

# I21: Introduction à l'algorithmique

Cours 2: Analyse des algorithmes

Nicolas Méloni

Licence 1 (2017-2020)

## Prévoir les ressources nécessaire à son exécution

- ❑ Le type de ressource dépend du contexte :

## Prévoir les ressources nécessaire à son exécution

- ❑ Le type de ressource dépend du contexte :
  - ❑ temps

## Prévoir les ressources nécessaire à son exécution

- ❑ Le type de ressource dépend du contexte :
  - ❑ temps
  - ❑ espace mémoire

## Prévoir les ressources nécessaire à son exécution

- ❑ Le type de ressource dépend du contexte :
  - ❑ temps
  - ❑ espace mémoire
  - ❑ consommation électrique

## Prévoir les ressources nécessaire à son exécution

- ❑ Le type de ressource dépend du contexte :
  - ❑ temps
  - ❑ espace mémoire
  - ❑ consommation électrique
  - ❑ coût financier

## Indépendance

L'analyse ne doit pas dépendre :

- ❑ de la machine sur laquelle tourne l'algorithme
  - ❑ du langage de programmation utilisé pour l'implanter
- ❑ Nécessité de concevoir un modèle d'étude indépendant : le modèle RAM.

## (R)andom (A)ccess (M)achine

Machine hypothétique pour laquelle :

- les opérations simple ( $+$ ,  $-$ ,  $*$ ,  $/$ ,  $\text{if}$ , appels) consomment une unité de temps
- les boucles sont des compositions d'opération simples, leur temps d'exécution dépend du nombre d'itérations et de la nature des opérations à l'intérieur de la boucle
- un accès mémoire consomme une unité de temps
- la quantité de mémoire n'est pas limitée



- Mesurer le temps d'exécution = compter le nombre d'étapes effectuées pour une instance donnée.

Malgré sa simplicité, le modèle permet une analyse très juste du comportement d'un algorithme sur une machine réelle.

## Problème : recherche d'un élément dans un tableau

*Entrée* : un tableau de  $n$  élément et un élément  $e$

*Sortie* : l'indice du tableau où se trouve l'élément ou 0 s'il ne s'y trouve pas

---

```
1  DEBUT
2     $i \leftarrow 1$ 
3    TQ  $i \leq n$  FAIRE
4      SI  $e = T[i]$  ALORS
5        RENVOYER  $i$ 
6      FSI
7       $i \leftarrow i+1$ 
8    FTQ
9    RENVOYER 0
10 FIN
```

---

---

- ❑ A priori, plus la taille de l'entrée est grande plus longue est la résolution du problème.
- ❑ Pour étudier l'efficacité d'un algorithme on considère toujours des instances du problème à taille fixée.
- ❑ La complexité d'un algorithme nous renseigne sur comment évolue le temps d'exécution avec la taille de l'entrée.
- ❑ C'est une fonction de  $n$  souvent notée  $C(n)$  ou  $T(n)$ .

## Problème de la recherche d'un élément :

- ❑  $n$  cases à tester
- ❑ 5 cases : au plus 5 tests
- ❑ 10 cases : au plus 10 tests

## Problème du ramassage de plots :

- ❑  $n!$  chemins à tester
- ❑ 5 plots : 120 chemins possibles
- ❑ 10 plots : 3628800 chemins possibles !

- Même à taille fixée, certaines instances peuvent être plus faciles à résoudre que d'autres.

## Problème de la recherche d'un élément :

- cas facile :
- cas difficile :
- cas moyen :

- Même à taille fixée, certaines instances peuvent être plus faciles à résoudre que d'autres.

## Problème de la recherche d'un élément :

- cas facile : l'élément recherché est au début du tableau (1 tour de boucle)
- cas difficile :
- cas moyen :

- Même à taille fixée, certaines instances peuvent être plus faciles à résoudre que d'autres.

## Problème de la recherche d'un élément :

- cas facile : l'élément recherché est au début du tableau (1 tour de boucle)
- cas difficile : l'élément recherché n'est pas dans le tableau ( $n$  tours de boucle)
- cas moyen :

- Même à taille fixée, certaines instances peuvent être plus faciles à résoudre que d'autres.

