Ecole Supérieure
d'Informatique et du Numérique
COLLEGE OF ENGINEERING & ARCHITECTURE

Fiche Révision Algo

Algorithmique Avancée - Préparation Partiels mi-semestre
3e année Cybersécurité - École Supérieure d'Informatique et du Numérique (ESIN)
Collège d'Ingénierie & d'Architecture (CIA)

Étudiant : HATHOUTI Mohammed taha

Filière : Cybersécurité

Année : 2025/2026

Enseignants : M.BAKHOUYA

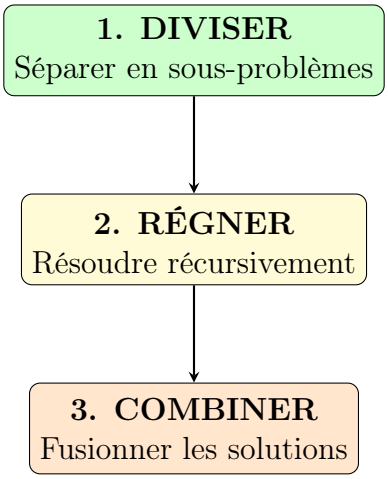
Date : October 26, 2025

Contents

1 Le Paradigme Diviser pour Régner

1.1 Principe général

Le paradigme **"Diviser pour Régner"** est une méthode de conception d'algorithmes qui procède en trois étapes :



1.2 Formule générale de récurrence

Pour un algorithme Diviser pour Régner, le temps d'exécution $T(n)$ suit généralement cette récurrence :

Récurrence générale

$$T(n) = \begin{cases} \Theta(1) & \text{si } n \leq c \\ a \cdot T\left(\frac{n}{b}\right) + D(n) + C(n) & \text{sinon} \end{cases}$$

Où :

- a = nombre de sous-problèmes
- b = facteur de division
- $D(n)$ = coût de la division
- $C(n)$ = coût de la combinaison

1.3 Exemple classique : Tri Fusion

1.3.1 Principe du Tri Fusion

```
TRI-FUSION(A, debut, fin):
    si debut < fin alors
        milieu = (debut + fin) / 2

        # 1. DIVISER
        TRI-FUSION(A, debut, milieu)       # Trier gauche
```

```
        TRI-FUSION(A, milieu+1, fin)       # Trier droite

        # 2. COMBINER
        FUSIONNER(A, debut, milieu, fin)   # Fusionner les deux
           parties
```

Listing 1: Algorithme du Tri Fusion

1.3.2 Analyse de complexité

- **Diviser :** Calculer le milieu $\Rightarrow D(n) = O(1)$
- **Régner :** Deux appels récursifs sur $n/2 \Rightarrow 2T(n/2)$
- **Combiner :** Fusionner deux listes triées $\Rightarrow C(n) = \Theta(n)$

Récurrence du Tri Fusion

$$T(n) = \begin{cases} \Theta(1) & \text{si } n = 1 \\ 2T(n/2) + \Theta(n) & \text{si } n > 1 \end{cases}$$

Résolution avec Master Theorem : $\boxed{T(n) = \Theta(n \log n)}$

Pourquoi Tri Fusion est efficace ?

- Complexité $O(n \log n)$ dans **tous les cas** (meilleur, moyen, pire)
- Beaucoup plus rapide que Tri par Insertion ($O(n^2)$) pour grands n
- Stable (préserve l'ordre des éléments égaux)

2 Les Notations Asymptotiques

2.1 Introduction

Les notations asymptotiques permettent de décrire le **comportement d'une fonction** quand n tend vers l'infini, en ignorant les constantes et les termes de faible ordre.

Pourquoi "asymptotique" ?

Le mot "asymptotique" signifie qu'on s'intéresse au comportement pour les **grandes valeurs de** n. Les petites valeurs et les constantes deviennent négligeables.

2.2 Notation O (Big-O) - Borne Supérieure

Définition mathématique

$$f(n) = O(g(n)) \Leftrightarrow \exists c > 0, \exists n_0 \text{ tels que :}$$
$$0 \leq f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$$

Signification : $g(n)$ est une **borne supérieure asymptotique** de $f(n)$. On dit que "f croît au plus aussi vite que g".

