Institut Galilée — Algorithmique, arbres et graphes
Année 2005-2006 — L2

Partiel du lundi 27 mars 2006

Polycopié du cours autorisé – pas de calculatrice
Le barème est indicatif, il pourra être modifié

Exercice 1 (Notation asymptotique)**.** 3 pt

1. *Est-ce que* $(n+3)\log n - 2n = \Omega(n)$ *?* (1,5 pt)
2. *Est-ce que* $2^{2n} = O(2^n)$ *?* (1,5 pt)

Correction 1. 1. On montre qu'il existe une constante positive c et un rang n_0 à partir duquel $(n+3)\log n - 2n \geq cn$.

$$\begin{aligned} (n+3)\log n - 2n &\geq n(\log n - 2) \quad (\forall n > 0) \\ \text{pour } n \geq 8, \quad \log n - 2 &\geq \log 8 - 2 = 1 \\ \text{Donc } \forall n \geq n_0 = 8, \quad (n+3)\log n - 2n &\geq 1 \times n. \end{aligned}$$

2. On montre que c'est faux, par l'absurde. Supposons que ce soit vrai, alors :

$$\exists c > 0, \exists n_0 \in \mathbb{N}, \forall n \geq n_0, 2^{2n} \leq c \times 2^n.$$

Donc par croissance du log, pour $n \geq n_0$ et $n > 0$:

$$\begin{aligned} 2 \times n &\leq \log c + n \\ \text{ce qui donne} \quad n &\leq \log c \end{aligned}$$

L'ensemble des entiers naturels n'étant pas borné et c étant une constante, ceci est une contradiction.

Exercice 2 (Complexité en moyenne du tri gnome)**.** *Le but de cet exercice est d'écrire le tri gnome en C et de déterminer le nombre moyen d'échanges effectués au cours d'un tri gnome.* 7 pt

Rappel du cours. « Dans le tri gnome, on compare deux éléments consécutifs : s'ils sont dans l'ordre on se déplace d'un cran vers la fin du tableau (ou on s'arrête si la fin est atteinte) ; sinon, on les intervertit et on se déplace d'un cran vers le début du tableau (si on est au début du tableau alors on se déplace vers la fin). On commence par le début du tableau. »

1. *Écrire une fonction C de prototype* `void trignome(tableau_t t)` *effectuant le tri gnome.* (2,5 pt)

Une inversion *dans une entrée* $a_0, \ldots, a_{n-1}$ *est la donnée d'un couple d'indices* (i, j) *tel que* $i < j$ *et* $a_i > a_j$.

Rappel. Un échange d'éléments entre deux indices i *et* j *dans un tableau est une opération qui intervertit l'élément à l'indice* i *et l'élément à l'indice* j*, laissant les autres éléments à leur place.*

2. *Si le tri gnome effectue un échange entre deux éléments, que peut on dire de l'évolution du nombre d'inversions dans ce tableau avant l'échange et après l'échange (démontrer) ?* (2,5 pt)

On suppose que le nombre moyen d'inversions dans un tableau de taille n *est* $\frac{n(n-1)}{4}$.

3. *Si un tableau* t *de taille* n *contient* $f(n)$ *inversions, combien le tri gnome effectuera d'échanges sur ce tableau (démontrer) ? En déduire le nombre moyen d'échanges effectués par le tri gnome sur des tableaux de taille* n*.* (2 pt)

Correction 2.

```
1.1  void trignome(tableau_t *t){
 2     int i = 0; //                  On part du début du tableau t
 3     while ( i < taille(t) - 1 ){// Tant qu'on a pas atteint la fin de t
 4       if ( cmptab(t, i, i + 1) < 0 ){// Si (i, i + 1) inversion alors :
 5         echangetab(t, i, i + 1); // 1) on reordonne par échange;
 6         if (i > 0) i--;//           2) on recule, sauf si on était
 7         else i++; //                  au début, auquel cas on avance.
 8       }
 9       else i++; //                   Sinon, on avance.
10     }
11   }
```