### Problème de la recherche d'un élément :

- cas facile : l'élément recherché est au début du tableau (1 tour de boucle)
- cas difficile : l'élément recherché n'est pas dans le tableau ( $n$  tours de boucle)
- cas moyen : l'élément recherché est au milieu (environ  $n/2$  tours de boucle)



- ❖ On considère traditionnellement trois cas :
  - ❖ Meilleur cas :  $\check{T}(n)$
  - ❖ Pire cas :  $\hat{T}(n)$
  - ❖ Cas moyen :  $\bar{T}(n)$

## Meilleur cas

- ❖ Le nombre minimal d'étapes effectuées par l'algorithme pour n'importe quelle instance de taille  $n$  du problème.

## Pire cas

- ❖ Le nombre maximal d'étapes effectuées par l'algorithme pour n'importe quelle instance de taille  $n$  du problème.

## Cas moyen

- ❖ Le nombre moyen d'étapes effectuées par l'algorithme pour l'ensemble des instances de taille  $n$  du problème.

La complexité d'un algorithme est toujours une fonction numérique. Elles sont difficiles à manipuler :

- ❑ *Trop de cas à gérer* : l'écriture ne peut se faire en une seule formule close.
- ❑ *Trop complexe* : l'écriture exacte fait intervenir de nombreux termes qui n'apporte pas beaucoup d'information.

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

Il existe une constante  $c$  et un entier  $n_0$  tels que  
 $\forall n \geq n_0, f(n) \leq cg(n)$

- ❖  $3n^2 - n + 6 = O(n^2)$  en prenant  $c = 3$  et  $n_0 = 6$
- ❖  $3n^2 - n + 6 = O(n^3)$  en prenant  $c = 1$  et  $n_0 = 4$
- ❖  $3n^2 - n + 6 \neq O(n)$  car  $\forall c, cn < 3n^2 - n + 6$  quand  $n > c + 1$

$f(n) = \Omega(g(n))$  (oméga)

Il existe une constante  $c$  et un entier  $n_0$  tels que  
 $\forall n \geq n_0, f(n) \geq cg(n)$ .

- ❖  $3n^2 - n + 6 = \Omega(n^2)$  en prenant  $c = 2$  et  $n_0 = 2$
- ❖  $3n^2 - n + 6 \neq \Omega(n^3) \forall c, 3n^2 - n + 6 < cn^3$  quand  $cn > 3$  et  $n > 6$
- ❖  $3n^2 - n + 6 = \Omega(n)$  en prenant  $c = 1$  et  $n_0 = 1$

### $f(n) = \Theta(g(n))$ (thétha)

Il existe deux constantes  $c_1$  et  $c_2$  ainsi qu'un entier  $n_0$  tels que  
 $\forall n \geq n_0, c_1g(n) \leq f(n) \leq c_2g(n)$ .  
( $f(n) = O(g(n))$  et  $f(n) = \Omega(g(n))$ )

- ❑  $3n^2 - n + 6 = \Theta(n^2)$
- ❑  $3n^2 - n + 6 \neq \Theta(n^3)$
- ❑  $3n^2 - n + 6 \neq \Theta(n)$

## Mnémotechnique

- ❖  $f(n) = O(g(n))$  :  $f$  est plus petite que  $g$
- ❖  $f(n) = \Omega(g(n))$  :  $f$  est plus grande que  $g$
- ❖  $f(n) = \Theta(g(n))$  :  $f$  est  $g$  son équivalente

## Règles de compositions

- ❖  $f(n) + g(n) = O(\max(f(n), g(n)))$
- ❖  $cf(n) = O(f(n)), \forall c > 0$
- ❖  $f_1(n) = O(g_1(n))$  et  $f_2(n) = O(g_2(n)) \Rightarrow$   
 $f_1(n)f_2(n) = O(g_1(n)g_2(n))$

Valable également pour le  $\Omega$  et  $\Theta$ .