2.2.1 Exemples

- $3n^2 + 5n + 2 = O(n^2)$ ✓
- $2n + 100 = O(n)$ ✓
- $n \log n = O(n^2)$ ✓
- $n^2 = O(n)$ ✗ (faux car n^2 croît plus vite que n)

Règle pratique

Pour un polynôme, la complexité Big-O est déterminée par le **terme de plus haut degré**, en enlevant son coefficient :

$$5n^3 + 2n^2 - 100n + 50 = O(n^3)$$

2.3 Notation Ω (Omega) - Borne Inférieure

Définition mathématique

$$f(n) = \Omega(g(n)) \Leftrightarrow \exists c > 0, \exists n_0 \text{ tels que :}$$
$$0 \leq c \cdot g(n) \leq f(n) \quad \forall n \geq n_0$$

Signification : $g(n)$ est une **borne inférieure asymptotique** de $f(n)$. On dit que "f croît au moins aussi vite que g".

2.3.1 Exemples

- $n^2 = \Omega(n)$ ✓
- $n \log n = \Omega(n)$ ✓
- $n = \Omega(n^2)$ ✗ (faux)

2.4 Notation Θ (Theta) - Borne Exacte

Définition mathématique

$$f(n) = \Theta(g(n)) \Leftrightarrow \exists c_1, c_2 > 0, \exists n_0 \text{ tels que :}$$
$$0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \quad \forall n \geq n_0$$

Signification : f et g ont le **même ordre de grandeur**. On dit que "f croît exactement comme g".

Relation entre les notations

$$\boxed{f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ ET } f(n) = \Omega(g(n))}$$

2.4.1 Exemple de preuve

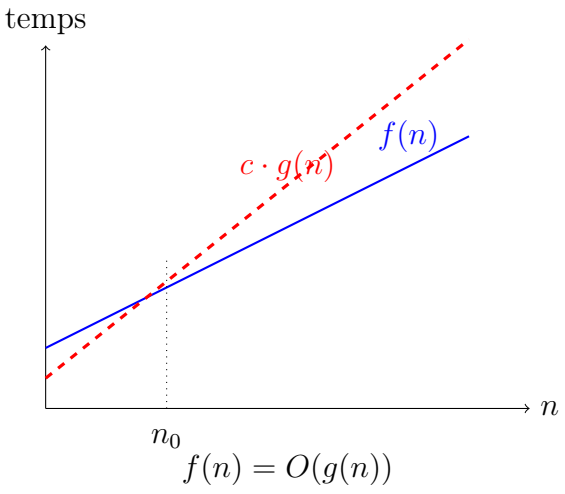
Montrons que : $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

Preuve :

1. Il faut trouver $c_1, c_2 > 0$ et n_0 tels que :
$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$
2. Pour $n \geq 7$, on a $-3n \geq -\frac{3}{7}n^2$, donc :
$$\frac{1}{2}n^2 - 3n \geq \frac{1}{2}n^2 - \frac{3}{7}n^2 = \frac{1}{14}n^2$$
3. De plus, pour $n \geq 1$:
$$\frac{1}{2}n^2 - 3n \leq \frac{1}{2}n^2$$
4. On peut donc prendre $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$, $n_0 = 7$.

Donc $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

2.5 Comparaison visuelle



$f(n) = O(g(n))$

3 Échelle de Complexité

3.1 Hiérarchie des complexités

Voici les complexités les plus courantes, classées de la plus rapide (haut) à la plus lente (bas) :

Échelle de complexité (à connaître PAR CŒUR)

$$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

3.2 Détails de chaque complexité

Complexité	Nom	Exemple
$O(1)$	Constante	Accès à un élément de tableau `A[i]`
$O(\log n)$	Logarithmique	Recherche dichotomique, arbres équilibrés
$O(n)$	Linéaire	Parcours d'un tableau, recherche linéaire
$O(n \log n)$	Quasi-linéaire	Tri fusion, tri rapide (moyen), tri par tas
$O(n^2)$	Quadratique	Tri par insertion (pire cas), 2 boucles imbriquées
$O(n^3)$	Cubique	3 boucles imbriquées, multiplication matricielle naïve
$O(2^n)$	Exponentielle	Sous-ensembles, problèmes combinatoires
$O(n!)$	Factorielle	Permutations, voyageur de commerce (brute force)

Table 1: Complexités courantes et leurs caractéristiques

3.3 Comparaison pour différentes valeurs de n

n	$\log_2 n$	n	$n \log n$	n^2	2^n
10	3	10	33	100	1024
100	7	100	664	10000	$\approx 10^{30}$
1000	10	1000	9966	1000000	$\approx 10^{301}$
10000	13	10000	132877	10^8	gigantesque

Table 2: Comparaison du nombre d'opérations selon la complexité

Attention !

