RÉSUMÉ COMPLET - COMPLEXITÉ ALGORITHMIQUE

USTHB Master SSI/RSD - 20/20

I. BASES DE L'ANALYSE D'UN ALGORITHME

1. Définitions Fondamentales

Problème : Question générique applicable à un ensemble d'éléments
Instance : Question appliquée à un élément spécifique
Algorithme : Ensemble fini d'instructions exécutables en temps fini

2. Propriétés d'un Algorithme

— Entrées : 0 ou plusieurs
— Sortie : au moins 1
— Instructions claires et non ambiguës
— Terminaison garantie après nombre fini d'étapes
— Implé mentable

3. Types de Complexité

Complexité temporelle : Nombre d'opérations élémentaires
Complexité spatiale : Quantité de mémoire utilisée

Trois cas d'analyse :

— **Meilleur cas :** Peu d'intérêt pratique
— **Cas moyen :** Nécessite distribution des données
— **Pire cas :** LE PLUS IMPORTANT - Borne supérieure garantie

4. Notation de Landau (Big-O)

Définitions essentielles :

— $f(n) = O(g(n))$ ssi $\exists c > 0, \exists n_0, \forall n \geq n_0 : f(n) \leq c \cdot g(n)$
— $f(n) = \Omega(g(n))$ ssi $g(n) = O(f(n))$
— $f(n) = o(g(n))$ ssi $\forall c > 0, \exists n_0, \forall n \geq n_0 : f(n) < c \cdot g(n)$
— $f(n) = \Theta(g(n))$ ssi $f = O(g)$ et $g = O(f)$

Ordres courants : $O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$

II. OUTILS MATHÉMATIQUES

1. Sommations - Formules Clés

Série arithmétique :

$$\sum_{k=1}^{n} k = \frac{n(n+1)}{2} = O(n^2)$$

$$\sum_{k=1}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} = O(n^3)$$

Série géométrique :

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1} - 1}{x - 1} \quad (x \neq 1)$$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x} \quad (|x| < 1)$$

Série harmonique :

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$$

Série emboîtée :

$$\sum_{k=p}^{n} (a_k - a_{k+1}) = a_p - a_{n+1}$$

2. Techniques de Calcul de Sommations

A. Linéarité :

$$\sum_{k=1}^{n} (c \cdot a_k + b_k) = c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k$$

B. Borner par le plus grand terme :

$$\sum_{k=1}^{n} a_k \leq n \cdot \max_{1 \leq k \leq n} a_k$$

C. Borner par série géométrique :
Trouver $r < 1$ tel que $\frac{a_{k+1}}{a_k} \leq r$, alors :

$$\sum_{k=0}^{n} a_k \leq a_0 \sum_{k=0}^{n} r^k = a_0 \frac{1 - r^{n+1}}{1 - r}$$

D. Intégration :
Si f monotone croissante :

$$\int_1^n f(x)dx \leq \sum_{k=1}^{n} f(k) \leq \int_1^{n+1} f(x)dx$$

3. Résolution d'Équations de Récurrence

Méthode 1 : Itération (Déroulement)

1. Remplacer $T(n)$ par sa définition récursive
2. Répéter jusqu'à identifier un pattern
3. Exprimer en forme close
4. Simplifier

Exemple : $T(n) = 2T(n/2) + n$

$$\begin{aligned} T(n) &= 2T(n/2) + n \\ &= 2(2T(n/4) + n/2) + n = 4T(n/4) + 2n \\ &= 2^k T(n/2^k) + kn \\ &\text{Pour } k = \log_2 n : T(n) = nT(1) + n \log n = O(n \log n) \end{aligned}$$

Méthode 2 : Substitution

1. Deviner la forme de la solution
2. Prouver par récurrence
3. Trouver les constantes appropriées

Méthode 3 : Arbre de récursion
Dessiner l'arbre des appels, calculer le coût par niveau, sommer tous les niveaux

Méthode 4 : Master Theorem (Méthode Générale)
Pour $T(n) = aT(n/b) + f(n)$ avec $a \geq 1, b > 1$:

— Si $f(n) = O(n^{\log_b a - \epsilon})$ pour $\epsilon > 0$: $T(n) = \Theta(n^{\log_b a})$
— Si $f(n) = \Theta(n^{\log_b a})$: $T(n) = \Theta(n^{\log_b a} \log n)$
— Si $f(n) = \Omega(n^{\log_b a + \epsilon})$ et $af(n/b) \leq cf(n)$ pour $c < 1$: $T(n) = \Theta(f(n))$

