CSI 3505 / Automne 2018: Conception et Analyse des Algorithmes I.

- CONCEPTION ET ANALYSE DES ALGORITHMES (3 h.p.s.s. - 3 cr.)
  - Analyse des cas moyens et du pire cas. Analyse de la complexité. Notations asymptotiques et classes de complexité de base. Techniques de conception d'algorithmes: exhaustive, diviser pour régner, programmation dynamique glouton, retour arrière. Complexité computationnelle de problèmes: arguments de borne inférieure. Classes P, NP, et NP complet; traitement des problèmes NP complet.
  - *Préalables*: CSI2510 ou CSI2501 (pour les étudiants en spécialisation mathématiques seulement: CSI2510, MAT2541 ou MAT2543).

- PROFESSEUR: Dr. Nejib Zaguia
  - SITE 5031, 562-5800 ext.:6782
  - zaguia@site.uottawa.ca
  - Heures de bureau: Mardi 12:00-13:00

- HORAIRE DU COURS:
  - Mardi 13:00 – 14:30, LMX 405
  - Jeudi 11:30 – 13:00, LMX 407

- MANUEL:

  "Foundations of Algorithms Using C++ Pseudocode" (Fifth Edition), R. Neapolitan, K. Naimipour, Jones and Bartlett Publishers.

- Les notes de cours seront disponibles sur Brightspace

EVALUATION:

- Il y aura 4 devoirs, un examen mi-session et un examen final.

- Schémas d'évaluation:

  Devoirs : 25% (il y aura 4 devoirs. Pas de retard accepté)

  Examen mi-session : 25% (aura lieu en classe **le jeudi 18 octobre**, livre fermé)

  Examen final : 50%

Pour passer le cours il faut obtenir au moins une moyenne de 50% à l'examen final et l'examen mi-session (au moins 37.5 sur 75)

- Objectifs du cours:
  - Apprendre différentes techniques de conception d'algorithmes et reconnaître les situations dans lesquelles les appliquer.
  - Comprendre l'importance de l'efficacité des algorithmes.
  - Apprendre comment analyser et comparer des algorithmes suivant leurs temps d'exécution et leurs besoins en mémoire.