Pour $n = 10000$:

- Un algorithme en $O(n \log n)$: ≈ 133000 opérations
- Un algorithme en $O(n^2)$: ≈ 100 millions d'opérations
- Un algorithme en $O(2^n)$: **IMPOSSIBLE** à exécuter !

3.4 Règles importantes

Règles à retenir

1. **Les constantes disparaissent :**

$$O(5n^2) = O(n^2) \quad \text{et} \quad O(100n) = O(n)$$

2. **Seul le terme dominant compte :**

$$O(n^2 + n + 100) = O(n^2)$$

$$O(3n^3 + 2n^2 + n) = O(n^3)$$

3. **Comparaison de fonctions :**

$$f(n) < g(n) \Leftrightarrow \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

3.4.1 Exemple de comparaison

Question : Comparer n et n^2

$$\lim_{n\to\infty} \frac{n}{n^2} = \lim_{n\to\infty} \frac{1}{n} = 0$$

Donc $n = O(n^2)$ mais $n^2 \neq O(n)$ ✓

4 Le Master Theorem - FORMULE CLÉ

4.1 Introduction

Le **Master Theorem** (Théorème Général ou Théorème Maître) est une méthode pour résoudre rapidement les récurrences de la forme :

Forme applicable

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

où $a \geq 1$, $b > 1$ et $f(n)$ est asymptotiquement positive.

4.2 Étape préliminaire

Avant d'appliquer les 3 cas, il faut **toujours** calculer :

Valeur critique

$$\boxed{n^{\log_b a}}$$

Cette valeur est la **clé** pour déterminer quel cas appliquer !

Rappel : $\log_b a = \frac{\log a}{\log b}$

4.3 Les 3 cas du Master Theorem

CAS 1 : Les feuilles dominent

Condition : $f(n) = O(n^{\log_b a - \varepsilon})$ pour un $\varepsilon > 0$
Autrement dit : $f(n)$ croît **plus lentement** que $n^{\log_b a}$
Résultat :

$$\boxed{T(n) = \Theta(n^{\log_b a})}$$

CAS 2 : Équilibre

Condition : $f(n) = \Theta(n^{\log_b a})$
Autrement dit : $f(n)$ croît **à la même vitesse** que $n^{\log_b a}$
Résultat :

$$\boxed{T(n) = \Theta(n^{\log_b a} \log n)}$$

On multiplie par un facteur logarithmique !

CAS 3 : La racine domine

Condition 1 : $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pour un $\varepsilon > 0$
Autrement dit : $f(n)$ croît **plus vite** que $n^{\log_b a}$
Condition 2 (régularité) : $a \cdot f(n/b) \leq c \cdot f(n)$ pour un $c < 1$ et n assez grand
Résultat :

$$\boxed{T(n) = \Theta(f(n))}$$

4.4 Méthode d'application - ÉTAPES

Procédure à suivre

1. Identifier a, b, et $f(n)$ dans $T(n) = aT(n/b) + f(n)$
2. Calculer $n^{\log_b a}$
3. Comparer $f(n)$ avec $n^{\log_b a}$:
   - Si $f(n)$ plus petit $\rightarrow$ CAS 1
   - Si $f(n)$ égal $\rightarrow$ CAS 2

- Si $f(n)$ plus grand $\rightarrow$ CAS 3 (vérifier régularité)

