Algorithmique avancée
Série d'exercices n° 3
(Correction)

Exercice 1 :

Dans cet exercice, on va supposer que le coût en temps du tri d'un tableau par insertion est $c.n^2$ pour un tableau de taille n (c étant une constante). On effectue l'algorithme hybride suivant :

– Diviser un tableau non trié de taille n en k sous-tableaux de taille égale (n/k).
– Trier chaque sous-tableau en utilisant le tri par insertion.
– Fusionner les sous-tableaux obtenus.

1. Calculer le temps de la fusion en fonction de k et de n.

▷ *On a k paquets P_i, $1 \leq i \leq k$ triés. A chaque paquet, on associe un curseur sur le plus petit élément non encore placé dans le tableau résultat T_F (le tableau de la fusion). A chaque fois qu'on pioche dans P_i pour le placer dans le tableau fusion, on incrémente le curseur c_i. On a un curseur c pour le prochain emplacement libre dans le tableau fusion. A l'initialisation, on a $c = 1$ et $c_i = 1$. On cherche parmi les P_i à l'indice c_i l'élément le plus petit et on le place dans T_F : au pire des cas on effectue k étapes pour chercher cet élément. Comme on doit recommencer n fois, le coût est de $k.n$.*

2. En déduire le temps que mettra l'algorithme hybride donné ci-dessus.

▷ *Soit T_n le coût total. Nous avons :*

$$T_n = \textit{coûts des } k \textit{ tris} + \textit{coût de la fusion.}$$

Par hypothèse le coût d'un tri est $c.\frac{n^2}{k^2}$ et d'après la question précédente le coût de la fusion est $k.n$. Donc,

$$T_n = c\frac{n^2}{k} + kn.$$

On peut simplifier le calcul en remplaçant $T(n)$ par

$$t(n) = c\frac{n^2}{k} + ckn.$$

Dans ce cas, la dérivée de $t(n)$ par rapport à k vaut

$$c\left(n - \frac{n^2}{k^2}\right)$$

et cette dérivée s'annule pour $k = \sqrt{n}$. Il n'est pas difficile alors de remarquer que $k = \sqrt{n}$ minimise $f(n)$ et dans ce cas $f(n) = \Omega(n^{3/2})$ est une borne inférieure qui est atteinte en choisissant comme paramètre $k = \sqrt{n}$.

3. Conclure en comparant avec le tri fusion et le tri par insertion. (Veuillez tenir compte que k peut être une fonction de n.)

▷ *Le tri par insertion a un coût au meilleur cas (resp. en moyenne/ au pire cas) de $\Theta(n)$ ($\Theta(n^2)$ puis encore $\Theta(n^2)$). Le tri de l'exercice coûte quelque soit le cas $\Theta(n^{3/2})$.*

Par contre, le tri fusion effectue $O(n \log n)$ étapes quel que soit le cas (pire/meilleur ou en moyenne). Le tri de l'exercice est donc forcément moins efficace que le tri-fusion.

Exercice 2 :

Dans cet exercice, on note $T(n)$ le temps nécessaire pour un algorithme de tri pour trier un tableau de taille n. On suppose que T vérifie :

$$T(1) = 0 \text{ et } T(n) = 3T(n/3) + 2cn$$

où c est une constante positive.

1. Montrer que $T(3^m) = 2c.m.3^m$ $(m \neq 0)$.

▷ *On calcule*
$$T(3^m) = 3.T(3^{m-1}) + 2.c.3^m$$
puis on développe $T(3^{m-1})$
$$T(3^m) = 3\left(3.T(3^{m-2}) + 2.c.3^{m-1}\right) + 2.c.3^m$$
pour obtenir
$$T(3^m) = 3^2.T(3^{m-2}) + 2.2.c.3^m\ .$$
On peut réitérer le processus encore une fois en développant $T(3^{m-2})$ *:*
$$T(3^m) = 3^2.\left(3.T(3^{m-3}) + 2.c.3^{m-2}\right) + 2.2.c.3^m\ .$$
On obtient
$$T(3^m) = 3^3.T(3^{m-3}) + 2.3.c.3^m\ .$$
En réitérant m fois, il vient
$$T(3^m) = 3^m.T(3^{m-m}) + 2.m.c.3^m$$
d'où le résultat.

2. En déduire que $T(n) = \mathcal{O}(n \log n)$.

▷ *Si* n *est une puissance de 3, i.e.* $n = 3^k$ *alors* $T(n) = 2.k.c.n$ *et* $k = \log_3 n$ *donc on a le résultat. Si* n *n'est pas une puissance de 3 alors il existe* k *tel que* $3^{k-1} \leq n < 3^k$*, dans ce cas puisque* T *représente un coût* $T(n) \leq T(3^k)$ *: d'où le résultat.*

Exercice 3 :

On a un tableau T de n entiers dans $\{0, 1, 2\}$.

1. Donner un algorithme linéaire pour trier T.

▷ *Une suggestion : compter les* 0*, compter les* 1 *puis remplir le même tableau. Coût :* n *pour les* 0*,* n *pour les* 1 *et* n *pour le remplissage donc* $O(n)$*.*

2. Généraliser au cas des entiers dans $\{0, 1, \cdots, k\}$. Discutez explicitement des cas $k = \mathcal{O}(\log n)$ et $k = \Omega(\log n)$.

▷ *En appliquant la même idée qu'à la question précédent on obtient un algorithme en* $O(k.n)$*. Donc pour* $k = o(\log n)$*, on aura un tri en temps* $o(n \log n)$*. Attention : on a une hypothèse forte car les entiers sont dans* $[0,\ k]$*. Si* k *est bien plus grand que* $\log n$*, il vaut mieux utiliser un algorithme comme le tri fusion.*

Exercice 4 :

On a deux tableaux X et Y contenant chacun n entiers. Etant donné une valeur entière α, écrire un algorithme pour décider (répondre oui ou non) si α est la somme d'un élément de X et d'un élément de Y.

▷ *Une suggestion :*

Pour i *de* 1 *à* n
    Pour j *de* 1 *à* n
        Si $X[i] + Y[j] == \alpha$ *alors retourner(OUI) FinSi*
    FinPour
FinPour
retourner(NON)

Cet algorithme a un coût en $O(n^2)$ *:* $O(n)$ *pour les* X *et* $O(n)$ Y *pour chaque* $X[i]$*. On ne nous demande pas de faire bien mieux (d'ailleurs c'est difficile).*