

# Cours d'algorithmique et programmation 3

Said Jabbour

UFR des Sciences  
Licence Sciences et Technologie  
mentions Mathématiques et Informatique  
Semestre 3

Année universitaire 2024–2025

# Sommaire

1. Complexité
2. Récursivité
3. La dichotomie
4. Algorithmes de Tri
5. Les listes chaînées
6. Une version itérative pour les listes chaînées
7. Une version récursive pour les listes chaînées
8. Les arbres binaires

## Section 1

### Présentation de l'unité

# Modalités de Contrôle de Connaissances : Algorithmique et Programmation 3

- ▶ Deux Contrôles Continu : CC1, CC2
- ▶ Un Projet (généralement un jeu) : PRJ
- ▶  $CC = \frac{2*CC1+2*CC2+TP}{5}$
- ▶ Note Algo3 = CC
- ▶ Note SAE Algo3 : Projet

## Plusieurs problèmes algorithmiques

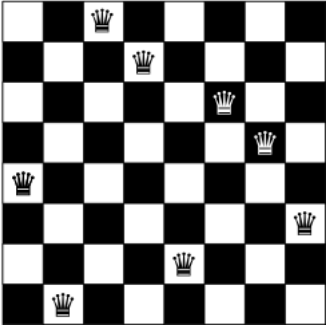
- ▶ **simples**
  - ▶ Plus grande valeur dans une liste
  - ▶ deuxième plus grande valeur dans une liste
  - ▶
- ▶ **Complicés**
  - ▶ **Trier** une liste
  - ▶ **résoudre** le problème des  $n$  reines
  - ▶ Parcours du cavalier
  - ▶ Fouille de texte
  - ▶ Coloriage de graphes, problème de couverture de graphes, etc.

## Objectifs

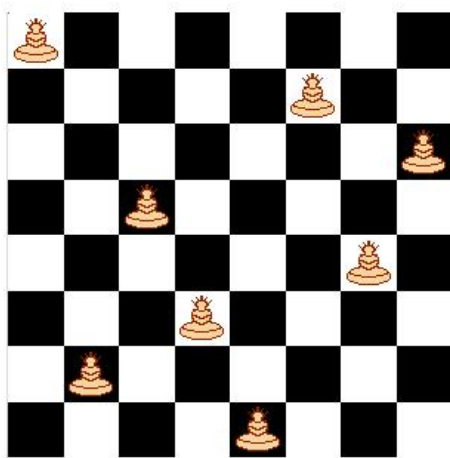
- ▶ Solutions algorithmiques pour répondre à ces problèmes
- ▶ Utilisation des structures de données adéquates
- ▶ Evaluer leurs complexités

