

## TD4 : Diviser pour régner

### Résolution de récurrences

**Exercice 1.**
*Combien de temps ?*

On veut établir une borne sur la complexité en temps des deux algorithmes suivants :

```

1 Algorithme : ALGO1( $n$ )
2 si  $n = 1$  :
3   | Retourner 0
4 sinon :
5   | pour  $i$  de 0 à  $n - 1$  :
6     |   | pour  $j$  de 0 à  $n - 1$  :
7       |   |   |  $\langle \text{op elem} \rangle$ 
8     |   ALGO1( $\lceil n/2 \rceil$ )
9   |   ALGO1( $\lceil n/2 \rceil$ )

```

```

1 Algorithme : ALGO2( $n$ )
2 si  $n = 1$  :
3   | Retourner 0
4 sinon :
5   | ALGO2( $\lceil n/2 \rceil$ )
6   |  $\langle \text{op elem} \rangle$ 
7   | ALGO2( $\lceil n/2 \rceil$ )
8   | pour  $i$  de 0 à  $n - 1$  :
9     |   |  $\langle \text{op elem} \rangle$ 
10  | ALGO2( $\lceil n/2 \rceil$ )

```

Pour chacun des deux algorithmes ALGO1 et ALGO2, effectuer le travail suivant :

1. écrire l'équation de récurrence respectée par la complexité  $t(n)$  de l'algorithme ;
2. résoudre la récurrence à l'aide du « Master Theorem ».

**Exercice 2.**
*Quel algorithme ?*

Pour résoudre un problème donné, vous avez le choix entre trois algorithmes : sur une entrée de taille  $n$ ,

- ALGOA divise le problème en 5 sous-problèmes de taille  $\lceil n/2 \rceil$  et combine les solutions en temps  $O(n)$  ;
- ALGOB divise le problème en 2 sous-problèmes de taille  $n - 1$  et combine les solutions en temps  $O(1)$  ;
- ALGOC divise le problème en 9 sous-problèmes de taille  $\lceil n/3 \rceil$  et combine les solutions en temps  $O(n^2)$ .



Quel algorithme choisissez-vous ? Justifier...

**Exercice 3.**
*Combien de lignes ?*


Combien de fois la ligne « Toujours pas fini... » est-elle affichée par le programme suivant, en fonction de l'entrée  $n$  ?

```

void affiche(int n) {
  if (n > 1) {
    cout << "Toujours pas fini..." << endl;
    affiche(n/2);
    affiche(n/2);
  }
}

```

### Algorithmes sur des tableaux

**Exercice 4.**
*Recherche d'un pic*

Un tableau  $T$  de  $n$  entiers tous distincts est un *tableau à pic* s'il existe un indice  $p$  avec  $0 \leq p \leq n - 1$  tel que  $T_{[0,p]}$  soit trié par ordre croissant et  $T_{[p,n]}$  soit trié par ordre décroissant. L'indice  $p$  est alors appelé le *pic* de  $T$ . Le but de l'exercice est d'écrire un algorithme qui retourne le pic d'un tableau à pic.

1. Donner un exemple de tableau à pic contenant 4 éléments.
2. Proposer un algorithme de complexité linéaire pour résoudre le problème.
3. En utilisant une stratégie « diviser pour régner », proposer un algorithme de meilleure complexité pour ce problème. On pourra se servir de  $T_{\lfloor n/2 \rfloor}$ , et évaluer la complexité de l'algorithme à l'aide du « Master Theorem ».

#### Exercice 5.

*Les  $p$  plus grands nombres triés*

Étant donné un tableau  $T$  contenant  $n$  entiers, on souhaite trouver les  $p$  plus grands nombres de  $T$ , dans l'ordre trié. Pour cela on peut appliquer plusieurs stratégies. Pour chacune des stratégies ci-dessous, évaluer une borne sur la complexité en temps de l'algorithme correspondant en fonction de  $n$  et  $p$ .

1. Trier le tableau  $T$  et retourner les  $p$  plus grands éléments.
2. Construire un tas sur  $T$  et extraire  $p$  fois l'élément maximal de ce tas (en le retirant à chaque fois).
3. Calculer le  $i^{\text{ème}}$  plus petit élément de  $T$ , pour chacun des entiers  $i = n, n-1, \dots, n-p+1$ , et retourner le tout.
4. Calculer le  $(n-p+1)^{\text{ème}}$  plus petit élément de  $T$ , récupérer les valeurs de  $T$  supérieures à celui-ci, puis les trier.

#### Exercice 6.

*Élément majoritaire*

On considère un tableau  $T$  de taille  $n$  contenant des entiers. On dit que l'élément  $p$  de  $T$  est *majoritaire dans  $T$*  si il est contenu dans strictement plus de  $n/2$  cases de  $T$ . Le but de l'exercice est d'écrire un algorithme qui retourne l'indice d'un élément majoritaire de  $T$  si celui-ci en contient un et  $-1$  sinon.

1. On veut d'abord un algorithme simple pour résoudre le problème.
  - (a) Écrire un tel algorithme : pour chaque indice  $i$  de  $T$  ( $0 \leq i \leq n-1$ ), compter le nombre d'éléments de  $T$  qui sont égaux à  $T[i]$ , puis conclure.
  - (b) Borner la complexité en temps de votre algorithme.
2. On veut maintenant élaborer un algorithme de type « diviser pour régner ».
  - (a) Montrer que  $T$  ne peut pas posséder d'élément majoritaire si ni  $T_{[0, \lfloor n/2 \rfloor]}$  ni  $T_{[\lfloor n/2 \rfloor, n]}$  n'en possèdent.
  - (b) En déduire un algorithme récursif pour le problème.
  - (c) Écrire l'équation de récurrence vérifiée par le temps de calcul de l'algorithme et la résoudre à l'aide du « Master Theorem ».