4. Écrire le résultat selon le cas

4.5 Exemples détaillés

4.5.1 Exemple 1 : $T(n) = 2T(n/2) + \Theta(n)$ (Tri Fusion)

Étape 1 : Identifier

- $a = 2$
- $b = 2$
- $f(n) = \Theta(n)$

Étape 2 : Calculer $n^{\log_b a}$

$$n^{\log_2 2} = n^1 = n$$

Étape 3 : Comparer

$$f(n) = n = \Theta(n) = \Theta(n^{\log_2 2})$$

C'est le **CAS 2** !
Étape 4 : Résultat

$$\boxed{T(n) = \Theta(n \log n)}$$

4.5.2 Exemple 2 : $T(n) = 9T(n/3) + n$

Étape 1 : Identifier

- $a = 9$
- $b = 3$
- $f(n) = n$

Étape 2 : Calculer $n^{\log_b a}$

$$n^{\log_3 9} = n^2$$

Étape 3 : Comparer

$$f(n) = n = O(n^{2-1}) = O(n^2)$$

$f(n)$ croît plus lentement $\rightarrow$ **CAS 1** !
Étape 4 : Résultat

$$\boxed{T(n) = \Theta(n^2)}$$

4.6 Tableau récapitulatif

Récurrence	a, b, $f(n)$	Cas	Résultat
$T(n) = 2T(n/2) + n$	$2, 2, n$	2	$\Theta(n \log n)$
$T(n) = 9T(n/3) + n$	$9, 3, n$	1	$\Theta(n^2)$
$T(n) = T(n/2) + 1$	$1, 2, 1$	2	$\Theta(\log n)$
$T(n) = 3T(n/4) + n \log n$	$3, 4, n \log n$	3	$\Theta(n \log n)$
$T(n) = 4T(n/2) + n$	$4, 2, n$	1	$\Theta(n^2)$

Table 3: Exemples d'application du Master Theorem

Quand le Master Theorem ne s'applique PAS

Le Master Theorem ne fonctionne pas si :

- $f(n)$ n'est pas polynomialement différent de $n^{\log_b a}$
- Exemple : $T(n) = 4T(n/2) + n^2 \log n$ (entre cas 2 et 3)
- Dans ce cas, utiliser l'arbre récursif ou la substitution

5 Calcul de Complexité d'Algorithmes

5.1 Algorithmes Itératifs (avec boucles)

5.1.1 Règles de base

Règles fondamentales

1. **Instruction simple :** $O(1)$
2. **Séquence d'instructions :** $O(1) + O(1) = O(1)$
3. **Boucle simple :** Nombre d'itérations $\times$ coût du corps
4. **Boucles imbriquées :** Multiplier les complexités
5. **Boucles successives :** Additionner puis garder le max

5.1.2 Exemple 1 : Boucle simple

```
pour i de 1 --> n faire
    instruction                 # O(1)
fin pour
```

Listing 2: Boucle simple

Analyse :

- Nombre d'itérations : n
- Coût par itération : $O(1)$

- **Total :** $n \times O(1) = O(n)$

5.1.3 Exemple 2 : Deux boucles imbriquées

```
pour i de 1 --> n faire
    pour j de 1 --> n faire
        instruction              # O(1)
    fin pour
fin pour
```

Listing 3: Boucles imbriquées indépendantes

Analyse :

- Boucle externe : n itérations
- Pour chaque i, boucle interne : n itérations
- **Total :** $n \times n = n^2$ opérations $\rightarrow O(n^2)$

5.1.4 Exemple 3 : Boucles dépendantes

```
pour i de 1 --> n faire
    pour j de 1 --> i faire
        instruction              # O(1)
    fin pour
fin pour
```

Listing 4: Boucle interne dépend de l'externe

Analyse détaillée :

$$\begin{aligned} \text{Total} &= \sum_{i=1}^{n} i \\ &= 1 + 2 + 3 + \cdots + n \\ &= \frac{n(n+1)}{2} \\ &= \frac{n^2 + n}{2} \\ &= O(n^2) \end{aligned}$$

Formule importante

$$\boxed{\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)}$$

5.1.5 Exemple 4 : Boucle logarithmique

```
i = 1
tant que i < n faire
    instruction                    # O(1)
    i = i * 2
fin tant que
```

Listing 5: Boucle avec doublement

Analyse :

- Valeurs de i : $1, 2, 4, 8, 16, \ldots, 2^k < n$
- On s'arrête quand $2^k \geq n$
- Donc $k = \lfloor \log_2 n \rfloor$
- **Total :** $O(\log n)$

5.1.6 Exemple 5 : Boucles successives

```
pour i de 1 --> n faire              # O(n)
    instruction                    # O(1)
fin pour

pour j de 1 --> n faire              # O(n)
    instruction                    # O(1)
fin pour
```