2. La seule hypothèse est, qu'au cours de l'exécution du tri grome (sur une entrée non spécifiée), à une certaine étape, un échange à eu lieu. Soit t' le tableau juste avant cet échange, t'' le tableau juste après cet échange, et i_0 la valeur de la variable i au moment de l'échange. Un échange ne se produit que lorsque $t[i] > t[i+1]$ et il s'agit d'un échange entre $t[i]$ et $t[i+1]$. Ainsi, dans t', on avait $t'[i_0] < t'[i_0+1]$ et t'' est égal à t' dans lequel on a procédé à l'échange entre les éléments d'indices i_0 et i_0+1. Cet échange élimine exactement une inversion. En effet :
   - dans t', (i_0, i_0+1) est une version, pas dans t''
   - les autres inversions sont préservées :
     - lorsque $i < j$ et i, j tous deux différents de i_0 et i_0+1, (i, j) est une inversion dans t' ssi c'est une inversion dans t'' ;
     - lorsque $i < i_0$ alors : (i, i_0) est une inversion dans t' ssi (i, i_0+1) est une inversion dans t'' et (i, i_0+1) est une inversion dans t' ssi (i, i_0) est une inversion dans t'' ;
     - lorsque $i_0+1 < j$ alors : (i_0, j) est une inversion dans t' ssi (i_0+1, j) est une inversion dans t'' et (i_0+1, j) est une inversion dans t' ssi (i_0, j) est une inversion dans t'' ;
   - et ceci prend en considération toutes les inversions possibles dans t' et t''.
3. Nous venons de voir que tout échange au cours du tri gnome élimine exactement une inversion. Le nombre d'échanges effectués au cours du tri gnome de t est donc la différence entre le nombre d'inversions au départ, $f(n)$, et le nombre d'inversions à fin du tri. Mais le tableau final est trié et un tableau trié ne contient aucune inversion. Donc le nombre d'échange est simplement $f(n)$.

   Le nombre d'échanges est égal au nombre d'inversions dans le tableau inital. Donc le nombre moyen d'échanges sur des tableaux de taille n est $\frac{n(n-1)}{4}$.

Exercice 3 (Tri par base)**.** *Soit la suite d'entiers décimaux* 141*,* 232*,* 045*,* 112*,* 143*. On utilise un tri stable pour trier ces entiers selon leur chiffre le moins significatif (chiffre des unités), puis pour trier la liste obtenue selon le chiffre des dizaines et enfin selon le chiffre le plus significatif (chiffre des centaines).* 10 pt

Rappel. Un tri est stable lorsque, à chaque fois que deux éléments ont la même clé, l'ordre entre eux n'est pas changé par le tri. Par exemple, en triant $(2, a), (3, b), (1, c), (2, d)$ *par chiffres croissants, un tri stable place* $(2, d)$ *après* $(2, a)$*.*

1. *Écrire les trois listes obtenues. Comment s'appelle cette méthode de tri ?* (1 pt)

On se donne un tableau t contenant N entiers entre 0 *et* $10^k - 1$*, où k est une constante entière. Sur le principe de la question précédente (où* $k = 3$ *et* $N = 5$*), on veut appliquer un tri par base, en base 10 à ces entiers.*

On se donne la fonction auxiliaire :

```
int cle(int x, int i){
  int j;
```

```
  for (j = 0; j < i; j++)
    x = x / 10; // <- arrondi par partie entière inférieure.
  return x % 10;
}
```

2. *Que valent* `cle(123, 0)`, `cle(123, 1)`, `cle(123, 2)` *(inutile de justifier votre réponse) ? Plus généralement, que renvoie cette fonction ?* (1,5 pt)

On suppose que l'on dispose d'une fonction auxiliaire de tri `void triaux(tableau_t t, int i)` *qui réordonne les éléments de* t *de manière à ce que*

$$\texttt{cle(t[0], i)} \leq \texttt{cle(t[1], i)} \leq \ldots \leq \texttt{cle(t[N - 1], i)}.$$

On suppose de plus que ce tri est stable.

3. *Écrire l'algorithme de tri par base du tableau* t *(utiliser la fonction* `triaux`*). On pourra considérer que* k *est un paramètre entier passé à la fonction de tri.* (2 pt)
4. *Si le temps d'exécution en pire cas de* `triaux` *est majoré asymptotiquement par une fonction* $f(N)$ *de paramètre la taille de* t*, quelle majoration asymptotique pouvez donner au temps d'exécution en pire cas de votre algorithme de tri par base ?* (1 pt)
5. *Démontrer par récurrence que ce tri par base trie bien le tableau* t*. Sur quelle variable faites vous la récurrence ? Où utilisez vous le fait que* `triaux` *effectue un tri stable ?* (3 pt)
6. *La fonction* `triaux` *utilise intensivement la fonction à deux paramètres* `cle`*. Si on cherche un majorant* $f(N)$ *au temps d'exécution de* `triaux`*, peut on considérer qu'un appel à* `cle` *prend un temps borné par une constante ?* (1 pt)
7. *Décrire en quelques phrases une méthode pour réaliser la fonction* `triaux` *de manière à ce qu'elle s'exécute en un temps linéaire en fonction de la taille du tableau (on pourra utiliser une structure de donnée).* (1,5 pt)

Correction 3.