Plan du Cours

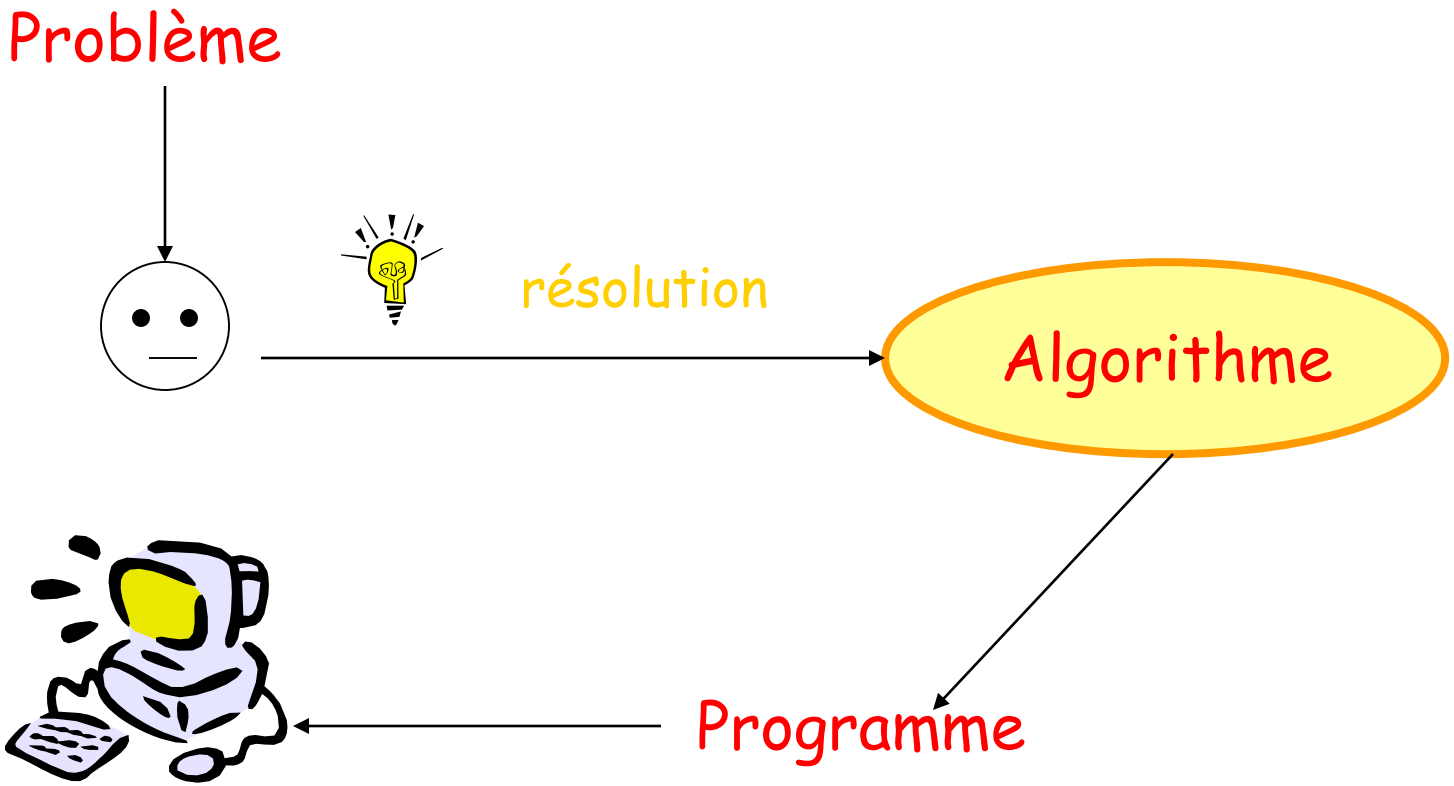
- Introduction (Chapitre 1)
  - Algorithmes?
  - Complexité
  - Analyse
  - Ordre et Notations asymptotiques.
  - Rappel mathématique
- Techniques de résolution: Diviser pour régner (Chapitre 2)
- Techniques de résolution: Programmation dynamique(Chapitre 3)
- Techniques de résolution: Algorithmes voraces(Chapitre 4)
- Techniques de résolution: Algorithmes retour arrière (Chapitre 5)
- Introduction a la théorie de Complexité du calcul(Chapitre 9)

Algorithme?

- Un processus de calcul fini qui donne la solution d'un problème posé. En plus ce processus doit être entièrement défini, au sens où à chaque étape du calcul, la suivante en découle sans ambiguïté.

- En général, l'exécution d'un algorithme ne doit pas laisser place à l'interprétation, à l'intuition ni à la créativité.

CSI 3505
Introduction



CSI 3505
Introduction

- **Problèmes et Exemplaires:**
  - **Problèmes**
    - *Données*: les paramètres du problème.
    - *Résultats*: la réponse à la question demandée
  - **Solution**
    - *suite d'instructions qui permet de trouver le résultat.*

CSI 3505
Introduction

- *Problème 1: Tri:*
  - Trier un ensemble de n nombres. "Mettre ces nombres dans un ordre croissant, ou décroissant".
  - Un exemplaire du problème du tri: Trier les 7 nombres
    12 4 67 2 10 23 8
- *Problème 2:*
  - Multiplier deux entiers positifs avec papier et crayon
  - Un exemplaire du problème 2: Multiplier les deux entiers 45 et 19.

En général, un problème consiste en une collection infinie d'exemplaires

- *Problème 2:*

  Multiplier deux entiers positifs avec papier et crayon.

Solution 1 « ou Algorithme 1 »:

i) Multiplier successivement le multiplicande par chaque chiffre du multiplicateur, de la droite vers la gauche et, en tenant compte des décalages nécessaires,

ii) Terminer avec une addition

```
  45
  19
   4
 ---
 405
 45
 ---
 855
```

CSI 3505
Introduction

- *Solution 2 « ou Algorithme 2 »: Multiplication a la russe*

i) Écrivez le multiplicateur et le multiplicande l'un a coté de l'autre

ii) Formez une colonne en dessous de chacun des opérandes en itérant la règle suivante jusqu'a ce que le nombre sous le multiplicateur soit égal a 1:

- divisez par deux le nombre sous le multiplicateur (sans tenir compte du reste)
- doublez le nombre sous le multiplicande.

iii) Rayez les nombres de la colonne du multiplicande correspondant a un nombre pair sous le multiplicateur.

iv) Additionner les nombres restants sous le multiplicande

45	19
~~22~~	~~38~~
11	76
5	152
~~2~~	~~304~~
1	608

19+76+152+ 608 = 855

CSI 3505
Introduction

- Normalement un algorithme doit fonctionner sans défaillance sur tous les exemplaires du problème qu'il prétend solutionner.