# Problème des huit reines

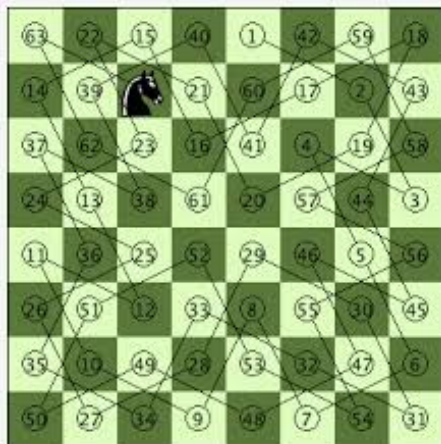
Index

0	2	→	
1	3	→	
2	5	→	
3	6	→	
4	0	→	
5	7	→	
6	4	→	
7	1	→	

# Problème des huit reines

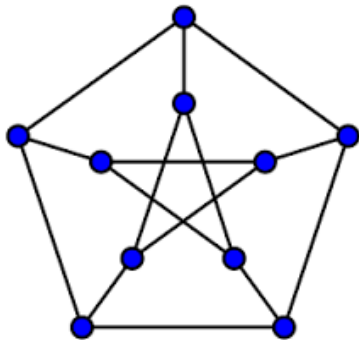


## Parcours du cavalier

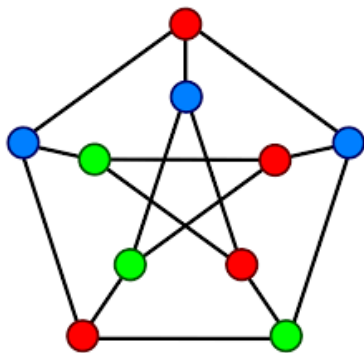




## Colriage de Graphes



## Colriage de Graphes

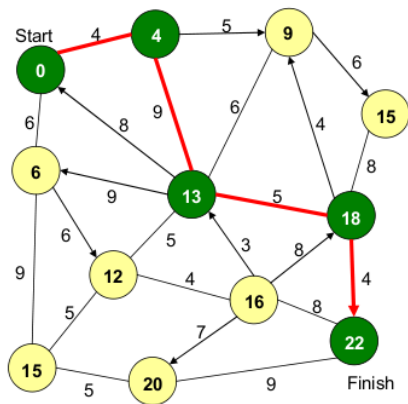


# Fouille de motifs ensemblistes

Client 1	oeufs, farine, Levure, sucre
Client 2	oeufs, farine, fromage, yaourt
Client 3	oeufs, farine, course, boisson
Client 4	pattes, yaourt, sauce tomates
Client 5	pattes sauce tomates
Client 6	boisson, cacahuètes, chips

sous-ensembles fréquents (au moins deux fois  
*{oeufs}*, *{pattes, saucetomates}*, etc.

## Plus court Chemin



## Sudoku Puzzle

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

# Résoudre un ensemble d'inéquations linéaires

$$x_1 + x_2 \geq 1$$

$$x_1 - x_2 \geq 0$$

$$-x_1 + x_2 \geq 0$$

$$-x_1 - x_2 \geq -1$$

$$x_1 \in \{0, 1\}, x_2 \in \{0, 1\}$$

## Section 2

### Complexité

# Complexité

**Temps de calcul**

**Ordres de grandeur Comparaisons de complexités**



Lequel de ces deux algorithmes est le meilleur ?

```
def min_max_v1(liste):  
    mini,maxi = liste[0],liste[0]  
    for i in range(1,len(liste)):  
        if maxi < liste[i]:  
            maxi = liste[i]  
        elif liste[i] > mini:  
            mini = liste[i]  
    return mini,maxi
```

```
def min_max_v2(liste):  
    mini,maxi = liste[0],liste[0]  
    for i in range(1,len(liste)):  
        if liste[i] < mini:  
            mini = liste[i]  
    for i in range(1,len(liste)):  
        if maxi < liste[i]:  
            maxi = liste[i]  
    return mini,maxi
```

# Temps de calcul

## Complexité en temps

$C(A, D)$  = temps d'exécution de l'algorithme  $A$  appliqué aux données  $D$ .

### Éléments de calcul de la complexité en temps

- ▶  $C(\text{opération élémentaire}) = \text{constant}$
- ▶  $C(\text{si } F \text{ alors } I \text{ sinon } J) = C(F) + \max(C(I), C(J))$
- ▶  $C(\text{pour tout } i \text{ de } e_1 \text{ à } e_2 \text{ faire } E_i) \leq C(e_1) + C(e_2) + \sum C(E_i)$
- ▶ Temps de calcul de procédures récursives : solution d'équations de récurrence

## Temps de calcul (2)

$C(A, D)$  dépend en général de  $D$

- **Complexité au pire**

$$C_{MAX}(A, n) = \max\{C(A, D) ; D \text{ de taille } n\}$$

- **Complexité au mieux**

$$C_{MIN}(A, n) = \min\{C(A, D); D \text{ de taille } n\}$$

- **Complexité en moyenne**

$$C_{MOY}(A, n) = \sum_{D \text{ de taille } n} p(D) \times C(A, D)$$

où  $p(D)$  est la probabilité d'avoir la donnée  $D$

$$C_{MIN}(A, n) \leq C_{MOY}(A, n) \leq C_{MAX}(A, n)$$

# Ordre de grandeur

	$\log_2(n)$	$n^{1/2}$	$n\log_2 n$	$n^2$	$n^3$	$2^n$
n=5	2.32	2.24	11.6	25	125	32
n=10	3.32	3.16	33.2	100	1000	1024
n=100	6.64	10	664	10000	$10^6$	$10^{30}$
n=1000	9.97	31.62	9970	$10^6$	$10^9$	$10^{300}$
n=10000	13.29	100	132900	$10^8$	$10^{12}$	$10^{3000}$

