Centrale Informatique MP 2012 — Corrigé

Ce corrigé est proposé par Guillaume Batog (Professeur agrégé) ; il a été relu par Gautier Marti (ENS Lyon) et Benjamin Monmege (ENS Cachan).

Le problème porte sur l'algorithme de tri rapide d'un tableau. Il est composé de quatre parties.

- La première partie, constituée uniquement de questions de cours, débute par la programmation d'un algorithme de tri simple, au choix, après évaluation de sa complexité. Puis les mêmes questions sont posées pour l'algorithme de tri rapide. Un petit effort de rédaction est demandé pour décrire les algorithmes.
- La deuxième partie étudie la complexité du tri rapide dans deux cas extrêmes : le pire, où le tableau est déjà trié, et le meilleur, où le pivot est l'élément médian du sous-tableau traité à chaque étape. Bien que les relations de récurrence obtenues soient classiques, les questions ne sont pas simples car leur formulation approximative nécessite une petite prise d'initiative mathématique pour rendre les choses parfaitement rigoureuses.
- La troisième partie, indépendante des précédentes, porte sur un algorithme de recherche d'une α-pseudo médiane dans un tableau de taille n ($\alpha \in]0;1[$ à déterminer en fin de partie) : au moins n^α éléments sont inférieurs (respectivement supérieurs) à une α-pseudo médiane. Deux implémentations sont proposées : l'une en effectuant des manipulations sur le tableau en entrée (la plus délicate du sujet), l'autre en utilisant une structure d'arbre (sans difficulté). Il s'ensuit une étude théorique de l'algorithme et de ses extensions.
- La quatrième partie propose de montrer que la complexité du tri rapide est sous-quadratique ($o(n^2)$) dans le pire des cas lorsqu'une pseudo médiane est choisie comme pivot. De longueur modeste, cette partie nécessite de posséder une petite expérience sur les questions de complexité. En effet, l'énoncé demande de justifier qualitativement des résultats tout en restant extrêmement rigoureux dans certains calculs. La dernière question est même ouverte.

La progression du problème est limpide, de nombreuses questions sont classiques ou faciles. Il est possible de traiter rapidement les trois premières parties, à moins d'être freiné par certaines approximations mathématiques de l'énoncé. Attention, les points de la dernière partie (sûrement nombreux) sont difficiles à décrocher, c'est pourquoi il ne faut pas se brûler les ailes en traitant précipitamment le début de l'épreuve.

Indications

Partie I

I.B Le tri à bulles est rapide à décrire et son analyse en complexité est assez simple. C'est le choix adopté dans ce corrigé.

I.C.1 Décrire le principe consistant, pour un tableau v de pivot p, à déplacer un indice g vers la droite, à déplacer un indice d vers la gauche et à échanger $v(g)$ et $v(d)$ à chaque fois que $v(g) > p > v(d)$.

Partie II

II.B.1 Supposer si besoin en première approximation que la séparation d'un tableau de taille $2n$ le coupe en deux sous-tableaux de taille n.

II.B.2 Pour tout $k \in \mathbb{N}^*$, utiliser $u_k = \mathrm{M}(2^k)/2^k$ pour trouver le terme général de la suite $(\mathrm{M}(2^k))_{k\geqslant 1}$.

Partie III

III.A.2 Introduire pour structurer les idées une référence *pas* contenant la distance entre deux éléments consécutifs d'un paquet de 3 dont on veut calculer la médiane : 3 *pas* contient alors la distance entre deux paquets de 3 consécutifs. À chaque parcours du tableau, *pas* est multiplié par 3.

III.C.1 L'algorithme calcule la médiane de $n/3$ paquets de 3, puis de $n/9$ paquets de 3 etc.

III.C.2 Établir le résultat par récurrence sur k en s'appuyant sur la version récursive de l'algorithme utilisée dans la question III.B.4.

III.C.3 Pour $k = 2$, le tableau suivant convient :

1	2	∞	3	4	∞	∞	∞	∞

où ∞ désigne une valeur suffisamment grande (qu'il est inutile d'expliciter). Utiliser ce modèle pour construire récursivement les tableaux souhaités.

III.C.4 Justifier que l'on obtient une $\ln 2/\ln 3$-pseudo médiane d'un tableau t de taille n en appliquant l'algorithme uniquement au sous-tableau $t[0..3^k - 1]$ où 3^k est la plus grande puissance de 3 inférieure à n.

Partie IV

IV.A Les résultats de la partie II suggèrent (qualitativement) que $\mathrm{C}(n_1) + \mathrm{C}(n_2)$ est d'autant plus grand que $|n_1 - n_2|$ l'est. Supposer que $\mathrm{K}_1 = \mathrm{K}_2 = 1$ pour ne pas rester bloqué par les imprécisions de l'énoncé.

IV.B Penser télescopage pour établir que $k < \sqrt{n/2}$ si $\alpha_k > n/2$. Développer $\sqrt{n/2}$ fois la relation de récurrence de la question IV.A (ne pas se soucier du problème posé par des indices non entiers).

