Examen du 08/02/2014 (1h)

TOP : Techniques and tOols for Programming – 1A

Documents interdits, à l'exception d'une feuille A4.

La notation tiendra compte de la présentation et de la clarté de la rédaction.
Le barême, approximatif, est donné sur 10 points puisque l'épreuve dure 1h ($\frac{10}{20} = 0,5 = \frac{1h}{2h}$).

★ Exercice 1: Complexité algorithmique (3pt).

▷ **Question 1:** (1pt) Étudiez le nombre d'additions réalisées par les algorithmes suivants dans le meilleur cas, le pire cas, puis dans le cas moyen en supposant que les tests ont une probabilité de $\frac{1}{2}$ d'être vrai.

Listing 1.a

```
1 for (i <- 1 to n) {
2    if (T(i)>a) {
3        s += T(i)
4    }
5 }
```

Listing 1.b

```
1 if (a > b) {
2   for (i <- 1 to n) {
3     x += a
4   }
5 } else {
6    x += b
7 }
```

▷ **Question 2:** (2pt) Donnez la complexité des programmes suivants. Vous donnerez une borne supérieure avec un $O()$ dans un premier temps, puis vous affinerez votre calcul en utilisant la notation $\Theta()$.

Listing 2.a

```
1 for (i <-  5 to n-5) {
2    for (j <- i-5 to i+5) {
3      x += 3
4    }
5 }
```

Listing 2.b

```
1 for (i <- 1 to n) {
2    for (j <- 1 to i) {
3         for (k <- 1 to j) {
4              x += 4
5         }
6    }
7 }
```

★ Exercice 2: La dichotomie (4pt).

▷ **Question 1:** Écrivez une fonction **récursive en scala** cherchant l'indice d'un élément donné dans un tableau trié. La recherche doit être dichotomique. Le prototype de la fonction doit être le suivant :

```
def dicho(tab:Array[Int], elm:Int):Int
```

La fonction doit retourner l'indice où se trouve l'élément dans tab s'il y est, ou -1 sinon.

▷ **Question 2:** Calculez la complexité de cette fonction. **Réponse**

$O(log(n))$ car on divise la quantité de chose à faire par 2 à chaque fois. Donc, en 1 étape, je gère 1 élément au max. En 2 étapes, 2 fois plus. En 3 étapes, encore 2 fois plus. En N étapes, je gère 2^N éléments. Donc, pour gérer p éléments, il faut n étapes tel que $2^n > p$, ie, $n > log(p)$. CQFD.

Fin réponse

▷ **Question 3:** Montrez la terminaison de cette fonction. **Réponse**

Quand on fait une fonction récursive, il est très important de montrer qu'elle s'arrête tout le temps, pour 2 raisons : (1) écrire une fonction récursive qui boucle à l'infini est facile et courant ; (2) montrer ceci est relativement simple, c'est pas une preuve avancée.

Une condition suffisante pour montrer la terminaison d'un algo recursif (mais pas forcément nécessaire), c'est de montrer qu'il y a une grandeur qui varie de façon strictement monotone, et qu'elle atteint à coup sûr les cas d'arrêt de la récursion.

Ici : la taille du tableau est divisé par 2 à chaque étape, et on s'arrête quand la taille est inférieure ou égale à 1. Suite strictement monotone (attention aux divisions entières pour ca), effectivement convergeante vers le cas terminal, c'est bon.

Si on avait marqué explicitement que la longueur entre min et max est 0 ou 1, on aurait pu se faire avoir avec des cas où max passe à gauche de min. Peut-être bien que cela aurait pu passer à travers, ie partir se promener chez les négatifs et donc partir à l'infini ($-\infty$). Mais cette simple précaution (écrire $i <= 1$ quand on pense à $i \in [0, 1]$) nous assure que ce ne sera pas le cas.

Fin réponse

▷ **Question 4:** Dérécursivez la fonction précédente, en justifiant ce que vous faites et pourquoi vous avez le droit de le faire. Le programme résultant doit être écrit en scala.

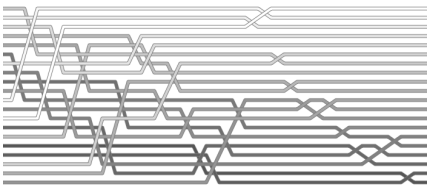
Réponse

Fin réponse

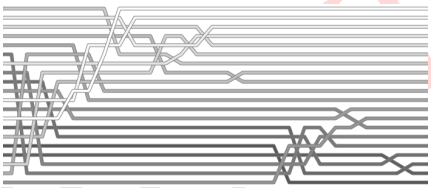
★ **Exercice 3: Identification d'algorithmes de tri** (d'après Aldo Cortesi – 3pt).

Les schémas suivants montrent le fonctionnement de divers algorithmes de tris. Chaque trait grisé indique une valeur, et l'axe des abscisses montre le temps qui passe tandis que l'axe des ordonnées montre la position de chaque valeur (=trait) dans les différentes cases du tableau. La case n°1 est en haut, et la case n°20 est en bas, et une couleur plus claire signifie une valeur plus petite. Quand deux traits se croisent, c'est que l'algorithme a inversé les deux valeurs à cet instant.

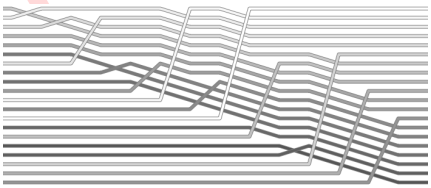
▷ **Question 1:** Identifiez le comportement des algorithmes suivants : tri à bulle, tri par insertion, tri par sélection, shell sort, quick sort. Argumentez vos réponses.



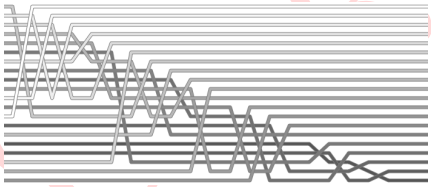
Algorithme A.



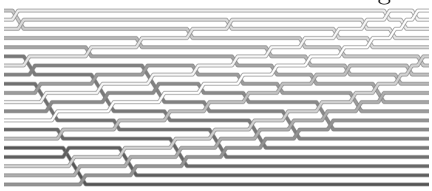
Algorithme B.



Algorithme C.



Algorithme D.



Algorithme E.

Réponse

Tri a bulle : parcours successifs du tableau en comparant les voisins 2 à 2. S'ils sont mal triés, on les inverse. Ce comportement a tendance à pousser les grosses valeurs vers la fin. On peut reconnaitre ce comportement dans l'algorithme B.

Tri par insertion : invariant : ce qui est avant la frontière est trié. A chaque étape, je prend l'élément juste après la frontière, et je le met à sa place dans la partie déjà triée. On peut reconnaitre ce comportement dans l'algorithme E.

Tri par selection : a chaque étape, je selectionne le minimum de la partie non triée et le pose à la frontière. On reconnait l'algorithme A.

Shell sort : comme un bubble sort, mais on commence par trier avec un écartement supérieur à 1. Au lieu d'inverser des voisins, on inverse des cases à distance 3 puis 2 puis on fait un bubble standard, mais sur un tableau un peu prétrié. C'est l'algorithme C.

Quick sort : c'est un algorithme récursif qui trie une partie du tableau puis l'autre (les parties ne sont pas forcément de taille identique). C'est l'algorithme D.

Fin réponse