Plan du cours

1. Introduction
   - Contenu du cours
2. Logique mathématique
   - Calcul propositionnel
   - Calcul des prédicats
   - Logique floue et aide à la décision
3. Induction
4. Analyse d'algorithmes
   - Comparaison asymptotique de fonctions
   - Complexité
5. Récurrence
6. Mathématique de la gestion
   - Théorie des graphes
   - Optimisation

Analyse d'algorithmes

- Algorithme :
  - « recette », séquence d'actions définies de façon non ambigüe, qui peut être appliquée pour résoudre un problème en un temps fini.
- Analyse d'un algorithme :
  - Approprié à la tâche à laquelle il est destiné ?
  - Meilleur qu'un autre ?
  - Possibilités d'amélioration ?
  - Ressources utilisées en temps et en mémoire ?
  - Performances indépendamment de l'implémentation ?

Outil d'analyse :
Comportement asymptotique de fonctions

- Questions :
  - Peut-on extrapoler le temps requis pour résoudre un problème sur base de jeux de données de test (plus petits…) ?
  - Comment comparer deux algorithmes indépendamment des machines sur lesquelles ils seront implémentés ?
  - Comment s'assurer qu'un algorithme s'adaptera bien à une augmentation de la taille du problème à résoudre ?

Analyse d'algorithmes

- Etudier le temps requis par un algorithme en fonction de la taille du problème.

$\rightarrow$ Comment comparer le taux de croissance de différentes fonctions ?

Analyse d'algorithmes

- Hypothèses simplificatrices :
  - Soient 2 algorithmes A1 et A2 avec :
    - A1 systématiquement plus rapide que A2 pour de « petits » jeux de données.
    - A2 systématiquement plus rapide que A1 pour de « grands » jeux de données.
    - ▸ A2 sera préféré à A1 car « presque toujours » meilleur.
  - On ne tient pas compte des paramètres de performances propres aux ordinateurs sur lesquel l'algorithme est implémenté.

Comparaison asymptotique de fonctions

- Fonctions étudiées :
  - Temps de calcul en fonction de la taille des données (input).
  - Domaine : nombre naturels.
  - Image : nombres réels positifs.

$$F : \mathbb{N} \rightarrow [0, \infty)$$

Dominance asymptotique

- F domine asymptotiquement G si :

$$\exists \quad N_0 \in \mathbb{N}, \quad c \in (0,\infty) \ :$$
$$\forall n \geq N_0 : \ G(n) \leq c \times F(n)$$

- Exemple : $F(n) = n^2$ domine $G(n) = 3n$

$N_0 = 0 \quad c = 3$

$$\begin{aligned} G(n) &= 3n \\ &\leq 3n^2 \\ &\leq 3F(n) \\ &\leq c \times F(n) \end{aligned}$$

Notation

- Définition : soient 2 fonctions F et G :

$$O(F) = \{G : F \text{ domine asymptotiquement } G\}$$

- Expressions courantes :

$$G \in O(F)$$
$$G \text{ est } O(F)$$
$$G = O(F)$$

Autres définitions

$$O(F) = \left\{ G : \frac{G(n)}{F(n)} \leq c, \forall n \geq N_0 \right\}$$

$$\Omega(F) = \left\{ G : \frac{G(n)}{F(n)} \geq d, \forall n \geq N_0 \right\}$$

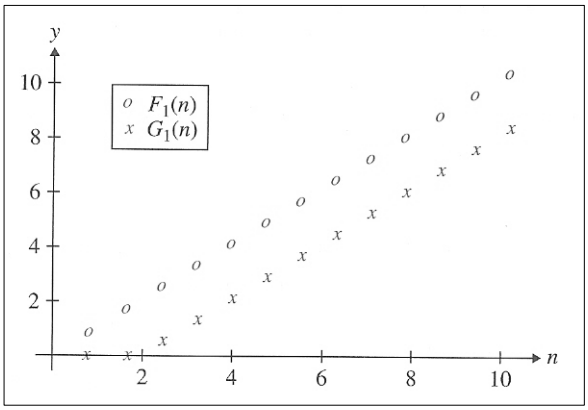
$$\Theta(F) = \left\{ G : d \leq \frac{G(n)}{F(n)} \leq c, \forall n \geq N_0 \right\}$$

Interprétation

- Si $G \in O(F)$ et $F \in O(G)$
- Alors F et G croissent approximativement de la même façon lorsque n est grand.

