



Session : 1

Date : 9 mai 2019

L2 Informatique et Maths-Info

UE : Algorithmique et complexité (HLIN401)

Durée de l'épreuve : 2h00

Documents autorisés : aucun

Matériel utilisé : aucun

Nombre de pages : 3

Les réponses doivent être justifiées (sauf mention contraire) et rédigées correctement. Le sujet est constitué de **cinq** exercices indépendants. Toute question non résolue peut être admise dans les questions suivantes. Le barème fourni est indicatif et susceptible de modification.

(2 pts) Exercice 1.


« Grand O »

Pour les paires de fonctions  $f$ ,  $g$  suivantes, déterminer si  $f(n) = O(g(n))$  et si  $g(n) = O(f(n))$ .

1.  $f(n) = 4n^2 \log^3(n) - 12n^2 + 4$  et  $g(n) = (3n \log(n) + 2n - 1) \times (n \log^2(n) - n + \log(n))$ .
2.  $f(n) = 2^{n \log n}$  et  $g(n) = n^{\log^2(n)}$ .

(2 pts) Exercice 2.

Complexité des algorithmes

-  Déterminer la complexité des deux algorithmes ci-dessous, en supposant que chaque opération élémentaire ( $\langle \text{op elem} \rangle$ ) prend un temps  $O(1)$ . Justifier proprement les calculs.

```
1 Algorithme : ALGO1(n)
2 pour i = n - 1 à 0 faire
3   j ← n;
4   tant que j > 1 faire
5     <op elem>;
6     j ← j/2;
7 pour k = 1 à n faire
8   <op elem>;
```

```
1 Algorithme : ALGO2(n)
2 si n > 1 alors
3   <op elem>;
4   ALGO2(n/3);
5   pour t = 1 à n/2 faire
6     <op elem>;
7   ALGO2(n/3);
8   <op elem>;
```

(2 pts) Exercice 3.

Le TriFUSION

Cet exercice a pour objet de redémontrer certains résultats du cours sur TriFUSION. On rappelle que pour trier un tableau  $T$  de taille  $n$ , le TriFUSION coupe  $T$  en deux sous-tableaux  $T[0, \lfloor n/2 \rfloor - 1]$  et  $T[\lfloor n/2 \rfloor, n - 1]$ , trie ces deux sous-tableaux de manière récursive, puis fusionne les deux tableaux triés.

1. Écrire l'algorithme FUSION qui prend en entrée deux tableaux triés  $T_1$  (de taille  $n_1$ ) et  $T_2$  (de taille  $n_2$ ) et renvoie un tableau trié (de taille  $n_1 + n_2$ ) qui contient tous les éléments de  $T_1$  et  $T_2$ . Analyser sa complexité.
2. Écrire l'équation de récurrence vérifiée par la complexité de l'algorithme TriFUSION et la résoudre à l'aide du « Master Theorem ».

(6 pts) Exercice 4.

*Tinder musical*

Une application cherche à faire se rencontrer ses membres en fonction de leurs goûts musicaux. Pour cela, chaque membre doit classer par ordre de préférence une liste d'artistes<sup>1</sup>. On dit que deux membres, Abdel et Chloé, ont des goûts musicaux proches lorsque qu'il y a peu d'inversions dans leurs classements : une inversion est une paire d'artistes  $(X, Y)$  telle qu'Abdel préfère  $X$  à  $Y$  et Chloé préfère  $Y$  à  $X$ . On cherche donc à compter le nombre d'inversions dans les classements d'Abdel et Chloé.

1. Compter le nombre d'inversions les classements suivants :

**Abdel** : Kaaris, Lady Gaga, Booba, Maître Gims, Kendji Girac ;

**Chloé** : Booba, Lady Gaga, Maître Gims, Kaaris, Kendji Girac.

2. Proposer un algorithme naïf qui résout le problème. Quelle est sa complexité ?

On cherche maintenant un algorithme de type « diviser-pour-régner ». Pour cela, on numérote les artistes dans l'ordre de préférence d'Abdel : l'artiste 0 est son préféré (Kaaris dans l'exemple), l'artiste  $n-1$  est celui qu'il aime le moins. On voit l'ordre de Chloé comme un tableau  $C$  d'entiers :  $C[i] = j$  signifie que l'artiste classé  $i$  par Chloé est celui classé  $j$  par Abdel (dans l'exemple,  $C = [2, 1, 3, 0, 4]$ ). Le nombre d'inversions est alors le nombre de paires d'indices  $(i, j)$  tel que  $i < j$  et  $C[i] > C[j]$ .