4. Fonctions Importantes

Logarithmes :

— $\log_a(xy) = \log_a x + \log_a y$
— $\log_a(x^n) = n \log_a x$
— $\log_a x = \frac{\log_b x}{\log_b a}$ (changement de base)
— $\log_2 n! = \Theta(n \log n)$

Exponentielles :

— $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!} \geq 1 + x$ pour tout x
— $n^b = o(a^n)$ pour tout $a > 1, b > 0$

III. ANALYSE D'ALGORITHMES DE TRI

1. Tri à Bulles (Bubble Sort)

Principe : Parcourir la liste en échangeant les paires consécutives non ordonnées
Algorithme :

```
Pour i de n à 1 faire
    Pour j de 2 à i faire
        Si Tab[j-1] > Tab[j] alors
            Échanger Tab[j-1] et Tab[j]
```

Complexité : $O(n^2)$ dans tous les cas
Espace : $O(1)$

2. Tri par Insertion

Principe : Insérer chaque élément à sa place dans la partie déjà triée
Algorithme :

```
Pour i de 2 à n faire
    v ← T[i]
    j ← i
    Tant que T[j-1] > v faire
        T[j] ← T[j-1]
        j ← j-1
    T[j] ← v
```

Complexité :

— Meilleur cas (déjà trié) : $O(n)$
— Pire cas : $O(n^2)$

Espace : $O(1)$

3. Tri par Sélection

Principe : Chercher le minimum dans la partie non triée et l'échanger avec l'élément frontière
Algorithme :

```
Pour i de 1 à n-1 faire
    m ← i
    Pour j de i+1 à n faire
        Si T[j] < T[m] alors m ← j
    Échanger T[i] et T[m]
```

Complexité : $O(n^2)$ dans tous les cas
Espace : $O(1)$

4. Tri Fusion (Merge Sort)

Principe : Diviser pour régner - diviser en deux, trier récursivement, fusionner
Algorithme :

```
TriFusion(T, g, d)
    Si g < d alors
        m ← (g+d)/2
        TriFusion(T, g, m)
        TriFusion(T, m+1, d)
        Fusion(T, g, m, d)
```

Récurrence : $T(n) = 2T(n/2) + O(n)$
Complexité : $O(n \log n)$ dans tous les cas
Espace : $O(n)$ (non en place)
Stable : Oui

5. Tri Rapide (Quick Sort)

Principe : Choisir pivot, partitionner, trier récursivement les deux parties
Algorithme :

```
TriRapide(T, g, d)
    Si g < d alors
        p ← Partition(T, g, d)
        TriRapide(T, g, p-1)
        TriRapide(T, p+1, d)
```

Partition :

```
Partition(T, g, d)
    pivot ← T[d]
    i ← g-1, j ← d
    Répéter
        i++ jusqu'à T[i]  pivot
        j-- jusqu'à T[j]  pivot
        Échanger T[i] et T[j]
    Jusqu'à j  i
    Retourner i
```

Complexité :
— Meilleur/Moyen cas : $O(n \log n)$
— Pire cas (déjà trié) : $O(n^2)$
Espace : $O(\log n)$ (pile récursion)
Instable

6. Tri par Tas (Heap Sort)

Principe : Construire un tas max, extraire le maximum répétitivement
Propriété du tas : Pour tout nœud i : $T[i] \geq T[2i]$ et $T[i] \geq T[2i+1]$
Algorithme :

```
TriTas(T, n)
    ConstruireTas(T, n)
    Pour i de n à 2 faire
        Échanger T[1] et T[i]
        Tamiser(T, 1, i-1)
```

Tamiser :

```
Tamiser(T, i, n)
    Si 2i  n alors  // i n'est pas feuille
```

```
        Trouver imax (plus grand entre i, 2i, 2i+1)
        Si imax  i alors
            Échanger T[i] et T[imax]
            Tamiser(T, imax, n)
```