Exemple 1

$$F_1(n) = n \qquad G_1(n) = \begin{cases} n-2 & n \geq 2 \\ 0 & n < 2 \end{cases}$$



Exemple 1

$$G_1 \in O(F_1) \quad \text{et} \quad F_1 \in O(G_1)$$

$$G_1 \in O(F_1) \qquad G_1(n) \leq F_1(n) \Rightarrow \begin{cases} c = 1 \\ N_0 = 0 \end{cases}$$

$$F_1 \in O(G_1) \quad \begin{cases} c = 2 \\ N_0 = 4 \end{cases} \Rightarrow \begin{aligned} F_1(n) = n &\leq n + (n-4) \\ &\leq 2n - 4 \\ &\leq 2(n-2) \\ &\leq 2G_1(n) \\ &\leq c \times G_1(n) \end{aligned}$$

Exemple 2

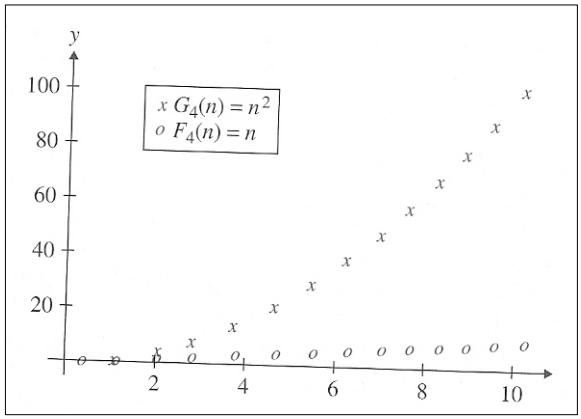
$$F_2(n) = n \qquad G_2(n) = \begin{cases} 100.000.000 & n = 0 \\ n & n > 0 \end{cases}$$

$$G_2 \in O(F_2) \quad \text{et} \quad F_2 \in O(G_2)$$

$$(c = 1 \quad \text{et} \quad N_0 = 1)$$

Exemple 4

$$F_4(n) = n \qquad G_4(n) = n^2$$



Exemple 4

$$F_4 \in O(G_4) \quad \text{mais} \quad G_4 \notin O(F_4)$$

$F_4 \in O(G_4)$ $\quad \begin{cases} c = 1 \\ N_0 = 2 \end{cases} \Rightarrow \quad F_4(n) = n < n^2 = G_4(n)$

$G_4 \notin O(F_4)$

$$\forall N_0, c:$$

$$\text{si } N_0 > c: \quad n = N_0 \quad \Rightarrow \quad G_4(n) > c \times F_4(n)$$

$$\text{si } N_0 \leq c: \quad n > c \quad \Rightarrow \quad G_4(n) > c \times F_4(n)$$

Généralisation de l'exemple 4

$$F(n) = n^k \qquad G(n) = n^{k+1}$$

$$k = 1, 2, 3, \ldots$$

$$F \in O(G) \quad \text{mais} \quad G \notin O(F)$$

Extension à des fonctions partielles

- Fonction partielle : définie pour presque tous les n (tous sauf un nombre fini de valeurs).
- Extension de la définition de dominance : F domine asymptotiquement G si :

$$\exists \quad N_0 \in \mathbb{N}, \quad c \in (0,\infty) \ :$$

$$\forall n \geq N_0 \text{ avec } F(n) \text{ et } G(n) \text{ définis} : \ G(n) \leq c \times F(n)$$

Propriétés (1)

- Théorème 1 :

$$a \in (0,\infty) \quad \rightarrow \quad a \times F \in O(F)$$

- Corollaire :

$$\forall a > 0 : \quad a \times F \in O(F) \quad \text{et} \quad F \in O(a \times F)$$

- Conséquence (a=1) : la dominance asymptotique est une relation réflexive.

