

# Institut Préparatoire aux Études d'Ingénieurs de Tunis



**Leila Ben Othman**

Première année de classes préparatoires  
MP-PC

Année universitaire: 2016 - 2017

# Complexité Algorithmique

# Complexité algorithmique

**Objectif:** Comparer deux algorithmes réalisant le même traitement.

→ Choisir le plus efficace → Quels critères?

- Evaluer les ressources nécessaires à l'exécution d'un programme:
  - ① Ressource spatiale: Occupation mémoire
  - ② Ressource temporelle: Temps de calcul/ Temps d'exécution

⇒ On s'intéressera essentiellement à la complexité temporelle (coût en termes de temps).

## Complexité temporelle

- On détermine comment le temps d'exécution d'un algorithme varie en fonction d'un paramètre qui est la **taille** du problème, qu'on notera **n**.
- Exemples de la taille d'un problème:
  - nombre d'éléments d'un tableau
  - nombre de bits nécessaires à la représentation des données
  - taille d'une matrice
  - nombre de sommets et d'arêtes dans un graphe
  - *etc*

## Complexité temporelle

- Lorsqu'on parle de complexité, on s'intéresse à des valeurs de  $n$  très grande.
- Il est nécessaire de préciser les instructions élémentaires:
  - opération arithmétique (addition, soustraction, etc)
  - comparaison
  - affectation
- L'objectif est de déterminer un comportement asymptotique du nombre d'instructions élémentaires nécessaires (choix d'une unité de mesure abstraite).
- On utilisera la notation  $\mathcal{O}(n)$  pour capturer cette notion de comportement asymptotique (notion mathématique)

## Comment évaluer la complexité d'un algorithme?

- On affecte à chaque instruction élémentaire considérée pour l'évaluation un **ordre de grandeur** en fonction de  $n$ .
- Une instruction élémentaire (affectation, comparaison, addition, soustraction, multiplication etc) a en général un faible temps d'exécution: coût constant  $\implies \mathcal{O}(1)$

- Coût d'un traitement séquentiel:

$$\begin{array}{l} p \\ q \\ \implies \text{coût}(p) + \text{coût}(q) \end{array}$$

- Coût d'un traitement conditionnel:

Si condition Alors

$p$   
Sinon

$q$   
FinSi  $\implies \text{coût}(\text{condition}) + \max(\text{coût}(p), \text{coût}(q))$

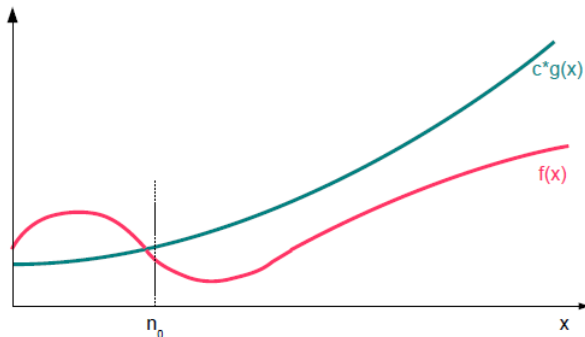
# Comment évaluer la complexité d'un algorithme?

## Coût d'un traitement itératif

- Le cas d'une boucle Pour (le coût de p ne dépend pas de i):  
 Pour i de 1 à n Faire  
     p  
 Fin Pour  $\implies n * \text{coût}(p)$
- Le cas d'une boucle Pour (le coût de p dépend de i):  
 Pour i de 1 à n Faire  
     p(i)  
 Fin Pour  $\implies \sum_{i=1}^n \text{coût}(p(i))$
- Le cas d'une boucle Tant que: On effectue une majoration du nombre d'itérations de la boucle et ainsi on peut majorer le coût de la boucle.

## Complexité temporelle

$f$  et  $g$  étant des fonctions,  $f = \mathcal{O}(g)$  s'il existe des constantes  $c > 0$  et  $n_0$  telles que  $f(x) < c.g(x)$  pour tout  $x > n_0$



$f = \mathcal{O}(g)$  signifie que  $f$  est dominée asymptotiquement par  $g$ .

## Complexité temporelle

Exemple: Soit  $\text{coût}(n) = n^2 + 3n$

- Dès que  $n$  devient un peu importante, le terme  $3n$  augmente moins vite que  $n^2$ . On dit qu'il est négligeable devant le terme  $n^2$ .
- Pour décrire l'efficacité d'un algorithme, seul le terme qui croît le plus vite a un intérêt  $\Rightarrow$  **les constantes et les facteurs multiplicatifs sont ignorés.**
- Pour  $n \geq 3$ , on a  $n^2 + 3n \leq 2n^2$ : on dit que  $n^2 + 3n = \mathcal{O}(n^2)$

## Exemples (1)

Évaluer la complexité de ces deux fonctions en termes de multiplications.