Listing 6: Boucles l'une après l'autre

Analyse :

- Première boucle : $O(n)$
- Deuxième boucle : $O(n)$
- Total : $O(n) + O(n) = O(2n) = O(n)$

Règle importante

Quand des boucles sont **successives** (pas imbriquées), on additionne puis on garde le terme dominant :

$$O(n^2) + O(n) = O(n^2)$$

$$O(n) + O(n) = O(n)$$

5.2 Algorithmes Récursifs

Pour analyser un algorithme récursif :

1. **Écrire** l'équation de récurrence $T(n)$

2. **Identifier** a, b, et $f(n)$
3. **Appliquer** le Master Theorem (si applicable)
4. **Sinon**, utiliser l'arbre récursif ou la substitution

5.2.1 Exemple : Recherche Dichotomique

```
RechercheB(A, v, debut, fin):
    si debut > fin alors
        retourner -1            # O(1)

    milieu = (debut + fin) / 2  # O(1)

    si A[milieu] = v alors
        retourner milieu        # O(1)
    sinon si A[milieu] > v alors
        retourner RechercheB(A, v, debut, milieu-1)   # T(n/2)
    sinon
        retourner RechercheB(A, v, milieu+1, fin)     # T(n/2)
```

Listing 7: Recherche Dichotomique Récursive

Récurrence :

$$T(n) = T(n/2) + O(1)$$

Application Master Theorem :

- $a = 1$, $b = 2$, $f(n) = 1$
- $n^{\log_2 1} = n^0 = 1$
- $f(n) = 1 = \Theta(1) \rightarrow$ CAS 2
- **Résultat :** $T(n) = \Theta(\log n)$

6 Algorithmes Classiques à Connaître

6.1 Recherche Dichotomique

Principe

Dans un tableau **trié**, chercher un élément en divisant l'espace de recherche par 2 à chaque étape.

6.1.1 Version Itérative

```
RechercheB(A, v):
    debut = 1
    fin = longueur(A)

    tant que debut <= fin faire
```

```
        milieu = (debut + fin) / 2

        si A[milieu] = v alors
            retourner milieu
        sinon si A[milieu] > v alors
            fin = milieu - 1
        sinon
            debut = milieu + 1

    retourner -1  # Pas trouve
```

Listing 8: Recherche Dichotomique Itérative

Complexité : $O(\log n)$

6.2 Tri par Insertion

```
TriInsertion(A, n):
    pour j de 2 --> n faire
        cle = A[j]
        i = j - 1

        tant que i > 0 et A[i] > cle faire
            A[i+1] = A[i]
            i = i - 1

        A[i+1] = cle
```

Listing 9: Tri par Insertion

Complexité :

- **Meilleur cas** (tableau déjà trié) : $O(n)$
- **Pire cas** (tableau trié à l'envers) : $O(n^2)$
- **Cas moyen** : $O(n^2)$

6.3 Tableau comparatif

Algorithme	Meilleur	Moyen	Pire
Recherche linéaire	$O(1)$	$O(n)$	$O(n)$
Recherche dichotomique	$O(1)$	$O(\log n)$	$O(\log n)$
Tri par insertion	$O(n)$	$O(n^2)$	$O(n^2)$
Tri fusion	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

Table 4: Complexités des algorithmes classiques

7 Exercices Types pour le CC1

7.1 Type 1 : Master Theorem

Exercice

Donner la complexité de $T(n) = 4T(n/2) + n$ en utilisant le Master Theorem.

Solution :

1. Identifier : $a = 4$, $b = 2$, $f(n) = n$
2. Calculer : $n^{\log_2 4} = n^2$
3. Comparer : $f(n) = n = O(n^{2-1})$
4. $f(n)$ plus petit $\rightarrow$ **CAS 1**
5. **Réponse :** $\boxed{T(n) = \Theta(n^2)}$

7.2 Type 2 : Prouver une notation Big-O

Exercice

Montrer que $n \log n = O(n^2)$

Solution :
Il faut montrer : $\exists c > 0, \exists n_0$ tels que $n \log n \leq c \cdot n^2$ pour tout $n \geq n_0$
Divisons par n : $\log n \leq c \cdot n$
Cette inégalité est vraie pour tout n assez grand car $\log n$ croît beaucoup plus lentement que n.
Prenons $c = 1$ et $n_0 = 2$:

- Pour $n = 2$: $\log 2 = 1 \leq 2$ ✓
- Pour $n = 10$: $\log 10 \approx 3.3 \leq 10$ ✓

Donc $n \log n = O(n^2)$ ✓

7.3 Type 3 : Complexité de boucles

Exercice

Quelle est la complexité de ce programme ?

```
pour i de 0 --> n faire
    pour j de 0 --> n faire
        print("hello")
    pour k de 0 --> n faire
        print("world")
```

Solution :

- Boucle externe (i) : n itérations
- Pour chaque i :
  - Boucle j : n itérations
  - Boucle k : n itérations
  - Total : $n + n = 2n$
- Total général : $n \times 2n = 2n^2$
- **Réponse :** $\boxed{O(n^2)}$

7.4 Type 4 : Analyse d'algorithme récursif

Exercice

Que fait cet algorithme et quelle est sa complexité ?

```
Atrouver(A, debut, fin, elem):
    si debut = fin alors
        si A[debut] = elem alors
            retourner 1
        sinon
            retourner 0
    m = (debut + fin) / 2
    gauche = Atrouver(A, debut, m, elem)
    droite = Atrouver(A, m+1, fin, elem)
    retourner gauche + droite
```

Solution :
Fonction : Compte le nombre d'occurrences de `elem` dans `A[debut..fin]`
Récurrence : $T(n) = 2T(n/2) + O(1)$

- $a = 2$, $b = 2$, $f(n) = 1$
- $n^{\log_2 2} = n$
- $f(n) = 1 = O(n^{-1}) \rightarrow$ CAS 1
- **Complexité :** $\boxed{O(n)}$

7.5 Type 5 : Synthèse

Exercice

Créer un algorithme pour trier un tableau 2D de dimensions $2 \times n$ tel que :

- Chaque ligne est triée (gauche $\rightarrow$ droite)
- Chaque colonne est triée (haut $\rightarrow$ bas)

Donner sa complexité.

Solution (approche fusion) :

```
TrierTableau2D(T, n):
    # 1. Fusionner tout dans un tableau
    temp = creer_tableau(2*n)
    pour j de 1    n faire
        temp[j] = T[1][j]
        temp[n+j] = T[2][j]

    # 2. Trier le tableau
    insertion(temp, 2*n)

    # 3. Redistribuer
    pour j de 1    n faire
        T[1][j] = temp[j]       # n plus petits
        T[2][j] = temp[n+j]     # n plus grands
```

Listing 10: Tri tableau 2D

Complexité :

- Fusion : $O(n)$
- Tri : $O((2n)^2) = O(4n^2) = O(n^2)$
- Distribution : $O(n)$
- **Total :** $\boxed{O(n^2)}$

8 Pièges et Erreurs Courantes

8.1 Erreur 1 : Oublier d'enlever les constantes

Erreur fréquente

Faux : $T(n) = 5n^2$ donc la complexité est $O(5n^2)$
Correct : $T(n) = 5n^2$ donc la complexité est $O(n^2)$

Règle : En notation Big-O, on enlève TOUJOURS les constantes multiplicatives.

8.2 Erreur 2 : Additionner au lieu de multiplier

Erreur fréquente

Pour deux boucles imbriquées de n :

```
pour i de 1 --> n:
    pour j de 1 --> n:
        ...
```

Faux : $O(n + n) = O(n)$ ✗

Correct : $O(n \times n) = O(n^2)$ ✓

Règle :

Boucles imbriquées → **MULTIPLIER**.
Boucles successives → **ADDITIONNER**.

8.3 Erreur 3 : Confondre les cas du Master Theorem

Piège classique

Pour $T(n) = 2T(n/2) + n$:
Faux : $f(n) = n$ et $n^{\log_2 2} = n$, donc $f(n)$ est plus grand → CAS 3
Correct : $f(n) = n = \Theta(n^{\log_2 2})$ → CAS 2 !
Résultat : $T(n) = \Theta(n \log n)$

8.4 Erreur 4 : Oublier la condition de régularité (Cas 3)

Ne pas oublier

Pour appliquer le CAS 3, il faut DEUX conditions :

1. $f(n) = \Omega(n^{\log_b a + \varepsilon})$
2. $a \cdot f(n/b) \leq c \cdot f(n)$ avec $c < 1$ (régularité)

Si la 2ème condition n'est pas vérifiée, le CAS 3 ne s'applique pas !