On coupe le classement de Chloé en deux sous-classements :  $C_{\text{sup}} = C[0, \lfloor n/2 \rfloor - 1]$  contient ses artistes préférés, et  $C_{\text{inf}} = C[\lfloor n/2 \rfloor, n-1]$  les autres artistes (dans l'exemple,  $C_{\text{sup}} = [2, 1]$  et  $C_{\text{inf}} = [3, 0, 4]$ ). On distingue alors les inversions  $(i, j)$  en trois catégories : celles de  $C_{\text{sup}}$  ( $i, j < \lfloor n/2 \rfloor$ ), celles de  $C_{\text{inf}}$  ( $i, j \geq \lfloor n/2 \rfloor$ ), et les inversions mixtes ( $i < \lfloor n/2 \rfloor \leq j$ ).

3. L'objectif de cette question est d'analyser l'algorithme suivant qui compte les inversions mixtes. **On suppose que  $C_{\text{sup}}$  et  $C_{\text{inf}}$  sont triés par ordre croissant.**

```
1 Algorithme : INVERSIONSMIXTES( $C_{\text{sup}}, C_{\text{inf}}$ )
2  $i \leftarrow 0$ ;  $j \leftarrow 0$ ;
3  $n_{\text{sup}} \leftarrow \text{taille}(C_{\text{sup}})$ ;  $n_{\text{inf}} \leftarrow \text{taille}(C_{\text{inf}})$ ;
4  $c \leftarrow 0$ ;
5 tant que  $i < n_{\text{sup}}$  et  $j < n_{\text{inf}}$  faire
6   si  $C_{\text{sup}}[i] > C_{\text{inf}}[j]$  alors
7      $c \leftarrow c + (n_{\text{sup}} - i)$ ;
8      $j \leftarrow j + 1$ ;
9   sinon
10     $i \leftarrow i + 1$ ;
11 retourner  $c$ 
```

- (a) Analyser la complexité de l'algorithme, exprimée en fonction de  $n = n_{\text{inf}} + n_{\text{sup}}$ .
  - (b) Montrer que si  $(i, j)$  est une inversion mixte, alors pour tout  $i' \in \{i+1, \dots, \lfloor n/2 \rfloor\}$ ,  $(i', j)$  est également une inversion mixte.
  - (c) Démontrer à l'aide de la question précédente l'invariant suivant : « à chaque entrée dans la boucle *tant que*, la variable  $c$  contient le nombre d'inversions mixtes  $(i', j')$  avec  $j' < j$  ».
  - (d) En déduire que INVERSIONSMIXTES compte bien le nombre d'inversions mixtes.
- 4.(a) Écrire un algorithme de type « diviser-pour-régner » qui prend en entrée le tableau  $C$  et compte le nombre total d'inversions. *Indication. L'algorithme est une modification du TRIFUSION utilisant INVERSIONSMIXTES.*
    - (b) Montrer que sa complexité est identique à celle du TRIFUSION.
    - (c) Démontrer sa correction.

---

1. Le classement est un ordre total.

(8 pts) Exercice 5.

Arbre binaire de recherche optimal

On souhaite stocker un dictionnaire  $D$  (une liste des mots) dans un arbre binaire de recherche, en utilisant l'ordre alphabétique pour comparer deux mots : par exemple `informatique > chimie`.

1. Exprimer la complexité, dans le pire cas, d'un appel à RECHERCHER dans un ABR représentant un dictionnaire, en fonction du nombre  $n$  de mots. Quelle est cette complexité si l'ABR est équilibré ?

On dispose pour chaque mot  $m$  d'une *fréquence d'apparition*  $f_m$ . Pour un ABR  $A$  contenant un dictionnaire  $D$ , on définit sa *complexité moyenne*  $C_A$  par  $C_A = \sum_{m \in D} f_m(1 + h(m))$  où  $h(m)$  est la hauteur du mot  $m$  dans l'ABR  $A$  :  $C_A$  représente la *complexité moyenne* de l'algorithme RECHERCHER pour l'arbre  $A$ . On veut trouver, étant donné  $D$ , l'ABR  $A$  qui minimise la complexité moyenne  $C_A$ .

