Partiel – Lundi 12 novembre 2012

Durée : 2h. Notes de cours et documents non autorisés. Le sujet comprend 3 exercices indépendants.

La notation tiendra compte de la qualité de la rédaction et du soin apporté à la copie. Pour maximiser votre note, justifiez donc correctement toutes vos réponses et écrivez lisiblement. Ce n'est pas un concours de vitesse !
Si vous bloquez sur une question d'un exercice, vous pouvez admettre le résultat et passer à la suite.

Exercice 1.
Dans cet exercice, on s'intéresse au calcul (simultané) du maximum et du minimum de n entiers. On mesure la **complexité dans le pire des cas et en nombre de comparaisons** des algorithmes.

1. Donner un algorithme naïf et sa complexité.

Une idée pour améliorer l'algorithme est de regrouper *par paires* les éléments à comparer, de manière à diminuer ensuite le nombre de comparaisons à effectuer.

2. Décrire un algorithme fonctionnant selon ce principe et analyser sa complexité.

Nous allons étudier l'optimalité d'un tel algorithme en fournissant une borne inférieure sur le nombre de comparaisons à effectuer. Nous utiliserons la méthode de l'*adversaire*.
Soit A un algorithme qui trouve le maximum et le minimum. Pour une donnée fixée, au cours du déroulement de l'algorithme, on appelle *novice* (N) un élément qui n'a jamais subi de comparaisons, *gagnant* (G) un élément qui a été comparé au moins une fois et a toujours été supérieur aux éléments auxquels il a été comparé, *perdant* (P) un élément qui a été comparé au moins une fois et a toujours été inférieur aux éléments auxquels il a été comparé, et *moyens* (M) les autres. Le nombre de ces éléments est représenté par un quadruplet d'entiers (i, j, k, l) qui vérifie bien sûr $i + j + k + l = n$.

3. Donner la valeur de ce quadruplet au début et à la fin de l'algorithme. Exhiber une stratégie pour l'adversaire, de sorte à maximiser la durée de l'exécution de l'algorithme. En déduire une borne inférieure sur le nombre de tests à effectuer.

Exercice 2.
On considère le problème consistant à rendre n centimes en monnaie, en utilisant le moins de pièces possible. On supposera que chaque pièce a une valeur entière.

1. Décrire un algorithme glouton permettant de rendre la monnaie en utilisant des pièces de un, cinq, dix, vingt et cinquante centimes. Démontrer que votre algorithme aboutit à une solution optimale.
2. On suppose que les pièces disponibles pour rendre la monnaie ont des valeurs qui sont des puissances de p, i.e. $p^0, p^1, \ldots, p^k$, où $p > 1$ et $k \geq 1$ sont deux entiers. Montrer que l'algorithme glouton aboutit toujours à une solution optimale.
3. Donner un ensemble de valeurs de pièces pour lequel l'algorithme glouton ne donnera pas de solution optimale. Votre ensemble doit permettre de trouver une solution pour chaque valeur de n.
4. Donner un algorithme à temps $\mathcal{O}(nk)$ qui rende la monnaie pour n'importe quel ensemble de k valeurs différentes de pièce, en supposant que l'une des pièces est 1 centime.

Exercice 3.
On considère un ensemble $E = \{p_1, \ldots, p_n\}$ de points du plan, donné sous forme d'une liste de couples $p_i = (x_i, y_i)$. On cherche les deux points p_i et p_j qui minimisent la distance

$$d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

On suppose par simplicité que n est une puissance de 2 et que toutes les abscisses sont distinctes deux-à-deux, de même que les ordonnées. On suppose que le calcul de la distance entre deux points se fait en temps $\mathcal{O}(1)$.
On propose pour résoudre le problème une stratégie de type « Diviser pour Régner ».

(i) Couper E en deux sous-ensembles G et D selon un axe vertical Δ.

(ii) Chercher les deux points les plus proches dans G, et les deux plus proches dans D. Soit d la distance minimale obtenue.

(iii) Chercher s'il existe deux points $p_g \in G$ et $p_d \in D$ tels que $d(p_g, p_d) < d$.

(iv) Retourner les deux points les plus proches.

1. Montrer qu'à l'étape (iii), on peut se restreindre à un ensemble S_Δ de points de S contenus dans une bande verticale autour de l'axe Δ.

2. Montrer que tout carré du plan, parallèle aux axes, de côté d contient au plus 4 points de G (resp. de D).

3. En déduire qu'à l'étape (iii), il suffit de calculer pour chaque point de S_Δ sa distance à un nombre constant de points de S_Δ.

4. Écrire un algorithme utilisant la stratégie décrite ci-dessus et prouver sa correction. On s'assurera que la complexité de l'étape (iii) est $\mathcal{O}(n \log n)$.

5. Quelle est la complexité globale de votre algorithme ?

6. Pouvez-vous améliorer votre algorithme pour qu'il fonctionne en temps $\mathcal{O}(n \log n)$?