Complexité : $O(n \log n)$ dans tous les cas
Espace : $O(1)$ (en place)
Instable

Tableau Comparatif

Algorithme	Meilleur	Pire	Espace	Stable
Tri à Bulles	$O(n^2)$	$O(n^2)$	$O(1)$	Oui
Tri Insertion	$O(n)$	$O(n^2)$	$O(1)$	Oui
Tri Sélection	$O(n^2)$	$O(n^2)$	$O(1)$	Non
Tri Fusion	$O(n \log n)$	$O(n \log n)$	$O(n)$	Oui
Tri Rapide	$O(n \log n)$	$O(n^2)$	$O(\log n)$	Non
Tri par Tas	$O(n \log n)$	$O(n \log n)$	$O(1)$	Non

IV. RÉCURSIVITÉ

1. Types de Récursivité

Directe : Fonction s'appelle elle-même
Indirecte : $f \to g \to f$

2. Analyse de Complexité Récursive

Étapes :

1. Identifier l'équation de récurrence
2. Identifier les conditions de base
3. Résoudre par une des méthodes (itération, substitution, Master Theorem)
4. Vérifier la solution

Exemple Factorielle :

```
Fact(n)
    Si n = 0 alors retourner 1
    Sinon retourner n * Fact(n-1)
```

Récurrence : $T(n) = T(n-1) + O(1) = O(n)$

Exemple Fibonacci :

```
Fib(n)
    Si n  1 alors retourner n
    Sinon retourner Fib(n-1) + Fib(n-2)
```

Récurrence : $T(n) = T(n-1) + T(n-2) + O(1) = O(2^n)$

3. Récurrence avec Division

Recherche Dichotomique :

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

Tri Fusion :

$$T(n) = 2T(n/2) + O(n) = O(n \log n)$$

Multiplication Matrices (Strassen) :

$$T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7}) \approx O(n^{2.81})$$

V. MÉTHODES POUR RÉUSSIR LES EXAMENS (20/20)

A. DÉTERMINER L'INVARIANT DE BOUCLE

Technique : Identifier ce qui reste vrai à chaque itération

Méthode systématique :

1. Identifier les variables modifiées
2. Déterminer la relation entre itérations
3. Exprimer l'invariant mathématiquement
4. Utiliser pour calculer la complexité

Exemples types d'examens :

Ex1 : Boucles imbriquées simples

```
Pour i de 1 à N faire
    Pour j de i+1 à N faire
        opération
```

Invariant : À l'itération i, on effectue $N - i$ opérations
Total : $\sum_{i=1}^{N}(N-i) = \frac{N(N-1)}{2} = O(N^2)$

Ex2 : Doublement de variable

```
i ← 1
Tant que i  N faire
    i ← 2i
    opération
```

Invariant : Après k itérations, $i = 2^k$
Terminaison : $2^k = N \Rightarrow k = \log_2 N$
Complexité : $O(\log N)$

Ex3 : Racine carrée itérative

```
x ← N
Tant que x > 1 faire
    x ← x
```

Invariant : $x_{k+1} = \sqrt{x_k}$
$x_k = N^{1/2^k}$, terminaison si $N^{1/2^k} \leq 2$
$\frac{1}{2^k} \log N \leq \log 2 \Rightarrow k \geq \log_2(\log N)$
Complexité : $O(\log \log N)$

Ex4 : Boucles avec conditions (type Examen)

```
i ← 1, j ← 1
Tant que i  N faire
    Si i = N alors i ← i+1
    Sinon j ← j+1; i ← j
```

Analyse : Simule deux boucles imbriquées
i passe par : 1, 2, 2, 3, 3, 3, ..., $N, N, ..., N$ (N fois)
Total : $\sum_{k=1}^{N} k = O(N^2)$

B. ÉQUATIONS DE RÉCURRENCE (Méthode d'Examen)

Type 1 : Récurrence linéaire

$$T(n) = T(n-a) + f(n)$$

Solution par itération :

$$\begin{aligned} T(n) &= T(n-a) + f(n) \\ &= T(n-2a) + f(n-a) + f(n) \\ &= T(n-ka) + \sum_{i=0}^{k-1} f(n-ia) \end{aligned}$$

Avec $k = n/a$

Type 2 : Division constante

$$T(n) = aT(n/b) + f(n)$$

Appliquer Master Theorem directement

Type 3 : Avec conditions (type Examen)

```
A(n):
    Si n  B alors retourner C
    Sinon retourner A(n-A) + D
```

Méthode :