## Logarithmique

- ❖  $f(n) = \log(n)$
- ❖ Propriété du log :
  - ❖  $\log_a(n) = \ln(n)/\ln(a)$
  - ❖  $a = b^{\log_b(a)}$
  - ❖  $\log_c(ab) = \log_c(a) + \log_c(b)$
  - ❖  $a^{\log_b(n)} = n^{\log_b(a)}$

## Polynomiale

- ❖  $f(n) = a_k n^k + a_{k+1} n^{k-1} + \dots + a_1 n + a_0, a_k > 0$
- ❖ Propriété des polynômes :
  - ❖  $f(n) = \Theta(n^k)$
  - ❖  $k = 1$  on parle de complexité linéaire
  - ❖  $k = 2$  on parle de complexité quadratique
  - ❖ On peut étendre la définition au puissance réelles :  
 $n\sqrt{n} = n^{1.5} = O(n^2)$
  - ❖  $\forall a > 1, b > 0, \lim_{n \rightarrow +\infty} \frac{\log(n)^a}{n^b} = 0 \Rightarrow \log(n)^a = O(n^b)$

## Exponentielle

- ❖  $f(n) = a^n$
- ❖ Propriété de l'exponentielle :
  - ❖  $a^0 = 1, a^1 = a, a^{-1} = 1/a$
  - ❖  $a^{m+n} = a^m a^n$
  - ❖  $(a^m)^n = (a^n)^m = a^{mn}$
  - ❖  $\forall a, b > 0, \lim_{n \rightarrow +\infty} \frac{n^b}{a^n} = 0 \Rightarrow n^b = O(a^n)$

## Factorielle

- ❖  $f(n) = n!$
- ❖ Propriété de la factorielle :
  - ❖  $0! = 1$
  - ❖  $n! = n \times (n - 1) \times \cdots \times 2 \times 1$
  - ❖  $(n + 1)! = (n + 1)n!$
  - ❖  $\forall a > 0, \lim_{n \rightarrow +\infty} \frac{a^n}{n!} = 0 \Rightarrow a^n = O(n!)$

Estimation des temps de calcul pour quelques complexités standards (vit. de calcul :  $3.5 \times 10^9$  op/s)

$n$	$\log(n)$	$n$	$n \log(n)$	$n^2$	$2^n$	$n!$
10	$0.001\mu s$	$0.003\mu s$	$0.007\mu s$	$0.029\mu s$	$0.293\mu s$	$0.001s$
20	$0.001\mu s$	$0.006\mu s$	$0.017\mu s$	$0.114\mu s$	$0.3ms$	$22ans$
30	$0.001\mu s$	$0.009\mu s$	$0.029\mu s$	$0.257\mu s$	$0.307s$	$2.4 \times 10^{15} ans$
40	$0.001\mu s$	$0.011\mu s$	$0.042\mu s$	$0.457\mu s$	$5.2min$	$7.3 \times 10^{30} ans$
50	$0.001\mu s$	$0.014\mu s$	$0.056\mu s$	$0.714\mu s$	$3.7journs$	$2.7 \times 10^{47} ans$
100	$0.001\mu s$	$0.029\mu s$	$0.132\mu s$	$0.003ms$	$1.1 \times 10^{13} ans$	
1000	$0.002\mu s$	$0.286\mu s$	$0.002ms$	$0.286ms$		
10000	$0.003\mu s$	$0.003ms$	$0.026ms$	$0.029s$		
100000	$0.003\mu s$	$0.029ms$	$0.329ms$	$2.8s$		
1000000	$0.004\mu s$	$0.286ms$	$0.004s$	$4.7min$		
10000000	$0.005\mu s$	$0.003s$	$0.046s$	$7.9h$		

- ❏ Identifier les valeurs initiales des variables impliquées et la condition d'arrêt
- ❏ Identifier les instructions où sont modifiées les variables dont dépend la condition d'arrêt
- ❏ Compter le nombre d'exécutions
- ❏ Utiliser des techniques de sommation sur les entiers

```
1  DEBUT
2     $i \leftarrow 1$ 
3    TQ  $i \leq n$  FAIRE
4       $i \leftarrow i+1$ 
5    FTQ
6  FIN
```

- initialisation :  $i \leftarrow 1$
- condition d'arrêt :  $i > n$
- $i$  est modifié à l'instruction 4 :  
 $i \leftarrow i + 1$
- les valeurs de  $i$  sont  $1, 2, 3, \dots$
- la condition d'arrêt est réalisée  
quand  $i = n + 1$
- nb tours de boucles :  $n$ ,  
complexité :  $\Theta(n)$

---

```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      bloc d'inst
5      i ← i+1
6    FTQ
7  FIN
```