8.5 Erreur 5 : Ne pas reconnaître les algorithmes classiques

À retenir PAR CŒUR

- Recherche dichotomique → TOUJOURS $O(\log n)$
- Tri fusion → TOUJOURS $O(n \log n)$
- 2 boucles for imbriquées de 1 à n → TOUJOURS $O(n^2)$

9 Checklist Avant le CC1

9.1 Formules à savoir PAR CŒUR

Formules essentielles

1. **Master Theorem :**
$$T(n) = aT(n/b) + f(n)$$
Calculer $n^{\log_b a}$ puis comparer avec $f(n)$

2. **Sommes importantes :**
$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2)$$
$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} = \Theta(n^3)$$

3. **Échelle de complexité :**
$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(2^n)$$

4. **Complexités classiques :**
   - Recherche dichotomique : $O(\log n)$
   - Tri fusion : $O(n \log n)$
   - 2 boucles imbriquées : $O(n^2)$

9.2 Méthodologie pour le Master Theorem

Procédure systématique

1. Écrire la récurrence : $T(n) = aT(n/b) + f(n)$
2. Identifier : $a =?$, $b =?$, $f(n) =?$
3. Calculer : $n^{\log_b a} =?$
4. Comparer $f(n)$ avec $n^{\log_b a}$:
   - $f(n)$ plus petit $\rightarrow$ CAS 1 $\rightarrow T(n) = \Theta(n^{\log_b a})$
   - $f(n)$ égal $\rightarrow$ CAS 2 $\rightarrow T(n) = \Theta(n^{\log_b a} \log n)$
   - $f(n)$ plus grand + régularité $\rightarrow$ CAS 3 $\rightarrow T(n) = \Theta(f(n))$
5. Écrire la réponse finale

9.3 Conseils pour le jour J

Stratégie d'examen

1. **Gérer son temps**
   - Lire tout le sujet d'abord
   - Commencer par les questions faciles
   - Ne pas rester bloqué sur une question
2. **Pour les questions Master Theorem**
   - Toujours calculer $n^{\log_b a}$ en premier
   - Écrire les étapes (montre que vous savez)
   - Vérifier la régularité si CAS 3
3. **Pour les preuves Big-O**
   - Écrire la définition mathématique
   - Donner des valeurs explicites de c et n_0
   - Vérifier avec un exemple
4. **Pour la complexité de boucles**
   - Compter les itérations de chaque boucle
   - Multiplier si imbriquées
   - Additionner si successives
   - Garder le terme dominant

9.4 Ce qui sera probablement dans le CC1

Prédictions basées sur l'annale

- ☐ 2-3 questions Master Theorem (différents cas)
- ☐ 1-2 preuves de notation Big-O
- ☐ 2-3 calculs de complexité (boucles)
- ☐ 1-2 analyses d'algorithmes récursifs
- ☐ 1 exercice de synthèse (type tri 2D)

Durée estimée : 1h30 - 2h
Points : Environ 20 points

BON COURAGE POUR VOTRE Partiel !

Références

Sources du cours

- **Cours principal :** Prof. Mohamed BAKHOUYA, *Algorithmique et Structures de Données Avancées*, Chapitre 1 : Rappel : méthodes de conception et calcul asymptotique, UIR, 2025-2026
- **Ouvrage de référence :** Thomas H. CORMEN, Charles E. LEISERSON, Ronald L. RIVEST, Clifford STEIN, *Introduction à l'algorithmique*, 2ème édition, DUNOD, 2002, 1146 pages
- **Assistant IA :** Claude (Sonnet 4.5), Anthropic, https://claude.ai
- **Complément :** Donald E. KNUTH, *The Art of Computer Programming*, Volume 1

Remerciements

Ce document de révision a été créé pour faciliter la préparation au CC1 d'Algorithmique Avancée. Il synthétise le Chapitre 1 du cours du Prof. BAKHOUYA et inclut des exemples, exercices corrigés et conseils pratiques.

Document créé le October 26, 2025
Documents autorisés - Imprimez et annotez !