EPFL - Information, Calcul et Communication - Partie théorique

# Série 2: Solutions

## 1 Complexités temporelles

algorithme 1: La complexité temporelle est $\Theta(n)$: dans le pire des cas (à savoir quand tous les nombres de la liste $L$ sont plus petits ou égaux à $M$), la boucle "tant que" est exécutée $n$ fois, car le nombre $i$ est incrémenté de 1 à chaque passage de la boucle.

algorithme 2: La complexité temporelle est également $\Theta(n)$: de l'ordre de $2n$ opérations sont effectuées, mais le facteur 2 ne nous importe pas ici.

algorithme 3: La complexité temporelle est $\Theta(n^2)$: ici, il y a deux boucles imbriquées, et le nombre d'instructions effectuées, dans le pire des cas (qui correspond au cas où toutes les différences $|L(i) - L(j)|$ sont plus petites ou égales à $x$), est de l'ordre du nombre de paires d'éléments dans l'ensemble $\{1, \ldots, n\}$, qui vaut $\frac{n(n-1)}{2}$.

algorithme 4: Malgré le fait qu'il y ait aussi deux boucles imbriquées dans cet algorithme, sa complexité temporelle est $\Theta(n)$: le nombre d'opérations effectuées à l'intérieur de chaque boucle "tant que" est au plus au plus égal à 5.

## 2 Création d'algorithmes

**a)**

| **moyenne arithmétique** |
|---|
| entrée : *liste $L$ de nombres réels, de taille $n$*<br>sortie : *nombre réel $m_A$* |
| $m_A \longleftarrow 0$<br>***Pour*** *$i$ allant de 1 à $n$*<br>$\quad m_A \longleftarrow m_A + L(i)$<br>$m_A \longleftarrow m_A/n$<br>**Sortir :** $m_A$ |

**b)** En utilisant l'indication de l'énoncé, on remarque que $\log_2(m_G) = \frac{1}{n}\left(\log_2(L(1)) + \cdots + \log_2(L(n))\right)$, et donc on peut écrire:

| **moyenne géométrique** |
|---|
| entrée : *liste $L$ de nombres réels positifs, de taille $n$*<br>sortie : *nombre réel positif $m_G$* |
| $M \longleftarrow ()$ *(liste vide)*<br>***Pour*** *$i$ allant de 1 à $n$*<br>$\quad M \longleftarrow M + (\log_2(L(i)))$ *(ajouter l'élément $\log_2(L(i))$ à la liste $M$)*<br>$m_G \longleftarrow 2^{\textbf{moyenne arithmétique}(M,n)}$<br>**Sortir :** $m_G$ |

**c)** Voici une possibilité: (mais on peut penser à d'autres façons de faire)

**le plus grand produit**

entrée : *liste $L$ de nombres réels positifs, de taille $n$*
sortie : *nombre réel positif $r$*

```
Si L(1) > L(2)
    x1 ⟵ L(1)
    x2 ⟵ L(2)
Sinon
    x1 ⟵ L(2)
    x2 ⟵ L(1)
Pour i allant de 3 à n
    Si L(i) > x1
        x2 ⟵ x1
        x1 ⟵ L(i)
    Sinon
        Si L(i) > x2
            x2 ⟵ L(i)
Sortir : x1 × x2
```

**d)** La complexité temporelle de l'algorithme ci-dessus est $\Theta(n)$.

# 3 Trier un tableau

**a)** La complexité temporelle de l'algorithme de tri par insertion est $\Theta(n^2)$ dans le pire des cas (qui correspond au tableau trié initialement à l'envers): chacune des insertions de l'algorithme peut en effet coûter de l'ordre de $n$ opérations, et de l'ordre de $n$ insertions sont effectuées.

**b)** Une première chose à remarquer est qu'il ne suffit pas de trier chaque ligne du tableau $A$ séparément. En effet, si on fait ceci, on obtient à partir du tableau $A$ donné en exemple:

$$A' = \begin{pmatrix} 3 & 4 & 7 & 8 & 11 & 12 & 14 \\ 2 & 3 & 5 & 5 & 12 & 13 & 15 \end{pmatrix}$$

où on voit par exemple que la première colonne n'est pas ordonnée comme on le voudrait. Une deuxième étape est donc nécessaire, qui consiste à ordonner les colonnes une à une. En suivant les notations de l'énoncé, voici l'algorithme complet:

**tri de tableau**

entrée : *tableau $A$ de dimensions $n \times 2$*
sortie : *tableau trié $A''$*

```
A'(1) ⟵ tri par insertion(A(1))
A'(2) ⟵ tri par insertion(A(2))
Pour i allant de 1 à n
    A''(1,i) ⟵ min(A'(1,i), A'(2,i))
    A''(2,i) ⟵ max(A'(1,i), A'(2,i))
Sortir : A''
```

Dans l'exemple de l'énoncé, cet algorithme produit le tableau:

$$A'' = \begin{pmatrix} 2 & 3 & 5 & 5 & 11 & 12 & 14 \\ 3 & 4 & 7 & 8 & 12 & 13 & 15 \end{pmatrix}$$

qui est bien une version triée du tableau $A$ d'origine.

Ceci dit, il importe maintenant de *vérifier* que cet algorithme aboutit à une version triée du tableau pour *tout* tableau de départ $A$, et pas seulement pour l'exemple ci-dessus: clairement, il est vrai que $A''(1,i) \leq A''(2,i)$ pour tout $i$ du fait de la dernière étape de l'algorithme. Reste à vérifier que $A''(1,i) \leq A''(1,j)$ et $A''(2,i) \leq A''(2,j)$ pour tout $i < j$ (la dernière étape de l'algorithme ne garantit pas en effet que l'ordre dans chaque ligne est conservé). Pour cela, raisonnons par l'absurde et supposons que l'une de ces deux inégalités ne soit pas vérifiée, par exemple, que $A''(1,i) > A''(1,j)$ pour une paire $i < j$ donnée. Ceci veut dire que

$$\min(A'(1,i), A'(2,i)) > \min(A'(1,j), A'(2,j))$$

mais comme les lignes de $A'$ sont triées, on sait que $A'(1,i) \leq A'(1,j)$ et $A'(2,i) \leq A'(2,j)$; il n'y donc aucune possibilité que l'inégalité ci-dessus soit satisfaite, ce qui prouve que notre hypothèse ne peut être vraie et donc que $A''(1,i) \leq A''(1,j)$. Le même raisonnement vaut pour le max et l'inégalité $A''(2,i) \leq A''(2,j)$.

*Note:* Il est aussi possible de:

- trier d'abord les colonnes puis les lignes du tableau (on arrive alors à la version triée de l'énoncé).

- réunir les deux lignes du tableau $A(1)$ et $A(2)$ en une seule ligne $L$ de taille $2n$, puis trier cette liste $L$ avec le tri par insertion, et finalement réarranger cette liste en un tableau de taille $n \times 2$. Pour cela, on a deux possibilités (en fait, même beaucoup plus que ça):

$$\begin{pmatrix} L(1) & L(2) & \dots & L(n-1) & L(n) \\ L(n+1) & L(n+2) & \dots & L(2n-1) & L(2n) \end{pmatrix}$$

ou

$$\begin{pmatrix} L(1) & L(3) & \dots & L(2n-3) & L(2n-1) \\ L(2) & L(4) & \dots & L(2n-2) & L(2n) \end{pmatrix}$$

**c)** La complexité temporelle du premier algorithme ci-dessus est $\Theta(n^2)$: la partie qui demande en effet le plus de temps de calcul est celle constituée des deux tris par insertion (effectués l'un après l'autre); la dernière boucle (en $\Theta(n)$) prend un temps négligeable en comparaison. Notez que la complexité du dernier algorithme proposé ci-dessus est également $\Theta(n^2)$, mais prend plus de temps en pratique, car trier une liste de taille $2n$ avec le tri par insertion prend deux fois plus de temps que trier deux sous-listes de taille $n$ (exercice pour vous!).

# 4 Pour le plaisir: une superstar arrive dans une réception mondaine*

Il y a donc $n+1$ personnes présentes à la réception (sans vous compter vous-même). La stratégie pour identifier la superstar est la suivante:

- poser d'abord la question: "est-ce que la personne 1 connaît la personne 2?" Si la réponse est oui, alors forcément, la personne 2 est la superstar.

- si la réponse à cette première question est non, poser ensuite la question: "est-ce que la personne 1 connaît la personne 3?" A nouveau, si la réponse est oui, alors la personne 3 est la superstar.

- sinon, poser la question: "est-ce que la personne 1 connaît la personne 4?" et ainsi de suite...

- la dernière question que vous serez éventuellement amené(e) à poser est: "est-ce que la personne 1 connaît la personne $n+1$?" Si la réponse est oui, alors la personne $n+1$ est la superstar, sinon, c'est forcément la personne 1 qui est la superstar.