Propriétés (2)

- Théorème 2 :

$$\begin{cases} H \in O(G) \\ G \in O(F) \end{cases} \Rightarrow \quad H \in O(F)$$

- Conséquence : la dominance asymptotique est une relation transitive.
- Corrolaire :

$$n^i \in O(n^j) \quad \text{pour} \quad i \leq j$$

Propriétés (3)

- Théorème 3 :

$$\begin{cases} G \in O(F) \\ H \in O(F) \end{cases} \Rightarrow \begin{cases} G+H \in O(F) \\ |G-H| \in O(F) \end{cases}$$

- Corollaire (par induction) :

$$\begin{cases} G_i \in O(F) \\ i = 1,\ldots,k \end{cases}$$

$$\Downarrow$$

$$\begin{cases} |a_1 G_1 + a_2 G_2 + \ldots + a_k G_k| \in O(F) \\ \forall a_1, a_2, \ldots, a_k \in \mathbb{R} \end{cases}$$

Relation d'équivalence

- Théorème 4 :

a. $F \in O(G)$ ssi $O(F) \subseteq O(G)$

b. $O(F) = O(G)$ ssi $F \in O(G)$ et $G \in O(F)$

c. $\{(\mathrm{F},\mathrm{G}) : O(F) = O(G)\}$ est une relation d'équivalence

- Cf. Θ

Fonctions polynomiales

- Définition : fonction polynomiale de degré k :

$$P(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$$

  - Avec $a_k \neq 0$.
  - Fonction partielle : définie seulement pour les n tels que $P(n)$ est positive $\rightarrow a_k > 0$.

Fonctions polynomiales

- Théorème 5 :

$$k_1, k_2 \in \mathbb{R} \quad 0 < k_1 < k_2 \quad x \in \mathbb{R}$$

$$\begin{cases} x^{k_1} \in O\left(x^{k_2}\right) \\ x^{k_2} \notin O\left(x^{k_1}\right) \end{cases}$$

- Corollaire (monômes) :

$$i, j \in \mathbb{N} \quad i < j \quad n \in \mathbb{N}$$

$$\begin{cases} n^i \in O\left(n^j\right) \\ n^j \notin O\left(n^i\right) \end{cases}$$

Fonctions polynomiales

- Théorème 6 : pour une fonction polynomiale de degré k :

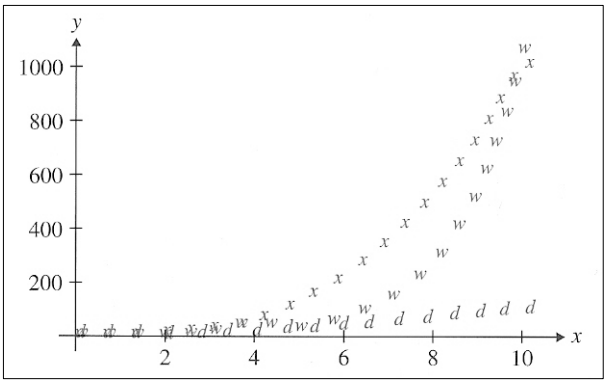
$$P(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_2 n^2 + a_1 n + a_0$$

On a :

$$P(n) \in O\left(n^k\right)$$

$$n^k \in O\left(P(n)\right)$$

Fonctions exponentielles



$$d : n^2 \quad x : n^3 \quad w : 2^n$$

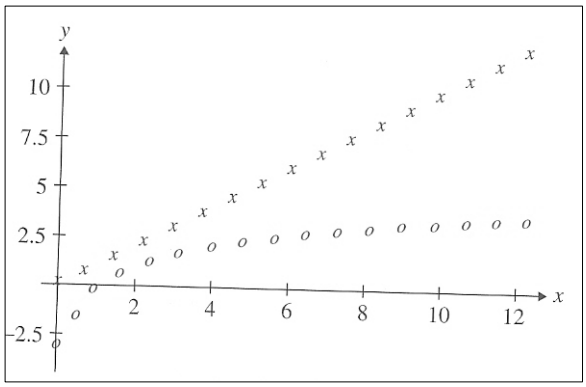
Fonctions exponentielles

- Théorème 7 :