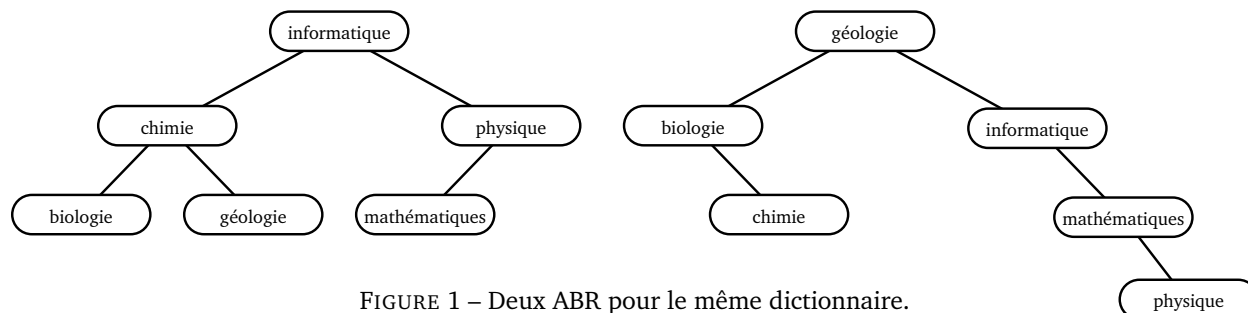


FIGURE 1 – Deux ABR pour le même dictionnaire.

Avec les fréquences 5% (biologie), 3% (chimie), 10% (géologie), 21% (informatique), 15% (mathématiques) et 2% (physique), l'ABR de gauche de la figure 1 a une complexité moyenne 1,21 =  $3 \times 5\% + 2 \times 3\% + 3 \times 10\% + 1 \times 21\% + 2 \times 2\% + 3 \times 15\%$ , car « biologie » est à hauteur 2, etc. et celui de droite 1,24.

2. On utilise l'algorithme glouton suivant pour créer l'ABR  $A$  : on insère, à partir d'un arbre vide, les mots par ordre de fréquence décroissante.
  - (a) Appliquer l'algorithme sur l'ensemble de mots de l'exemple et calculer la complexité moyenne obtenue.
  - (b) Quelle est la complexité de cet algorithme glouton ?
  - (c) Montrer que l'algorithme glouton n'est pas optimal. On peut construire un exemple avec trois mots pour lequel l'algorithme glouton n'est pas optimal.

On va maintenant décrire un algorithme de programmation dynamique pour calculer l'ABR optimal, c'est-à-dire qui minimise la complexité moyenne  $C_A$ . On suppose que les mots sont numérotés de 0 à  $n-1$  par ordre alphabétique :  $D[i]$  est le  $i^{\text{ème}}$  mot du dictionnaire  $D$  et  $f[i]$  est sa fréquence. On note  $C(i, k)$  la complexité moyenne optimale pour un ABR qui contient les mots  $D[i]$  à  $D[k]$  ( $C(i, k) = 0$  si  $i > k$ ). On cherche donc à calculer  $C(0, n-1)$ .

3. On veut montrer que  $C(i, k) = \sum_{j=i}^k f[j] + \min_{i \leq r \leq k} (C(i, r-1) + C(r+1, k))$  si  $i \leq k$ . Pour cela, on considère un ABR  $A$  contenant les mots  $D[i]$  à  $D[k]$ , dont la racine est le mot  $D[r]$  (avec  $i \leq r \leq k$ ).
  - (a) Quels sont les mots contenus dans le sous-arbre gauche  $\text{saG}(A)$  de  $A$  ? Et ceux contenus dans son sous-arbre droit  $\text{saD}(A)$  ?
  - (b) Montrer que  $C_A = \sum_{j=i}^k f[j] + C_{\text{saG}(A)} + C_{\text{saD}(A)}$ .
  - (c) Démontrer l'équation de récurrence.
4. On définit  $F(i, k) = \sum_{j=i}^k f[j]$ . Décrire un algorithme de programmation dynamique de complexité  $O(n^2)$  pour calculer tous les  $F(i, k)$ ,  $0 \leq i \leq k < n$ .
5. Dédire de ce qui précède un algorithme pour calculer  $C(0, n-1)$ . Quelle est sa complexité ?
6. (bonus) Décrire un algorithme permettant de calculer l'ABR optimal (de complexité moyenne  $C(0, n-1)$ ). Vous pouvez indiquer des modifications à apporter à l'algorithme précédent pour permettre la reconstruction de la solution a posteriori.