TD 5. Diviser pour régner

Exercice 1. *Le maître*

1. Analyser la complexité des algorithmes suivants, dans lesquels `<op_elem>` désigne une opération élémentaire de complexité $O(1)$.

ALGO1(n) :

```
Si n = 1 : Renvoyer 0
Pour i de 0 à n−1:
    Pour j de 0 à n−1:
        <op elem>
ALGO1(⌈n/2⌉)
ALGO1(⌊n/2⌋)
```

ALGO2(n) :

```
Si n = 1: Renvoyer 0
ALGO2(⌈n/2⌉)
<op elem>
ALGO2(⌊n/2⌋)
Pour i de 0 à n−1 : <op elem>
ALGO2(⌈n/2⌉)
```

2. Pour résoudre un problème donné, on dispose de trois algorithmes. Calculer la complexité de chacun d'après sa description, et comparer les résultats. Sur une entrée de taille n,

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

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

```c
void affiche(int n) {
  if (n > 1) {
    printf("Toujours pas fini...\n");
    affiche(n/2);
    affiche(n/2);
  }
}
```

Exercice 2. *Température extrême*

On suppose qu'au cours d'une journée, les températures commencent par croître (strictement) jusqu'à la température maximale, puis décroissent (strictement) en fin de journée. On dispose d'un tableau des températures mesurées (disons toutes les minutes), et on souhaite savoir quelle a été la température maximale.

1. Proposer un algorithme de complexité linéaire pour résoudre le problème.
2. En utilisant une stratégie « diviser pour régner », proposer un algorithme de meilleure complexité pour ce problème. *On pourra se servir de* $T_{[\lfloor n/2 \rfloor]}$.

Exercice 3. *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 s'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.
   - **i.** 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.
   - **ii.** Borner la complexité en temps de votre algorithme.
2. On veut maintenant élaborer un algorithme de type « diviser pour régner ».
   - **i.** 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.
   - **ii.** En déduire un algorithme récursif pour le problème.
   - **iii.** Analyser la complexité de l'algorithme obtenu.

Exercice 4. *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.
5. Quel problème peut poser le fait d'analyser la complexité simplement en comptant le nombre de multiplications ?

Exercice 5. *Transformée de Fourier rapide*

Soit $p \in \mathbb{C}[x]$ un polynôme de degré $< d$ et $\omega = e^{2i\pi/d}$. La *transformée de Fourier discrète* de p est le vecteur $\text{DFT}_\omega^d(p) = (p(\omega^0), p(\omega^1), \ldots, p(\omega^{d-1})) \in \mathbb{C}^d$. On suppose que p est représenté par le vecteur de ses d coefficients, et les opérations élémentaires sont les opérations sur les complexes.

1.
   - **i.** Décrire un algorithme qui étant donné p et $\alpha \in \mathbb{C}$ calcule $p(\alpha)$. Analyser sa complexité.
   - **ii.** En déduire un algorithme qui étant donné p et $\omega = e^{2i\pi/d}$ calcule $\text{DFT}_\omega^d(p)$. Analyser sa complexité.

On décrit maintenant une stratégie « diviser pour régner » pour calculer $\mathrm{DFT}_\omega^d(p)$ plus rapidement. *On suppose que d est une puissance de* 2.

2. Si $p = \sum_{i=0}^{d-1} p_i x^i$, on définit $p^{(0)} = \sum_{i=0}^{d/2} p_{2i} x^i$ et $p^{(1)} = \sum_{i=0}^{d/2} p_{2i+1} x^i$.

- **i.** Exprimer p à l'aide de $p^{(0)}$ et $p^{(1)}$. Combien coûte le calcul de $p^{(0)}$ et $p^{(1)}$ à partir de p ?
- **ii.** Exprimer les coefficients de $\mathrm{DFT}_\omega^d(p)$ en fonction de ceux de $\mathrm{DFT}_{\omega^2}^{d/2}(p^{(0)})$ et $\mathrm{DFT}_{\omega^2}^{d/2}(p^{(1)})$. *Indication. Justifier et utiliser le fait que $\omega^{2k} = \omega^{2k-d}$ pour $k \geq d/2$.*
- **iii.** En déduire un algorithme de type « diviser pour régner » pour calculer $\mathrm{DFT}_\omega^d(p)$ et analyser sa complexité.

Exercice 6. *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.

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 quatre *blocs* : $A = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}$ et $B = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$, où A_{00}, A_{01}, ..., 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 $C = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}$.

- **i.** Exprimer chacune des quatre matrices C_{00}, C_{01}, C_{10} et C_{11} en fonction des matrices A_{00}, ..., B_{11}.
- **ii.** En déduire un algorithme de type « diviser pour régner » pour effectuer le calcul du produit $C = A \times B$.
- **iii.** 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 :

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

i. 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.*

ii. En déduire un nouvel algorithme de type « diviser pour régner » pour effectuer le calcul du produit $C = A \times B$.

iii. Analyser la complexité obtenue.

.

Exercice 7. *Plus proche paire de points*

Étant donnés un ensemble $S = \{p_1, \dots, p_n\}$ de points dans le plan, représentés par leurs coordonnées $p_i = (x_i, y_i)$, on cherche la distance (euclidienne) minimale entre deux points de ces points. *On suppose que n est une puissance de 2, que toutes les abscisses sont distinctes, ainsi que toutes les ordonnées, et que le calcul de la distance euclidienne s'effectue en temps $O(1)$.*

1. Donner un algorithme de complexité $O(n^2)$ pour ce problème.

On adopte une stratégie « diviser pour régner » : 1. couper l'ensemble de points en deux sous-ensembles G et D selon un axe vertical Δ ; 2. chercher les deux points les plus proches dans G, de même dans D, et noter d la distance minimale obtenue ; 3. chercher s'il existe deux points $p_g \in G$ et $p_d \in D$ à distance $< d$.

2. Montrer qu'à l'étape 3., on peut se restreindre à un sous-ensemble S_Δ des points contenus dans une bande verticale autour de Δ.

3. Montrer que tout carré de côté d, parallèle aux axes, contient au plus 4 points de G (resp. de D).

4. En déduire qu'à l'étape 3., il suffit de calculer, pour chaque point de S_Δ, sa distance à un nombre constant de points de S_Δ.

5. Écrire l'algorithme correspondant à la stratégie ci-dessus. *Il est recommandé de calculer deux tableaux contenant les points : l'un les stocke triés par abscisses croissantes, et l'autre par ordonnées croissantes.*

6. Analyser la complexité de l'algorithme.