### La factorielle

```
def fact(n):
    res=1
    for i in range(2, n+1):
        res=res*i
    return(res)
```

### Évaluation d'un polynôme représenté par une liste de coefficients $[a_0, a_1, a_2, \dots, a_{n-1}]$

```
def eval(L,x):
    s=0
    for i in range(len(L)):
        s=s+L[i]*(x**i)
    return(s)
```

## Exemples (2)

Ces deux fonctions calculent le nombre de diviseurs d'un entier  $n$ . Évaluer la complexité de ces deux fonctions en termes d'instructions élémentaires.

```
def g1(n):
    d = 0
    k = 1
    while k <= n:
        if n % k == 0:
            d += 1
        k += 1
    return d
```

```
def g2(n):
    d = 0
    k = 1
    while k < m.sqrt(n):
        if n % k == 0:
            d += 2
        k += 1
    if k == m.sqrt(n):
        d += 1
    return d
```

## Classes de complexité

- $\mathcal{O}(1)$  - temps constant: Le temps d'exécution ne dépend pas du paramètre  $n$ .
- $\mathcal{O}(\log n)$ : temps logarithmique; Le temps augmente légèrement quand  $n$  augmente.
- $\mathcal{O}(n)$  - temps linéaire: Le temps d'exécution augmente linéairement en fonction de  $n$  (si  $n$  double, le temps double)

## Classes de complexité

- $\mathcal{O}(n^2)$  - temps quadratique: Quand  $n$  double, le temps d'exécution est multiplié par 4.
- $\mathcal{O}(n^k)$  - temps polynomiale: Quand  $n$  double, le temps d'exécution est multiplié par  $2^k$ .
- $\mathcal{O}(2^n)$  - temps exponentiel: Quand  $n$  double, le temps d'exécution est élevé à la puissance 2.

## Classes de complexité

Ordre de grandeur des temps d'exécution d'un problème de taille  $10^6$  sur un ordinateur à un milliard d'opérations par seconde - Extrait du livre Informatique pour tous en classes préparatoires aux grandes écoles, Edition Eyrolles.

|                       | Nom courant         | Temps pour $n = 10^6$ | Remarques                                                                 |
|-----------------------|---------------------|-----------------------|---------------------------------------------------------------------------|
| $\mathcal{O}(1)$      | temps constant      | 1ns                   | Le temps d'exécution ne dépend pas des données traitées, ce qui est rare. |
| $\mathcal{O}(\log n)$ | temps logarithmique | 10ns                  | En pratique, cela correspond à une exécution quasi-instantanée.           |

## Classes de complexité

Ordre de grandeur des temps d'exécution d'un problème de taille  $10^6$  sur un ordinateur à un milliard d'opérations par seconde (suite).

|          | Nom courant       | Temps pour $n = 10^6$ | Remarques                                                                                                                                  |
|----------|-------------------|-----------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| $O(n)$   | temps linéaire    | 1ms                   | Le temps ne devient supérieur à une minute que pour des données de taille comparable à celles des mémoires vives disponibles actuellement. |
| $O(n^2)$ | temps quadratique | 1/4h                  | Ce temps reste acceptable pour des données de taille moyenne ( $n < 10^6$ ).                                                               |

## Classes de complexité

Ordre de grandeur des temps d'exécution d'un problème de taille  $10^6$  sur un ordinateur à un milliard d'opérations par seconde (suite).

|                    | Nom courant       | Temps<br>pour $n = 10^6$                    | Remarques                                                                                                                                           |
|--------------------|-------------------|---------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| $\mathcal{O}(n^k)$ | temps polynomiale | 30 ans<br>si $k=3$                          | Ici, $n^k$ est le plus haut degré d'un polynôme en $n$ .<br>Il n'est pas rare de voir des complexités en $\mathcal{O}(n^3)$ ou $\mathcal{O}(n^4)$ . |
| $\mathcal{O}(2^n)$ | temps exponentiel | plus de $10^{300000}$<br>milliards d'années | Impraticable sauf pour de très petites données ( $n < 50$ ).                                                                                        |

## Complexité temporelle

- Quand on cherche la complexité d'un algorithme, on s'intéresse au pire des cas: c'est un majorant du temps d'exécution.
- On peut également parler de complexité dans le meilleur des cas, ainsi que de la complexité moyenne:
  - La complexité au pire : correspond au cas le plus défavorable (temps d'exécution maximal)
  - La complexité au mieux : correspond au cas le plus favorable (temps d'exécution minimal). En pratique, cette complexité n'est pas très utile.
  - La complexité moyenne : correspond au cas médian (moyenne des temps d'exécution). La complexité moyenne se base sur des modèles probabilistes complexes difficiles à mettre en ouvre.

## Complexité temporelle

- Calcul de la somme/moyenne des éléments d'un tableau:  $\mathcal{O}(n)$
- Recherche du min/max dans un tableau :  $\mathcal{O}(n)$
- Recherche séquentielle d'un élément dans un tableau :  $\mathcal{O}(n)$
- Recherche dichotomique (tableau trié):  $\mathcal{O}(\log(n))$ .
- Algorithmes de tri : sélection, à bulles, insertion  $\mathcal{O}(n^2)$