1. Poser $T(n) = T(n - A) + D$
2. Par itération : $T(n) = T(n - kA) + kD$
3. Condition d'arrêt : $n - kA \leq B \Rightarrow k = \lceil (n - B)/A \rceil$
4. Donc $T(n) = O(n)$ si A, B constants

C. PROBLÈME DE CODAGE DE HUFFMAN (Très Fréquent)

Principe : Arbre binaire où symboles = feuilles, codes = chemins

Construction de l'arbre :

1. Calculer fréquences de chaque symbole
2. Créer feuille pour chaque symbole avec son poids
3. Répéter jusqu'à avoir 1 arbre :
   - Prendre 2 arbres de poids minimum
   - Créer nœud parent avec poids = somme
4. Assigner 0 à gauche, 1 à droite (ou inverse)

Calculer code d'un symbole : Parcourir de la racine à la feuille, concaténer 0/1

Décoder : Parcourir l'arbre selon les bits jusqu'à atteindre une feuille

Complexité :

- Construction : $O(n \log n)$ avec file priorité
- Encodage lettre : $O(h)$ où h = hauteur
- Décodage message : $O(m)$ où m = longueur code binaire

D. STRUCTURE DE DONNÉES (Type Graphe)

Représentation Matrice d'Adjacence ↔ Listes

De Matrice vers PS/LS :

```
k ← 1
Pour i de 1 à N faire
    PS[i] ← k
    Pour j de 1 à N faire
        Si M[i,j] = 1 alors
            LS[k] ← j
            k ← k+1
PS[N+1] ← k
```

Complexité : $O(N^2)$ (parcours matrice)

De PS/LS vers Matrice :

```
Initialiser M à 0
Pour i de 1 à N faire
    Pour k de PS[i] à PS[i+1]-1 faire
        M[i, LS[k]] ← 1
```

Complexité : $O(N + m)$ où m = nb arcs

E. VALIDATION DE SOLUTION (NP-Complet)

Structure type :

1. Vérifier appartenance des éléments
2. Vérifier unicité (pas de doublons)
3. Vérifier contrainte principale (somme, etc.)
4. Vérifier contraintes additionnelles

Exemple Sous-ensemble Somme :

```
Validation(Solution S, Somme Target, Ensemble E):
    valide ← vrai
    somme ← 0
    Pour chaque élément x de S faire
        Vérifier x  E
        Vérifier x unique dans S
        somme ← somme + x
    Vérifier somme = Target
    Retourner valide
```

Complexité : $O(|S| \times |E|)$ ou $O(|S|^2)$ selon implémentation

F. CONSEILS STRATÉGIQUES EXAMEN

Boucles imbriquées :

— Identifier borne inférieure et supérieure
— Exprimer nombre d'itérations pour chaque niveau
— Multiplier ou sommer selon imbrication

Boucles tant que :

— Trouver relation de récurrence pour variable contrôle
— Résoudre pour trouver nombre d'itérations
— Types courants : $i \leftarrow 2i$ $(\log n)$, $i \leftarrow i^2$ $(\log \log n)$, $i \leftarrow \sqrt{i}$

Récursivité :

— Toujours écrire équation de récurrence
— Identifier cas de base
— Utiliser Master Theorem si forme $T(n) = aT(n/b) + f(n)$
— Sinon itération ou substitution

Arbres de résolution :

— Binaire : 2^n nœuds max, complexité $O(2^n)$
— n-aire : jusqu'à $n!$ nœuds, complexité exponentielle
— Avec élagage : analyser contraintes pour borner

Erreurs à éviter :

— Ne pas confondre $O(n!)$ et $O(n^n)$
— Attention aux constantes qui varient avec n
— Dans récurrence, vérifier cas de base
— Complexité spatiale : ne pas oublier pile récursion

G. CHECK-LIST AVANT EXAMEN

Je sais faire :

☐ Calculer complexité boucles simples, imbriquées, conditionnelles
☐ Identifier et résoudre équations de récurrence
☐ Appliquer Master Theorem
☐ Analyser algorithmes de tri (tous)
☐ Construire arbre Huffman et coder/décoder
☐ Calculer sommations (arithmétique, géométrique, harmonique)
☐ Déterminer invariants de boucle

☐ Écrire algorithme validation pour problème NP
☐ Convertir structures de données (graphe, arbre)
☐ Estimer taille arbre résolution (backtracking)