IV.D Reprendre tous les raisonnements de cette partie dans le cas général d'une α-pseudo médiane avec $\alpha \in \,]\,0\,;1\,[$. Faire l'hypothèse que la complexité est majorée par $\mathrm{O}(n^\gamma)$ avec $\alpha\gamma \leqslant 1$ pour ne pas rester bloqué par les imprécisions de l'énoncé.

I. Tri rapide d'un tableau

I.A Une variable *tmp* est utilisée pour ne pas perdre la valeur de $t(j)$ au moment de déplacer $t(i)$ à la place j du tableau t.

```
let echange i j t =
   let tmp = t.(j) in
   t.(j) <- t.(i);
   t.(i) <- tmp;;
```

> Lorsque des nombres sont manipulés, il est possible d'échanger le contenu de deux références sans utiliser de variable intermédiaire :
>
> `a := !a + !b; b := !a - !b; a := !a - !b;`

I.B Décrivons l'algorithme de tri à bulles pour un tableau t de taille n, indicé de 0 à $n-1$. Il s'agit d'échanger deux valeurs consécutives dans t qui ne sont pas rangées dans l'ordre croissant. Ces échanges s'opèrent de gauche à droite en partant du premier élément (boucle `for` interne dans le programme). Un tel parcours du tableau de gauche à droite est effectué $n-1$ fois (boucle `for` externe) : à la fin du k-ième parcours, le k-ième plus grand élément du tableau t est placé en position $n-k$. C'est pourquoi il suffit d'arrêter le k-ième parcours à la position $n-k$. Finalement, le tableau est trié à la fin de l'exécution de l'algorithme. Une implémentation de cet algorithme est donnée par la fonction `tribulle` suivante, de type

```
tribulle : int vect -> unit = <fun>
```

```
let tribulle t =
   let n = vect_length t in
   for k = 1 to n-1 do
      for j = 1 to n-k do
         if t.(j) < t.(j-1)
         then echange j (j-1) t;
      done;
   done;;
```

Évaluons la complexité de l'algorithme de tri à bulles. Quel que soit le tableau en entrée, le nombre de comparaisons effectuées est inchangé : $n-k$ au cours du k-ième parcours pour k allant de 1 à $n-1$, donc au total

$$\sum_{k=1}^{n-1} n-k = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2}$$

L'algorithme de tri à bulles est de complexité quadratique en termes de comparaisons.

> D'autres algorithmes peuvent convenir, en gardant à l'esprit que l'énoncé incite fortement à trier un tableau, et non une liste. Pour cela, les algorithmes « simples » (disons de complexité quadratique) les plus adaptés sont le tri à bulles et le tri par sélection, tous deux « en place », c'est-à-dire en modifiant le tableau lui-même, sans utiliser d'espace supplémentaire. Le tri par insertion quant à lui ne peut pas être réalisé en place et il nécessite de translater des morceaux de tableau.

I.C.1 Dans l'algorithme suivant, le pivot p est le premier élément du (sous-)tableau t de taille n. Si ce n'est pas le cas, il suffit préalablement d'échanger le pivot avec le premier élément du tableau. Notons g (gauche) la 2^{e} position du tableau t et d (droite) la dernière. L'idée consiste

- à déplacer la position g vers la droite tant que $t(g)$ est inférieur au pivot p ;
- puis à déplacer la position d vers la gauche tant que $t(d)$ est strictement supérieur au pivot p ;
- enfin à échanger $t(g)$ et $t(d)$ (si $g < d$) car $t(d) \leqslant p < t(g)$.

Ces trois étapes sont répétées tant que la position gauche g est à gauche de la position droite d. À l'issue de ces opérations, tous les éléments à gauche de la position d (incluse) sont inférieurs au pivot et tous ceux à droite lui sont supérieurs. De ce fait, il reste à échanger le pivot avec $t(d)$ pour obtenir la séparation attendue. Le nombre de comparaisons de valeurs du tableau est égal au nombre de positions possibles de g et d, à savoir entre $n-1$ et $n+1$.

La phase de séparation est linéaire en la taille du tableau.

I.C.2 La fonction `separation` ci-dessous programme l'algorithme décrit dans la question précédente (pour un tableau de taille $n \geqslant 2$).

```
let separation v i1 i2 =
   let pivot = v.(i1)
   and g = ref (i1+1)
   and d = ref i2 in
      while !g <= !d do
         while !g <= i2 && v.(!g) <= pivot do
            incr g                              (* montee de g *)
         done;
         while v.(!d) > pivot do
            decr d                             (* descente de d *)
         done;
         if !g < !d   (* aucun echange si g et d se croisent *)
            then (
               echange !g !d v;
               incr g;
               decr d )
      done;
      echange i1 !d v;
      !d;;
```

> Une autre méthode, que l'on retrouve traditionnellement dans un cours de taupe, consiste à déplacer le pivot vers la droite en utilisant deux références :
>
> - `!g` désigne la position de fin d'un préfixe de `t` constitué de valeurs inférieures ou égales au pivot (ce dernier se trouvant à la position `!g`) ;
> - `!d` +1 désigne la position de début d'un suffixe de `t` constitué de valeurs strictement supérieures au pivot (initialement, ce suffixe est vide).
>
> Lorsque `!g` =`!d`, le tableau vérifie la propriété de partition pour le pivot se trouvant à la position `!g`.