TD3: Diviser pour Régner

Résolution de récurrences

Exercice 1. **Exo TD/TP :** *Cherche l'algo, cherche !*

1. Écrire deux algorithmes de recherche d'un élément dans un tableau **trié** de taille n : un algorithme itératif de complexité linéaire en n et une recherche dichotomique.
2. Prouver les complexités de vos algorithmes. Pour le second algorithme, on pourra établir une équation de récurrence pour la complexité puis utiliser le 'Master Theorem'.
3. Implémenter vos deux algorithmes et comparer leur temps de calcul respectifs en faisant varier n (on génerera au hasard des tableaux de taille n, ainsi que l'élément à rechercher).

Exercice 2. **Exo TD :** *Combien de temps ?*

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

```
Algorithme : ALGO1(n)
si n = 1 alors
    Retourner 0;
sinon
    pour i de 0 à n−1 faire
        pour j de 0 à n−1 faire
            <op elem>
    ALGO1(⌈n/2⌉);
    ALGO1(⌈n/2⌉);
```

```
Algorithme : ALGO2(n)
si n = 1 alors
    Retourner 0;
sinon
    ALGO2(⌈n/2⌉);
    <op elem>
    ALGO2(⌈n/2⌉);
    pour i de 0 à n−1 faire
        <op elem>
    ALGO2(⌈n/2⌉);
```

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 3. **Exo TD :** *Quel algorithme ?*

Pour résoudre un problème donné, vous avez le choix entre trois algorithmes : sur un 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 4. **Exo TD :** *Combien de lignes ?*

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

```
def affiche(n):
  if n > 1:
    print("Toujours pas fini...")
    affiche(n/2)
    affiche(n/2)
```

Algorithmes sur des tableaux

Exercice 5. **Exo TP :** *Tris*

✎ Implémenter les algorithmes TRIBULLES et TRIFUSION. Comparer leur temps d'exécution en faisant varier la taille du tableau d'entrée.

Exercice 6. **Exo TD :** *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 $T[p,n-1]$ 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 7. **Exo TP :** *En rang !*

✎ Implémenter les trois algorithmes de calcul de rang vu en cours (respectivement en $O(n^2)$, $O(n \log n)$ et $O(n)$). Comparer leur temps d'exécution.
Pour le rang linéaire, faire afficher le nombre de tirages aléatoires effectués à chaque appel récursif et vérifier qu'en moyenne, cette valeur est autour de 2.

Exercice 8. **Exo TD :** *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 $T[0,\lfloor n/2 \rfloor - 1]$ ni $T[\lfloor n/2 \rfloor, n-1]$ 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* ».

Exercice culturel : Algorithme de Strassen

Exercice 9. **Exo TD :** *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}, ..., 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 $C = \left(\begin{smallmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{smallmatrix}\right)$.

   (a) Exprimer chacune des quatre matrices C_{00}, C_{01}, C_{10} et C_{11} en fonction des matrices A_{00}, ..., 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 :

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

   (a) Montrer que $C = \left(\begin{smallmatrix} P_5+P_4-P_2+P_6 & P_1+P_2 \\ P_3+P_4 & P_1+P_5-P_3-P_7 \end{smallmatrix}\right)$. *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* ».