Université d'El Oued
Département d'informatique

Algorithmique avancée et Complexité
Corrigé – Examen du 20 Janvier 2026

Exercice 1 – Notation asymptotique (3 points)

1) Comparaison des fonctions (2 pts)

Le terme dominant de $f(n)$ est n^3 et celui $g(n)$ est $n^2 \log n$.

Comme n^3 croît plus vite que $n^2 \log n$, on a

$f(n) \in \Theta(n^3)$, g(n) $\in \Theta(n^2 \log n)$, et

$f(n) \in \Omega(g(n))$ et $g(n) \in \mathrm{O}(f(n))$

2) Affirmation vraie ou fausse (1 pt)

$$n \log n \in O(n^{1.5})$$

✅ **Vrai**

Justification :

$$\lim_{n\to+\infty} \frac{n \log n}{n^{1.5}} = \lim_{n\to+\infty} \frac{\log n}{\sqrt{n}} = 0$$

Donc $n \log n$ croît strictement moins vite que $n^{1.5}$, d'où : $n \log n \in O(n^{1.5})$.

Exercice 2 – Calcul de x^n : algorithmes et complexité (3 points)

1) Complexité temporelle (1 pt)

- **Algorithme 1 (itératif)**
  → Une boucle de n itérations

$$\boxed{O(n)}$$

- **Algorithme 2 (récursif linéaire)**
  → Un appel récursif par décrément de n

$$\boxed{O(n)}$$

- **Algorithme 3 (exponentiation rapide)**
  → Division du problème par 2

$$\boxed{O(\log n)}$$

2) Équation de récurrence (1 pt)

Pour l'algorithme 3 :

$$T(n) = T\left(\frac{n}{2}\right) + O(1)$$

avec condition initiale :

$$T(0) = O(1)$$

3) Comparaison des algorithmes (1 pt)

- Algorithmes 1 et 2 : complexité linéaire $O(n)$
- Algorithme 3 : complexité logarithmique $O(\log n)$

Conclusion : L'algorithme 3 est le **plus efficace** pour de grandes valeurs de n.

Exercice 3 – Tri rapide (QuickSort) (6 points)

Tableau initial : $A = [15, 9, 17, 3, 12, 27, 6, 20]$

Pivot = **premier élément**

1) Application pas à pas (2 pts)

Étape 1
Pivot = 15
Partition :

- Gauche : [9, 3, 12, 6]
- Droite : [17, 27, 20]

Étape 2 (Sous-tableau gauche [9, 3, 12, 6])
Pivot = 9
Partition :

- Gauche : [3, 6]
- Droite : [12]

Étape 3 (Sous-tableau [3, 6])
Pivot = 3
Partition :

- Gauche : []
- Droite : [6]

Étape 4 (Sous-tableau droit initial)
Pivot = 17
Partition :

- Gauche : []

- Droite : [27, 20]

Étape 5 (Sous-tableau [27, 20])
Pivot = 27
Partition :

- Gauche : [20]
- Droite : []

Résultat final : [3, 6, 9, 12, 15, 17, 20, 27]

2) Pseudo-code de QuickSort (2 pts)

```
QuickSort(A, low, high):
    if low < high:
        p ← Partition(A, low, high)
        QuickSort(A, low, p - 1)
        QuickSort(A, p + 1, high)
```

3) Complexité temporelle (2 pts)

- **Meilleur cas** (partition équilibrée) : $O(n \log n)$
- **Pire cas** (tableau déjà trié, pivot extrême) : $O(n^2)$

Exercice 4 – Programmation dynamique : Alignement de chaînes (8 points)

1) Sous-problème et principe (1 pt)

On définit :

$$\boldsymbol{DP[i][j]} \textbf{ = coût minimal pour aligner } \boldsymbol{X[1..i]} \textbf{ et } \boldsymbol{Y[1..j]}$$

La programmation dynamique est adaptée car :

- les sous-problèmes se recouvrent
- le problème possède une structure optimale

2) Relation de récurrence (2 pts)

$$DP[i][j] = min\begin{cases} DP[i-1][j] \ + \ gap & \text{(Suppression)} \\ DP[i][j-1] + gap & \text{(insertion)} \\ DP[i-1][j-1] + cost(x_i, y_j) & \text{(match/substitution)} \end{cases}$$

Conditions initiales :

$$DP[0][j] = j \cdot gap, \qquad DP[i][0] = i \cdot gap$$

3) Diagramme de programmation dynamique (2 pts)

i\j	0	G	C	A	T	G	C	U
0	0	1	2	3	4	5	6	7
G	1	0	1	2	3	4	5	6
A	2	1	1	1	2	3	4	5
T	3	2	2	2	1	2	3	4
T	4	3	3	3	2	2	3	4
A	5	4	4	3	3	3	3	4
C	6	5	4	4	4	4	3	4
A	7	6	5	4	5	5	4	4

→ Coût minimal d'alignement :

$$\boxed{DP[7][7] + 4}$$

4) Alignement optimal (1 pt)

Par backtracking à partir de $DP[m][n]$, on obtient un alignement optimal possible, par exemple (4 substitution):

```
G A T T A C A
G C A T G C U
```

5) Implémentation Top-Down avec mémoïsation (2 pts)

```
Align(i, j):
    if i = 0: return j
    if j = 0: return i
    if memo[i][j] existe: return memo[i][j]

    cost ← 0 si X[i] = Y[j], sinon 1

    memo[i][j] ← min(
        Align(i-1, j-1) + cost,
        Align(i-1, j) + 1,
        Align(i, j-1) + 1
    )

    return memo[i][j]
```