$$k \in \mathbb{R} \quad k \geq 2$$

$$\begin{cases} n^k \in O(2^n) \\ 2^n \notin O(n^k) \end{cases}$$

Fonctions logarithmiques



$x : n \quad o : \ln(n)$

Fonctions logarithmiques

- Corollaire du théorème 7 :

$$\ln(n) \in O(n)$$
$$n \notin O(\ln(n))$$

Comparaison

Table 5.1 Complexity Table for Several Functions

$F(n)$	$log_2(n)$	n	n^2	n^5	2^n
$n = 10$	3×10^{-6} sec	10^{-5} sec	10^{-4} sec	0.1 sec	10^{-3} sec
$n = 20$	4×10^{-6} sec	2×10^{-5} sec	4×10^{-4} sec	3 sec	1 sec
$n = 50$	6×10^{-6} sec	5×10^{-5} sec	3×10^{-3} sec	5 min	36 yrs
$n = 100$	7×10^{-6} sec	10^{-4} sec	10^{-2} sec	3 hrs	4×10^{16} yrs
$n = 1000$	1×10^{-5} sec	10^{-3} sec	1 sec	32 yrs	3.9×10^{287} yrs
$n = 100,000$	2×10^{-5} sec	0.1 sec	2.7 hrs	3×10^{11} yrs	$> 10^{30,089}$ yrs

Complexité d'un algorithme

- Comment estimer le temps d'exécution d'un programme pour un jeu de données de taille donnée à partir des temps observés pour des jeux de données de test (plus petits) ?
- Comment comparer les performances de deux algorithmes indépendant des machines sur lesquelles ils seront implémentés ?
- Comment s'assurer qu'un algorithme s'adaptera bien à des jeux de données de taille plus élevée ?

Complexité

- Mesure de la complexité en temps d'un algorithme.
- Utilisation de la notation O.
- Exemple :
  - Un algorithme de tri en $O(n^2)$ sera moins bon qu'un autre en $O(n \ln(n))$.

Principe de comptage

Control Structures

Sequence: A list of steps $s_1 : s_2 : \ldots : s_k$ to be performed in the order given.
Selection: A choice of steps. In many programming languages, constructs such as *if-then*, *if-then-else*, *select*, and *case* provide the methods to make choices.
Repetition (also called **iteration** and **looping**)**:** A block of code is executed repeatedly, either for a certain number of times or until some condition becomes true. In many programming languages, constructs such as *for*, *while*, *do-while*, and *repeat-until* provide for repetition. (Repetition can also be accomplished with recursion. The analysis is often similar, but we will not discuss recursion in this chapter.)

Exemple 1

Algorithm: Ordering Three Values I

INPUT: Distinct alphabetic values for three variables A, B, and C
OUTPUT: The list of alphabetic values ordered by increasing value

```
if (A > B) then
    Swap(A, B)
if (B > C) then
    Swap(B, C)
if (A > B) then
    Swap(A, B)
print A, B, C
```


Exemple 1

- Que compter ?
  - Comparaisons ?
  - Echanges ?
- Pour une séquence :

Complexity of Sequence

If, for some $n \in \mathbb{N}$, an algorithm has blocks $B_1, B_2, \ldots, B_n$ in sequence and, on input I to the algorithm, block B_i executes the key operations at most $F_i(B_i)$ times for $1 \leq i \leq n$, then the entire algorithm executes the key operations at most

$$F_1(B_1) + F_2(B_2) + \cdots + F_n(B_n)$$

times.