## Autres algorithmes

#### Exercice 7.

*Somme d'un arbre binaire complet*

On se donne un arbre binaire  $A$  complet, c'est-à-dire dont tous les niveaux sont complets. Le but de l'exercice est de proposer un algorithme  $\text{SOMAB}(\text{rac}(A))$  qui calcule la somme des clés de tous les nœuds de  $A$ . Pour un nœud  $x$  de  $A$ , on notera  $\text{AB}(x)$  le sous-arbre binaire de  $A$  enraciné en  $x$ ,  $\text{FilsG}(x)$  et  $\text{FilsD}(x)$  désignant respectivement les fils gauche et droit de  $x$ . Ainsi,  $\text{SOMAB}(x)$  retourne la somme des clés des nœuds de  $\text{AB}(x)$ .

1. Exprimer la valeur retournée par  $\text{SOMAB}(x)$  en fonction de celles retournées par  $\text{SOMAB}(\text{FilsG}(x))$  et  $\text{SOMAB}(\text{FilsD}(x))$ .
2. En déduire l'algorithme attendu.
3. Évaluer la complexité de votre problème en fonction de  $n$ , le nombre de nœuds de  $A$ , soit directement, soit en utilisant le « Master Theorem ».

#### Exercice 8.

*Exponentiation rapide*

Étant donné  $x$  et  $n$ , on souhaite calculer (rapidement !)  $x^n$ . On s'intéresse à la complexité en nombre de multiplications effectuées.

1. Donner un algorithme de complexité  $O(n)$  pour calculer  $x^n$ .

On cherche maintenant à décrire un algorithme plus rapide.

2. Exprimer  $x^n$  en fonction de  $x^{\lfloor n/2 \rfloor}$ , en distinguant le cas  $n$  pair du cas  $n$  impair.
3. En déduire un algorithme récursif de calcul de  $x^n$ .
4. Exprimer la complexité de l'algorithme obtenu, à l'aide du « *Master Theorem* ».
5. Quel problème peut poser le fait d'analyser la complexité simplement en comptant le nombre de multiplications ?

**Exercice 9.**

*Algorithme de Strassen (1969)*

Soit  $A = (a_{ij})_{1 \leq i, j \leq n}$  et  $B = (b_{ij})_{1 \leq i, j \leq n}$  deux matrices. Leur produit  $C = (c_{ij})_{1 \leq i, j \leq n}$  est défini par

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}$$

pour  $1 \leq i, j \leq n$ . On s'intéresse à la complexité, en nombres d'additions et multiplications, du calcul du produit de deux matrices. On suppose qu'une matrice  $M$  est représentée par un tableau  $M$  tel que  $M[i][j]$  est le coefficient  $m_{ij}$  de  $M$ .

1. Quelle est la complexité du calcul d'un coefficient  $c_{ij}$ , en appliquant la formule ci-dessus ? En déduire la complexité du calcul complet de la matrice  $C = A \times B$ .

Pour améliorer la complexité, on tente la stratégie « diviser pour régner » suivante : on découpe chaque matrice en blocs de  $n/2$  lignes et colonnes. On écrit donc  $A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}$  et  $B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$ , où  $A_{00}, A_{01}, \dots, B_{11}$  sont des matrices de  $n/2$  lignes et colonnes. On suppose à partir de maintenant que  $n$  est une puissance de 2.

2. On découpe le produit  $C = A \times B$  de la même façon, sous la forme  $C = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$ .
  - (a) Exprimer chacune des quatre matrices  $C_{00}, C_{01}, C_{10}$  et  $C_{11}$  en fonction des matrices  $A_{00}, \dots, B_{11}$ .
  - (b) En déduire un algorithme de type « diviser pour régner » pour effectuer le calcul du produit  $C = A \times B$ .
  - (c) Analyser la complexité de l'algorithme obtenu.
3. Comme pour l'algorithme de Karatsuba de multiplication des entiers, on peut économiser un appel récursif. Pour cela, on définit les matrices suivantes :

$$\begin{aligned} P_1 &= A_{00} \times (B_{01} - B_{11}) & P_5 &= (A_{00} + A_{11}) \times (B_{00} + B_{11}) \\ P_2 &= (A_{00} + A_{01}) \times B_{11} & P_6 &= (A_{01} - A_{11}) \times (B_{10} + B_{11}) \\ P_3 &= (A_{10} + A_{11}) \times B_{00} & P_7 &= (A_{00} - A_{10}) \times (B_{00} + B_{01}) \\ P_4 &= A_{11} \times (B_{10} - B_{00}) \end{aligned}$$

- (a) Montrer que  $C = \begin{pmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{pmatrix}$ . On pourra montrer que  $C_{00} = P_5 + P_4 - P_2 + P_6$  et admettre les autres égalités. En déduire un nouvel algorithme de type « diviser pour régner » pour effectuer le calcul du produit  $C = A \times B$ .
- (b) Montrer que le temps de calcul de ce nouvel algorithme vérifie  $T(n) = 7T(n/2) + O(n^2)$ .
- (c) Résoudre la récurrence à l'aide du « *Master Theorem* ».