

Série 2

1 Complexités temporelles

Estimez la complexité temporelle des algorithmes ci-dessous (repris en partie de la dernière série) en fonction du paramètre n et en utilisant la notation $\Theta(\cdot)$.

algorithme 1
entrée : liste L de nombres entiers, de taille n , nombre entier M sortie : nombre entier positif ou nul s
<pre> <i>i</i> ← 1 Tant que $i \leq n$ et $L(i) \leq M$ <i>i</i> ← $i + 1$ <i>s</i> ← $i - 1$ Sortir : s </pre>

algorithme 2
entrée : nombre entier positif n sortie : nombre entier positif s
<pre> <i>s</i> ← 0 Pour i allant de 1 à $2n$ <i>s</i> ← $s + i$ Sortir : s </pre>

algorithme 3
entrée : liste L de nombres entiers, de taille n , nombre entier positif x sortie : oui/non
<pre> Pour i allant de 1 à $n - 1$ Pour j allant de $i + 1$ à n Si $L(j) - L(i) > x$ Sortir : oui Sortir : non </pre>

algorithme 4
entrée : nombre entier positif n sortie : nombre entier positif s
<pre> <i>s</i> ← 0 <i>j</i> ← 1 Pour i allant de 1 à n Tant que j n'est ni un multiple de 5 ni un multiple de 7 <i>j</i> ← $j + 1$ <i>s</i> ← $s + j$ <i>j</i> ← $j + 1$ Sortir : s </pre>

2 Création d'algorithmes

a) Ecrivez un algorithme qui calcule la *moyenne arithmétique* d'une liste L de n nombres réels :

$$m_A = \frac{L(1) + L(2) + \dots + L(n)}{n}$$

b) Ecrivez maintenant un algorithme qui *utilise l'algorithme précédent comme sous-algorithme* pour calculer la *moyenne géométrique* d'une liste L de n nombres réels positifs :

$$m_G = (L(1) \cdot L(2) \cdot \dots \cdot L(n))^{1/n}$$

Indication : Calculer $\log_2(m_G)$ en utilisant le fait que $\log_2(a \cdot b) = \log_2(a) + \log_2(b)$ et $\log_2(a^b) = b \log_2(a)$.

c) Ecrivez un algorithme qui calcule le produit des deux plus grands nombres d'une liste L de n nombres réels positifs (par exemple, si $L = (3, 6, 18, 12, 7)$ et $n = 5$, alors la sortie doit être $12 \times 18 = 216$).

d) Quelle est la complexité temporelle de votre dernier algorithme en fonction de la taille n de la liste L ? (utiliser la notation $\Theta(\cdot)$)

3 Trier un tableau

a) Estimez la complexité temporelle de l'algorithme de tri par insertion vu au cours, en fonction de la taille n de la liste L à trier (utiliser la notation $\Theta(\cdot)$). Rappelez-vous que la complexité temporelle d'un algorithme est définie dans notre cours comme le nombre d'instructions exécutées par l'algorithme *dans le pire des cas*, i.e., pour les pires données d'entrée possibles.

b) Soit maintenant n un nombre entier positif et A un tableau de nombres entiers, de dimensions $2 \times n$. Si $n = 7$, un tel tableau pourrait être par exemple :

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

On aimerait trier ce tableau de façon à ce que dans chaque ligne et dans chaque colonne, les nombres soient rangés dans l'ordre croissant (de gauche à droite et de haut en bas, respectivement). Une version triée du tableau ci-dessus est par exemple :

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

mais contrairement au cas d'une liste, qui n'a qu'une seule version triée, il y a ici plusieurs versions triées possibles du tableau A (par exemple, le 7 et le 8 peuvent être intervertis ci-dessus et le tableau reste trié).

Ecrivez un algorithme qui prenne en entrée le tableau A et sa dimension horizontale n , et dont la sortie soit une version triée du tableau A . Votre algorithme devra utiliser le sous-algorithme **tri par insertion**(L, n) d'une liste L de taille n .

Notation : $A(i, j)$ désigne l'élément de la i^e ligne et j^e colonne du tableau A (dans l'exemple ci-dessus : $A(2, 4) = 13$). On peut également utiliser la notation $A(1)$ et $A(2)$ pour désigner respectivement les première et seconde lignes du tableau A .

c) Quel est la complexité temporelle de votre algorithme ? (utiliser la notation $\Theta(\cdot)$)

4 Analyse d'un algorithme de tri

On considère l'algorithme suivant :

tri par insertion
entrée : liste L de taille n sortie : liste L triée
Pour i allant de 2 à n Si $L(i) < L(i - 1)$ $L \leftarrow \text{insérer}(L, i)$ Sortir : L

où le sous-algorithme insérer est défini par :

insérer
entrée : liste L , indice i sortie : liste L
Tant que $i > 1$ et $L(i) < L(i - 1)$ $L \leftarrow \text{permuter}(L, i, i - 1)$ $i \leftarrow i - 1$ Sortir : L

On exécute cet algorithme sur une liste de 100 éléments. Après 50 itérations de la boucle principale (c'est-à-dire lorsque $i = 51$), on observe que permuter a été appelé exactement 1275 fois au total. Que peut-on déduire ?

- ☐ La liste était déjà triée
- ☐ La liste était triée en ordre décroissant
- ☐ Les 51 premiers éléments étaient exactement en ordre inverse
- ☐ On ne peut rien conclure sans plus d'informations

5 Complexité asymptotique

Considérons trois algorithmes :

- Algorithme A : effectue $5n^2 + 100n \log(n) + 1000$ opérations
- Algorithme B : effectue $0.001n^3$ opérations

À partir de quelle valeur approximative de n l'algorithme B devient-il plus lent que l'algorithme A ?

- ☐ $n > 100$
☐ $n > 1000$
☐ $n > 5000$
☐ B est toujours plus rapide que A

6 Réutilisation d'un algorithme existant

On considère l'algorithme suivant qui cherche deux éléments d'une liste ordonnée dont la somme vaut une valeur cible T :

deux font la paire
entrée : <i>liste ordonnée L de n entiers, valeur cible T</i> sortie : <i>booléen</i>
<pre> <i>i</i> ← 1 <i>j</i> ← <i>n</i> Tant que <i>i</i> < <i>j</i> Si $L(i) + L(j) = T$ Sortir : Vrai Si $L(i) + L(j) < T$ <i>i</i> ← <i>i</i> + 1 Sinon <i>j</i> ← <i>j</i> - 1 Sortir : Faux </pre>

On souhaite maintenant résoudre un problème similaire : trouver trois indices $i < j < k$ tels que $L(i) + L(j) + L(k) = 0$ dans une liste ordonnée. En utilisant une approche similaire (une boucle externe sur i , puis l'algorithme ci-dessus sur le reste), quelle est la complexité obtenue ?

- ☐ $\Theta(n)$
☐ $\Theta(n^2)$
☐ $\Theta(n^3)$
☐ $\Theta(n \log n)$

Écrivez le sous-algorithme correspondant à la boucle externe qui appelle `deux font la paire` pour chaque valeur de i . L'algorithme doit retourner `Vrai` s'il existe trois éléments dont la somme est nulle, et `Faux` sinon.