il peut y avoir plusieurs mesures un même jour (qui peuvent mesurer la même température, ou non - S est donc un multi-ensemble puisqu'un même (t, d) peut être présent plusieurs fois), et il peut également n'y en avoir aucune.

La structure de données pour S doit pouvoir gérer les opérations suivantes :

- insérer$(t; d)$ qui ajoute à S la mesure $(t; d)$; et
- moyenne$(d_1; d_2)$ où $d_1 \leq d_2$ sont deux dates. Cette requête donne en résultat la moyenne uniforme des températures de tous les (t, d) de S tels que $d_1 \leq d \leq d_2$, c'est-à-dire la somme de ces tempértures divisée par le nombre de mesures.

1. Montrer brièvement comment, avec une structure de données naïve, on peut obtenir une complexité de $O(1)$ pour chaque insertion et $O(n)$ pour chaque moyenne, où n est la taille de S.

2. Montrer comment, avec une autre méthode, on peut obtenir une complexité de $O(D)$ pour chaque insertion et $O(1)$ pour chaque requête de moyenne, où D est le nombre total de jours écoulés entre le 31 décembre 1999 et aujourd'hui.

3. Proposer une structure de données avec laquelle chaque insertion a complexité $O(\log n)$ et chaque requête de moyenne a complexité $O(\log n)$.