---

---

■ complexité du bloc :  $f(i)$

■ nb tours de boucles :  $n$

■ complexité :  $\sum_{i=1}^n f(i)$



## Série arithmétique

- ❑  $u_{n+1} = u_n + r$
- ❑  $\sum_{i=1}^n u_i = (n + 1) \frac{u_0 + u_n}{2}$

## Exemple

- ❑  $(u_n) = (0, 1, 2, 3, \dots)$
- ❑  $u_0 = 1, u_{n+1} = u_n + 1 \ (\forall n, u_n = n)$
- ❑  $\sum_{i=0}^n i = (n + 1) \frac{n}{2}$

## Série géométrique

- ❖  $u_{n+1} = qu_n$
- ❖ si  $q \neq 1$ ,  $\sum_{i=1}^n u_i = u_0 \frac{1-q^{n+1}}{1-q}$

## Exemple

- ❖  $(u_n) = (1, 2, 4, 8, 16, \dots)$
- ❖  $u_0 = 1, u_{n+1} = 2u_n$  ( $\forall n, u_n = 2^n$ )
- ❖  $\sum_{i=0}^n 2^i = \frac{1-2^{n+1}}{1-2} = 2^{n+1} - 1$

```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      j ← 1
5      TQ j ≤ i FAIRE
6        j ← j+1
7      FTQ
8    i ← i+1
9  FTQ
10 FIN
```

- ❑ complexité boucle intérieure :  
 $C_{in}(i) = i$
- ❑ nb tours de boucles :  $n$
- ❑ complexité :  $\sum_{i=1}^n C_{in}(i) = (n+1)n/2 = \Theta(n^2)$

```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      j ← 1
5      TQ j ≤ 2i FAIRE
6        j ← j+1
7      FTQ
8    i ← i+1
9  FTQ
10 FIN
```

- complexité boucle intérieure :  
 $C_{in}(i) = 2^i$
- nb tours de boucles :  $n$
- complexité :  $\sum_{i=1}^n C_{in}(i) = 2^{n+1} - 1 = \Theta(2^n)$

```
1  DEBUT
2    i ← n
3    TQ i ≥ 1 FAIRE
4      i ← [i/2]
5    FTQ
6  FIN
```

- ❑ on ne connaît pas les valeurs exactes prises par  $i$
- ❑ on les majore par la suite  $(n, n/2, n/4, \dots)$
- ❑ après  $k$  tours de boucles on a  $i \leq n/2^k$
- ❑ la condition d'arrêt est satisfaite quand  $n/2^k < 1$  i.e.  
 $n < 2^k \Leftrightarrow \log_2(n) < k$
- ❑ nb tours de boucles majoré par  $\lfloor \log_2(n) \rfloor + 1$
- ❑ complexité :  $C(n) = O(\log(n))$

```
1  DEBUT
2    i ← n
3    TQ i ≥ 1 FAIRE
4      i ← [i/2]
5    FTQ
6  FIN
```

- On peut montrer de façon similaire que  $C(n) = \Omega(\log(n))$  et donc que  $C(n) = \Theta(\log(n))$

```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      proc(i) //procédure
5              //en  $\Theta(f(i))$ 
6    i ← i+1
7  FTQ
8  FIN
```

- On encadre la complexité de la procédure  $C_{proc}(i)$  :

$$c_1 f(i) \leq C_{proc(i)} \leq c_2 f(i)$$

(moins trivial qu'il n'y paraît)

- On en déduit :

$$\sum_{i=1}^n c_1 f(i) \leq \sum_{i=1}^n C_{proc(i)} \leq \sum_{i=1}^n c_2 f(i)$$

```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      proc(i) // procedure
5              // en Θ(f(i))
6      i ← i+1
7    FTQ
8  FIN
```

❖ finalement :

$$c_1 \left( \sum_{i=1}^n f(i) \right) \leq C(n) \leq c_2 \left( \sum_{i=1}^n f(i) \right)$$

❖ On en déduit le résultat :

$$C(n) = \Theta \left( \sum_{i=1}^n f(i) \right)$$



```
1  DEBUT
2    i ← 1
3    TQ i ≤ n FAIRE
4      proc(i) //procédure
5              //en Θ(f(i))
6      i ← i+1
7    FTQ
8  FIN
```

- ❖ Ce résultat justifie l'abus de notation :

$$\sum_{i=1}^n (\Theta f(i)) = \Theta \left( \sum_{i=1}^n f(i) \right)$$