

TD1 : Introduction et rappels

« Grand O »

Exercice 1.

Bon algorithme, bon programme

Pour résoudre un problème algorithmique, dont la variable d'entrée est notée n , on dispose de deux algorithmes : ALGO-BOF dont la complexité en temps est de $O(n^2)$ et ALGO-TOP dont la complexité en temps est de $O(n \log n)$.

- ALGO-BOF est codé par un super programmeur qui garantit que le programme fait exactement $2n^2$ opérations élémentaires, il le fait en plus tourner sur sa super machine, capable d'effectuer 20 000 000 000 d'opérations élémentaires par seconde.
- ALGO-TOP, quant-à-lui est codé par un programmeur moyen qui pense que le programme ne fait pas plus de $50n \log n$ opérations élémentaires et il le fait tourner en salle TP tout en regardant YouTube, sa machine n'effectuant alors que 1 000 000 000 d'opérations élémentaires par seconde.

 À partir de quelle valeur de n le programme codant ALGO-TOP est-il plus rapide que celui codant ALGO-BOF ?


Exercice 2.

FAQ

1. Est-il vraiment correct de dire « cet algorithme a une complexité en temps en au plus $O(n^2)$ » ?
2. A-t-on $2^{n+1} = O(2^n)$? Et $2^{2n} = O(2^n)$?
3. Montrer que si on a $f(n) = O(g(n))$ et $g(n) = O(h(n))$ alors on a aussi $f(n) = O(h(n))$.
4. Proposer deux fonctions f et g telles que $f(n) = O(g(n))$ et $g(n) = O(f(n))$. Si ce n'est pas le cas pour votre proposition, donner deux telles fonctions qui ne sont pas proportionnelles (c'est-à-dire, telles qu'il n'existe pas une constante c telle que $f(n) = c \cdot g(n)$).
5. Proposer deux fonctions f et g telles que $f(n) \neq O(g(n))$ et $g(n) \neq O(f(n))$.

Exercice 3.

O à la chaîne

-  Pour les paires de fonctions (f, g) suivantes, est-ce que $f(n) = O(g(n))$? Et $g(n) = O(f(n))$?
- | | | | |
|---|--|---|--|
| a. $f(n) = n + 100$ et $g(n) = n$ | b. $f(n) = \sqrt{n}$ et $g(n) = n^{2/3}$ | c. $f(n) = \sqrt{n}$ et $g(n) = (\log n)^3$ | |
| d. $f(n) = n^{1,01}$ et $g(n) = n \log^2 n$ | e. $f(n) = 2^n$ et $g(n) = 3^n$ | f. $f(n) = 10n + \log n$ et $g(n) = n + \log^2 n$ | |
| g. $f(n) = n^2 / \log n$ et $g(n) = n \log^2 n$ | h. $f(n) = n^5$ et $g(n) = 3^{\log n}$ | i. $f(n) = 2^n$ et $g(n) = n!$ | |

Exercice 4.

Restes du cours

Prouver les résultats suivants, qui sont donnés dans un lemme du cours :

1. Si $h = O(f)$ alors $f + h = O(f)$.
2. $O(f) \times O(g) = O(f \times g)$ (c-à-d, si $h_1 = O(f)$ et $h_2 = O(g)$ alors $h_1 \times h_2 = O(f \times g)$).

Structures de données

Exercice 5.

Pile ou file

On dispose d'une structure de données *liste-2-chainée* qui implémente une liste doublement chaînée. Pour une variable L de ce type, on a deux primitives $\text{deb}(L)$ et $\text{fin}(L)$ qui renvoient respectivement les nœuds initial et final de L . Et pour un nœud x de L , $\text{prec}(x)$ et $\text{suiv}(x)$ renvoient respectivement le nœud précédent et suivant de x , avec $\text{prec}(\text{deb}(L)) = \text{null}$ et $\text{suiv}(\text{fin}(L)) = \text{null}$. Il est possible de créer un nouveau nœud x avec l'appel `nouveauNoeud x`. Par défaut un tel nœud x vérifie $\text{prec}(x) = \text{null}$ et $\text{suiv}(x) = \text{null}$. Enfin, l'appel `supprimeNoeud x` libère la mémoire occupée par le nœud x .

Toutes ces opérations ont un temps d'exécution en $O(1)$.

1. En se servant de la structure liste-2-chainée, proposer une implémentation d'une pile, supportant les opérations Empiler et Dépiler devant s'exécuter en temps constant.
2. De même, proposer une implémentation d'une file, supportant les opérations Enfiler et Défiler devant s'exécuter en temps constant.

Analyses d'algo

Exercice 6.

Tri à bulles

Voici une version du classique TRI-A-BULLES :

Données : Un tableau T contenant n nombres réels.

Résultat : Le tableau T trié.

```


1 pour  $i$  de  $n-1$  à 1 faire
2   pour  $j$  de 0 à  $i-1$  faire
3     si  $T[j] > T[j+1]$  alors Échanger les contenus de  $T[j]$  et  $T[j+1]$ ;

```

1. Dérouler l'algorithme sur le tableau $T = [12, 3, 7, 0]$.
2. Calculer la complexité en temps de l'algorithme.
3. Prouver la validité de l'algorithme TRI-A-BULLES.

Exercice 7.

Combien de temps ?

 Établir la complexité en temps des trois algorithmes suivants (les opérations élémentaires ont été omises).

```

1 Algorithme : ALGO1( $n$ )
2 pour  $i$  de 0 à  $n-1$  faire
3   pour  $j$  de 0 à  $n-1$  faire
4     pour  $k$  de 0 à  $j$  faire
5       <op elem>
6 pour  $i$  de 0 à  $n-1$  faire
7   <op elem>

```

```

1 Algorithme : ALGO2( $n$ )
2 si  $n = 0$  alors return val;
3 ALGO2( $n-1$ );
4 <op elem>
5 ALGO2( $n-1$ );
6 <op elem>

```

```

1 Algorithme : ALGO3( $n$ )
2 <op elem>
3 tant que  $n > 1$  faire
4    $n \leftarrow n/3$ ;
5 <op elem>

```

Exercice 8.

Somme de 3

Étant donné un tableau T de taille n , on veut écrire un algorithme qui trouve trois indices distincts i, j et k de $\{0, \dots, n-1\}$ tels que $T[i] + T[j] = T[k]$, ou qui signale si trois tels indices n'existent pas.

1. Écrire un tel algorithme de complexité en temps $O(n^3)$.
2. On va essayer d'avoir un algorithme de complexité quadratique. Pour cela, on va traiter d'abord le sous problème suivant : étant donné un tableau S trié de taille n et un nombre x , écrire un algorithme de complexité linéaire en temps qui décide s'il existe deux indices distincts i et j tels que $T[i] + T[j] = x$ (on pourra commencer par comparer $T[0] + T[n-1]$ et x).
3. En déduire un algorithme de complexité en temps quadratique pour résoudre le problème initial.