E3A Informatique MP 2006 — Corrigé

Ce corrigé est proposé par Guillaume Batog (ENS Cachan) ; il a été relu par Benjamin Monmege (ENS Cachan) et Vincent Puyhaubert (Professeur en CPGE).

Cette épreuve propose d'implémenter quatre algorithmes de tri sur des tableaux et d'étudier la complexité des plus simples d'entre eux.

- La première partie porte sur le tri par sélection. Il s'agit exclusivement de questions de cours de première année, à savoir l'écriture itérative puis récursive de l'algorithme de tri, ainsi que l'analyse de sa complexité.
- La deuxième partie concerne le tri par insertion. Elle est de même nature que la première. On y aborde entre autres les approches séquentielle et dichotomique de la recherche de la position d'insertion.
- La troisième partie porte sur le tri rapide. La principale difficulté tient à la compréhension de l'algorithme présenté dans l'énoncé. Les premières questions consistent simplement à appliquer cet algorithme sur un exemple. L'implémentation de la fonction de partition est la plus difficile du sujet car elle nécessite une part d'initiative, notamment dans l'écriture de sous-programmes.
- La quatrième partie, la plus longue, concerne le tri par tas. Dans cette méthode, un tableau s'interprète comme un arbre binaire presque complet que l'on munit d'une structure de tas. Les opérations de base sur les tas sont utilisées pour trier le tableau en retour. Les premières questions, plutôt faciles, permettent de se familiariser avec la structure de tas à travers de nombreux tests. L'implémentation de l'algorithme de tri n'a lieu qu'au cours des trois dernières questions, plus difficiles car les algorithmes ne sont pas présentés dans l'énoncé.

Depuis 2007, le format de l'épreuve d'informatique des E3A a changé. Bien que ce sujet ne soit plus représentatif, il constitue un entraînement à la programmation sur les tableaux. Ses trois premières parties peuvent être traitées en n'utilisant que le programme de première année.

Indications

1.2 Programmer l'algorithme présenté en début de partie.

1.3 Traduire la boucle `for` de `triSelec1` sous forme récursive avec `let rec`.

1.4 Dénombrer les comparaisons effectuées lors d'un appel à la fonction `indiceMaxi`.

2.1 Au cours d'une recherche séquentielle, le tableau est parcouru de gauche à droite. Pour le calcul de la complexité en moyenne, les positions d'insertion sont supposées équiprobables.

Le principe de la dichotomie consiste à ce qu'un problème sur une donnée se réduit au même problème sur une sous-donnée de taille moitié. Montrer que, quel que soit le tableau de taille n, le nombre $\mathrm{T}(n)$ de comparaisons effectuées par l'algorithme dichotomique vérifie

$$\mathrm{T}(n) \leqslant 1 + \max\left\{\mathrm{T}\left(\left\lfloor\frac{n}{2}\right\rfloor\right), \mathrm{T}\left(\left\lceil\frac{n}{2}\right\rceil\right)\right\}$$

2.2 Ne pas déborder du tableau pendant son parcours.

2.3 Écrire une fonction auxiliaire récursive réalisant la recherche dichotomique et ayant deux arguments qui représentent les extrémités du sous-tableau considéré.

2.4 Prendre garde à ne jamais écraser les valeurs du tableau.

2.5 Programmer l'algorithme présenté en début de partie.

3.3 Faire attention au cas où un des sous-tableaux de la partition est vide. Utiliser la fonction `partition` comme une « boîte noire ».

3.4 Implémenter les sous-programmes préconisés par l'énoncé.

- `parcoursMontant` et `parcoursDescendant` réalisent chacun *un seul* parcours décrit dans l'énoncé. Chaque fonction retourne la position à laquelle le parcours s'est arrêté.
- `parcoursDouble` applique successivement les deux parcours précédents jusqu'à ce qu'ils se rejoignent. Cette fonction retourne la position où le pivot doit être placé.

Dans un premier temps, écrire les programmes en supposant que les valeurs du tableau sont deux à deux distinctes, afin d'écarter d'éventuels problèmes portant sur les inégalités larges ou strictes dans les tests.

4.1 Remplir successivement les différents niveaux de l'arbre en parcourant le tableau de gauche à droite.

4.2 Conjecturer le résultat sur l'exemple précédent.

4.4 Chercher une même fonction « inverse » pour `indiceFilsG` et `indiceFilsD`.

4.5 Écrire la condition d'être une feuille à l'aide de `indiceFilsG`.

4.6 Écrire la condition correspondante à l'aide de `indiceFilsD` et `estFeuille`.

4.7 Écrire la condition correspondante à l'aide de `indiceFilsG`.