Exemple 1

- Nombre de comparaisons :
  - 1 + 1 + 1 = 3
- Complexité :
  - $O(1)$

Exemple 2

Algorithm: Ordering Three Values II

INPUT: Distinct values for three variables A, B, and C
OUTPUT: The list of letters ordered by increasing value

```
if (A > B or B > C) then   /* list is out of order, so */
    if (A > B) then Swap(A, B)
    if (B > C) then Swap(B, C)
    if (A > B) then Swap(A, B)
print A, B, C
```

Exemple 2

- Pour une sélection :

Complexity of Selection

Let $n \in \mathbb{N}$. Let a step of an algorithm be to make a selection for execution of one of n blocks, $B_1, B_2, \ldots, B_n$, and, on input I, where each B_i executes the key operations at most $F_i(B_i)$ times for $1 \leq i \leq n$ and it then takes $G(I)$ executions of the key operations to make the selection of the block to execute. Then, the key operations are executed at most

$$G(I) + \max\{F_1(B_1), F_2(B_2), \ldots, F_n(B_n)\}$$

times.

Exemple 2

- Nombre de comparaisons :
  - 2 + Max { 0, 1 + 1 + 1 } = 5
- Complexité :
  - $O(1)$

Exemple 3

Algorithm: Ordering Three Values III

INPUT: Distinct values for three variables A, B, and C
OUTPUT: The list of values in increasing order

```
1.  if (C < A and C < B) then
2.      if (A < B) then                 /* C < A < B, so */
3.          Swap(A, B)
4.          Swap(A, C)
5.      else                            /* C < B ≤ A, so */
6.          Swap (C, A)
7.  else
8.      if (B < A and B < C) then
9.          if (A < C) then             /* B < A < C, so */
10.             Swap (A, B)
11.         else                        /* B < C ≤ A, so */
12.             Swap (A, B)
13.             Swap (B, C)
14.     else
15.         if (C < B) then             /* A ≤ C < B, so */
16.             Swap (B, C)
17. print A, B, C
```


Exemple 3

- Nombre de comparaisons :
  - 2 + Max { 1, 2 + Max (1 , 1) } = 5
- Complexité :
  - $O(1)$
- Remarque :
  - En comptant les échanges : maximum 2, alors que maximum 3 pour les versions précédentes !

Exemple 4

Algorithm: BubbleSort I

INPUT: A list of N distinct integers $List[1], \ldots, List[N]$
OUTPUT: The same integers rearranged so that the values of $List[1], \ldots, List[N]$ are in increasing order

BubbleSort-I(*List*, N)
1. for $Pass = 1$ to $N - 1$ do
2. for $Position = 1$ to $N - 1$ do
3. if ($List[Position] > List[Position + 1]$) then
4. *Swap* ($List[Position]$, $List[Position + 1]$)

Exemple 4 - Répétition

Complexity of Repetition

If an algorithm contains a loop B, where:

- On input I, the loop is executed at most n times for some $n \in \mathbb{N}$, and
- On input I, for $1 \leq i \leq n$, the ith execution of the body of the loop executes the key operations at most $F_i(B_i)$ times, and
- The test of whether to stop or to execute the loop another time executes the key operations at most c times,

then the number of times the entire algorithm executes the key operations during the loop is at most

$$\sum_{i=1}^{n}(c + F_i(B_i)) + c$$

Exemple 4

- Nombre de comparaisons :
  - Boucle intérieure : N - 1
  - Boucle extérieure : $(N-1) \times (N-1)$
- Complexité :
  - $O(N^2)$

Exemple 5

Algorithm: BubbleSort II (slightly optimized)

INPUT: A list of N distinct integers $List[1], \ldots, List[N]$
OUTPUT: $List[1], \ldots, List[N]$, with values in increasing order

```
BubbleSort-II(List, N)
1. for Pass = 1 to N − 1 do
2.         Limit = N − Pass
3.         for Position = 1 to Limit do
4.                 if (List[Position] > List[Position + 1])
5.                         Swap(List[Position], List[Position + 1])
```

Exemple 5

- Nombre de comparaisons :
  - Boucle intérieure : N - 1
  - Boucle extérieure :
    $(N-1) + (N-2) + \ldots + 2 + 1 = N \times (N-1) / 2$
- Complexité :
  - $O(N^2)$