## Ordre de grandeur (2)

Coût $C(n)$	Evolution quand la taille est 10 fois plus grande : $C(n \times 10)$
$\log_2(n)$	$C(n) + 3,32$
$n^{1/2}$	$C(n) \times 3,16$
$n$	$C(n) \times 10$
$n \log_2(n)$	$C(n) \times (10 + e)$
$n^2$	$C(n) \times 100$
$n^3$	$C(n) \times 1000$
$2^n$	$C(n)^{10}$

# Comparaisons de complexités : Ordres de grandeur asymptotique

►  $C(n) = O(f(n))$

$\exists k > 0$  et  $N \in \mathbb{N}$  tels que  $\forall n \geq N$  on a  $C(n) \leq k f(n)$ .

**On dit alors que  $C(n)$  est "grand O" de  $f(n)$ .**

►  $C(n) = \Omega(f(n))$

$\exists k > 0$  et  $N \in \mathbb{N}$ , tels que  $\forall n \geq N$ ,  $C(n) \geq k f(n)$

**On dit alors que  $C(n)$  est "grand oméga" de  $f(n)$ .**

►  $C(n) = \Theta(f(n))$

si  $C(n) = O(f(n))$  et  $C(n) = \Omega(f(n))$

**On dit alors que  $C(n)$  est "grand thêta" de  $f(n)$ .**

# Comparaisons de complexités

## Exemples

- ▶  $n = O(n^2)$  En effet si  $n \geq 1$  on a que  $n \leq n^2$  et donc la condition est vérifiée avec  $N = 1$  et  $k = 1$ .
- ▶  $\log_2(n) = O(n^\alpha)$  pour tout  $\alpha > 0$
- ▶  $n^\alpha = O(a^n)$  pour tout  $\alpha > 0$  et  $a > 1$

## Propriété

- ▶ Si  $C(n) = O(f(n))$  alors  $f(n) = \Omega(C(n))$

**Preuve :**  $\forall n \geq N$ , on a que  $C(n) \leq k \times f(n)$  avec  $k > 0$ .  
Par conséquent, à partir de  $N$  on a que  $f(n) \geq 1/k C(n)$ .

# Comparaisons de Complexités

## Exemples

$$2n^2 + n = \Theta(n^2)$$

$$\forall n \geq 1, 2n^2 + n \leq 2n^2 + n^2 = 3n^2.$$

$$\text{Conséquence : } 2n^2 + n = O(n^2)$$

$$\forall n > 0, 2n^2 + n \geq 2n^2 \rightarrow 2n^2 + n = \Omega(n^2)$$



# Comparaisons de Complexités

## Exemples

$$2n^2 + n = \Theta(n^2)$$

$$\forall n \geq 1, 2n^2 + n \leq 2n^2 + n^2 = 3n^2.$$

$$\text{Conséquence : } 2n^2 + n = O(n^2)$$

$$\forall n > 0, 2n^2 + n \geq 2n^2 \rightarrow 2n^2 + n = \Omega(n^2)$$

# À vous

Quel est la complexité de l'algorithme suivant ?

```
def algo1(liste):  
    res = liste[0]  
    for i in range(1, len(liste)):  
        if res < liste[i]:  
            res = liste[i]  
    return res
```

Quel est la complexité de l'algorithme suivant ?

```
def algo2(n):  
    somme = 0  
    for i in range(n-1):  
        for j in range(i+1, n)  
            somme += i+j  
    return somme
```

# À vous

Quel est la complexité de l'algorithme suivant ?

```
def algo3(n):  
    while n>1:  
        print(n)  
        n = n // 2
```

Quel est la complexité de l'algorithme suivant ?

```
def alg4(n):  
    m = 1  
    while m < n:  
        print(n)  
        m = m * 2
```