4.8 Distinguer les cas où un nœud a 0, 1 ou 2 fils.

4.9 Faire descendre le nœud courant jusqu'à ce que la condition de domination soit respectée. Justifier que, lors d'une descente, le nœud courant doit être échangé avec son fils de valeur maximale.

4.10 Les feuilles sont des tas mais leurs pères ne le sont pas forcément.

4.11 Programmer l'algorithme présenté en début de partie.

Selon le rapport du jury, la discrimination entre les compositions s'est surtout faite sur les points suivants :

- une mauvaise lecture de l'énoncé ;
- des réponses inutilement compliquées ;
- les exercices inhabituels (parties 3 et 4) délaissés et les exercices familiers (parties 1 et 2) traités à la va-vite ;
- l'absence de commentaires, la programmation obscure et la mauvaise présentation des programmes (le rapport conseille de donner des noms parlants aux identificateurs et d'adopter un style d'indentation clair).

Rappelons que l'élément d'indice 0 d'un tableau ne sera jamais pris en compte tout au long du sujet. La longueur des tableaux est toujours en paramètre des fonctions à programmer. C'est pourquoi il n'est aucunement nécessaire d'utiliser la fonction `vect_length`.

1. Le tri par sélection

1.1 La fonction `indiceMaxi` parcourt le tableau `t` de gauche à droite à la recherche du maximum. L'invariant de la boucle `for` s'énonce ainsi : à la fin de l'étape `i`,

- `!max` contient le maximum des entiers stockés entre les positions 1 et `i` ;
- `!res` est la position du maximum le plus à gauche entre ces mêmes positions.

L'inégalité stricte `t.(i) > !max` évite de modifier la position du maximum le plus à gauche si ce maximum est rencontré une nouvelle fois.

```
let indiceMaxi t n =
  let res = ref 0
  and max = ref 0 in
    for i = 1 to n do
      if t.(i) > !max
      then
        begin
          max := t.(i);
          res := i
        end
    done;
    !res;;
```

Le rapport du jury mentionne une erreur à ne pas commettre : « Plusieurs candidats ont commis la faute de chercher la valeur du plus grand élément (algorithme sans doute traité en classe), puis faire un deuxième parcours pour trouver le plus grand indice. »

1.2 La fonction `triSelec1` traduit directement l'algorithme de tri par sélection présenté en début d'énoncé. L'appel `permute t i j` réalise la permutation des entiers aux positions `i` et `j` du tableau `t`. La variable de stockage `stock` permet de ne pas perdre l'entier `t.(i)` lors de la première affectation.

```
let permute t i j =
  let stock = t.(i) in
    t.(i) <- t.(j);
    t.(j) <- stock;;

let triSelec1 t n =
  for i = n downto 1 do
    let j = indiceMaxi t i in
      permute t i j
  done;;
```

> Dans la description de `triSelec1` de l'énoncé, la mention *donnée-résultat* pour le tableau d'entiers signifie que la fonction modifie ce tableau au cours de son exécution : elle effectue un tri *sur place*. Ainsi il est inutile de réserver un espace mémoire supplémentaire pour construire le tableau trié.

1.3 Le programme suivant est l'écriture récursive de la boucle `for` de `triSelec1`.

```
let rec triSelec2 t n =
  match n with
    | 0 -> ()
    | _ ->
        let j = indiceMaxi t n in
          permute t n j;
          triSelec2 t (n-1);;
```

1.4 Calculons le nombre de comparaisons d'entiers effectuées lors de l'algorithme. Elles ont lieu uniquement au cours de la recherche d'un maximum dans un tableau. Pour un tableau de taille i, il y en a exactement i. Puisque l'algorithme consiste à rechercher un maximum dans des tableaux de taille successivement n, $n-1$, …, 1, le nombre total de comparaisons vaut

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

Le tri par sélection effectue $\Theta(n^2)$ comparaisons d'entiers.

> La complexité d'un algorithme porte sur le nombre d'opérations élémentaires effectuées lors de son exécution et exprime un ordre de grandeur de ce nombre, lorsque la taille des données devient importante (c'est-à-dire, lorsque cette taille tend vers l'infini, d'où l'utilisation des notations de Landau). C'est pourquoi il faut préciser les opérations élémentaires dénombrées.
>
> Pour le tri par sélection, le nombre de comparaisons effectuées est toujours égal à $n(n+1)/2$ sur un tableau de taille n. La notation $\Theta(n^2)$ traduit le fait que la complexité est quadratique à la fois dans le pire des cas et dans le meilleur des cas. Rappelons que $a_n = \Theta(b_n)$ signifie que $a_n = \mathrm{O}(b_n)$ et $b_n = \mathrm{O}(a_n)$. C'est un « équivalent large » en quelque sorte : il existe α et β tels que pour n assez grand, $\alpha\, a_n \leqslant b_n \leqslant \beta\, a_n$.