

## TD1: Algorithmes, modèle de calcul, complexité

### « 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  $g$ , à valeurs strictement positives, 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  $h_1 \times h_2 = O(f \times g)$ ).

## Complexité algorithmique

#### Exercice 5.

*Puissance lente*

On considère les algorithmes ALGOIT et ALGOD&C du cours. On les appelle uniquement pour  $x = 2$ , et  $n$  quelconque.

1. Dans le modèle du cours (Word-RAM), quelle est alors la taille de l'entrée ? ALGOD&C est-il polynomial ?
2. On se place dans le modèle RAM, dans lequel les opérations élémentaires sont celles sur les bits. Quelle est alors la taille de l'entrée ? L'algorithme ALGOD&C fait-il un nombre d'appels récursifs polynomial en la taille de l'entrée ? L'algorithme ALGOIT est-il polynomial ?
3. Si on code  $n$  en unaire, quelle est alors la taille de l'entrée ? ALGOIT est-il polynomial ?


**Exercice 6.**

Complexité par l'exemple

1. Écrire un programme qui prend une valeur  $n$  en entrée et qui effectue un nombre d'opérations proportionnel à  $n^2$ , sans écrire  $n^2$  dans votre code !
2. Même question avec  $\log n$  au lieu de  $n^2$ , sans écrire  $\log n$  dans votre code !
3. Même question avec  $n!$ , sans écrire  $n!$  dans votre code !

**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        $\langle \text{op elem} \rangle$ 
6 pour  $i$  de 0 à  $n-1$  faire
7    $\langle \text{op elem} \rangle$ 

```

```

1 Algorithme : ALGO2( $n$ )
2 si  $n = 0$  alors return val;
3 ALGO2( $n-1$ );
4  $\langle \text{op elem} \rangle$ 
5 ALGO2( $n-1$ );
6  $\langle \text{op elem} \rangle$ 

```

```

1 Algorithme : ALGO3( $n$ )
2  $\langle \text{op elem} \rangle$ 
3 tant que  $n > 1$  faire
4    $n \leftarrow n/3$ ;
5  $\langle \text{op elem} \rangle$ 

```

## Algorithmes

**Exercice 8.**

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 9.**

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.