1. Liste 1 : 141, 232, 112, 143, 045. Liste 2 : 112, 232, 141, 143, 045. Liste 3 : 045, 112, 141, 143, 232. Ce tri s'appelle un tri par base.
2. On a `cle(123, 0) = 3`; `cle(123, 1) = 2`; `cle(123, 2) = 1`. Cette fonction prend un nombre n et un indice i en entrée et renvoie le $i+1$ ième chiffre le moins significatif de l'expression de n en base 10. Autrement dit, si n s'écrit $\overline{b_{k-1} \ldots b_0}$ en base 10, alors `cle(n, i)` renvoie b_i si i est inférieur à $k-1$ et 0 sinon.

3.1
```
  void trignome(tableau_t t, int k){
    int i;
    for (i = 0; i < k; i++){// k passages
      triaux(t, i);
    }
  }
```

4. Sur un tableau de taille N, le tri gnome effectue k appels à `triaux` et chacun de ces appels se fait aussi sur un tableau de taille N. Les autres opérations (contrôle de boucle) sont négligeables. Chacun de ces appels est majoré en temps par $f(N)$. Dans le pire cas, le temps d'exécution du tri gnome est donc majoré par $kf(N)$.
5. Étant donné un entier n d'expression $\overline{b_{k-1} \ldots b_0}$ en base 10, pour $0 \leq i \leq k-1$, on lui associe l'entier $c_i(n)$ d'expression $\overline{b_i \ldots b_0}$ (les $i+1$ premiers digits les moins significatifs de n).
   On démontre par récurrence sur i qu'en $i+1$ étapes de boucle le tri gnome range les éléments du tableau t par ordre croissant des valeurs de c_i. Cas de base. Pour $i = 0$ (première étape de boucle) le tri gnome a fait un appel à `triaux` et celui-ci a rangé les éléments de t par ordre croissant du digit le moins significatif. Comme c_0 donne ce digit le moins significatif, l'hypothèse est vérifiée. On suppose l'hypothèse vérifié après l'étape de boucle i ($i+1$ ème étape). En une étape supplémentaire par l'appel à `triaux`, les éléments de t sont trié selon

leur digit d'indice $i+1$ ($i+2$ ème digit). Soient $t[j]$ et $t[k]$ deux éléments quelconques du tableau après cette étape, avec $j < k$. Comme le tri auxilliaire, `triaux` est stable si $t[j]$ et $t[k]$ ont même digits d'indice $i+1$ alors ils sont rangés dans le même ordre qu'à l'étape précédente. Mais par hypothèse à l'étape précédente ces deux éléments étaient rangés selon des c_i croissants, il sont donc rangés par c_{i+1} croissants. Si les digits d'indices $i+1$ de $t[j]$ et $t[k]$ différent alors celui de $t[k]$ est le plus grand et ainsi $c_{i+1}(t[j]) < c_{i+1}(t[k])$. Dans les deux cas $t[j]$ et $t[k]$ sont rangés par c_{i+1} croissants. Ainsi l'hypothèse est vérifiée après chaque étape de boucle.

Lorsque $i = k-1$, pour tout indice j du tableau $c_i(t[j]) = t[j]$ ainsi le tableau est bien trié à la fin du tri gnome.

La récurrence est faite sur i qui va de 0 à $k-1$ (on peut aussi considérer qu'elle porte sur k). La stabilité de `triaux` a servi à démonter le passage de l'étape i à l'étape $i+1$ de la récurrence.

6. L'exécution de la fonction `cle` demande $i+1$ étapes de boucle. Comme, lors d'un appel à `triaux`, i est borné par k, le temps d'exécution de `cle` est linéaire en k (). Mais k est considéré comme une constante. Le temps d'exécution de `cle` peut don être considéré comme borné par une constante ().
7. Pour réaliser `triaux` en temps linéaire, il suffit de faire un tri par dénombrement, avec données satellites. Il est alors pratique d'utiliser une structure de pile : on crée dix piles vides numérotées de 0 à 9, on parcours t du début à la fin en empilant chaque élément sur la pile de numéro la valeur de la clé (son $i+1$ ème digit). Ceci prend N étapes. Lorsque c'est fini on dépile en commençant par la pile numéroté 9 et en stockant les éléments dépilés dans t en commençant par la fin. Ceci prend encore N étapes.