- Donc un algorithme donné est invalide s'il existe un exemplaire du problème pour lequel il ne fonctionne pas. Il est beaucoup plus difficile de démontrer l'exactitude d'un algorithme.

- Des différents algorithmes peuvent résoudre un même problème et que des différents programmes peuvent servir à décrire un même algorithme.

CSI 3505
Introduction

Efficacité des algorithmes

- Pour un problème donné, on peut trouver plusieurs algorithmes pour le résoudre. Comment déterminer lequel est préférable?

- *L'approche empirique:* consiste a programmer ces algorithmes et à les essayer sur divers exemplaires à l'aide d'un ordinateur.

- *L'approche théorique:* consiste à déterminer mathématiquement la quantité de ressources (temps, espace, etc.) nécessaire aux algorithmes en fonction de la taille des exemplaires considérés.

- Il est clair que la quantité de ressources nécessaire aux algorithmes dépendra de la taille des exemplaires.

Pour simplifier notre étude on suppose toujours que les étapes de calcul consistent d'opérations élémentaires qui nécessite une unité de temps.

CSI 3505
Introduction

- Problème du tri: Trier un ensemble de n nombres.

  "Mettre ces nombres dans un ordre croissant, ou décroissant".

- Un exemplaire du problème du tri: Trier les 7 nombres

  12 4 67 2 10 23 8

CSI 3505
Introduction

- Une solution: algorithme du tri par insertion:

  12 4 67 2 10 23 8.

- A chaque étape on insert un nouveau élément dans la bonne position dans la sous liste triée.
- comparer 12 et 4 pour obtenir
  - **4** **12** 67 2 10 23 8
- comparer 67 et 12 pour obtenir
  - **4** **12** **67** 2 10 23 8
- comparer 2 et 67 puis 2 et 12 puis 2 et 4 pour obtenir
  - **2** **4** **12** **67** 10 23 8
- comparer 10 et 67, puis 10 et 12, puis 10 et 4 pour obtenir
  - **2** **4** **10** **12** **67** 23 8
- comparer 23 et 67, puis 23 et 12 pour obtenir
  - **2** **4** **10** **12** **23** **67** 8
- comparer 8 et 67, puis 8 et 23, puis 8 et 12, puis 8 et 10, puis 8 et 4 pour obtenir la liste triée
  - **2** **4** **8** **10** **12** **23** **67**.

- On a fait 15 comparaisons pour trier cette liste par insertion.

CSI 3505
Introduction

- La liste 3 5 8 13 67 78 88
  demande 6 comparaisons pour être trier par insertion.

- La liste 88 78 67 13 8 5 3
  demande 21 comparaisons pour être trier par insertion.

- Pour une liste de n nombres, on a besoin au plus de

$$1 + 2 + 3 + \ldots + (n-1) = n(n-1)/2$$

  comparaisons pour le tri par insertion.

- Et il existe des exemplaires qui nécessitent ce nombre de comparaisons.

CSI 3505
Introduction

La taille d'un exemplaire correspond formellement au nombre de bits requis pour le représenter dans l'ordinateur, à l'aide d'un encodage raisonnablement compact.

Exemple 2.

- Un polynôme de degré $n \geq 0$ est une expression de le forme

  $$P(x)=a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

  où $a_n, a_{n-1}, \ldots, a_0$ sont des nombres réels.

- Les polynômes sont important: approximation des fonctions continues, fonctions génératrices 'technique puissante en combinatoire"…

Problème:

Étant donné un polynôme P(x) et une valeur x, évaluer P(x)?

CSI 3505
Introduction

Exemplaire du Problème:

Évaluer P(3), où $P(x)= 2x^3 + 4x^2 +5x +2$.

En utilisant l'algorithme naïf

- $2x^3$ ---------- 3 multiplications
- $4x^2$ ---------- 2 multiplications
- $5x$ ---------- 1 multiplication
- $2x^3 + 4x^2$ ---------- 1 addition
- $(2x^3 + 4x^2) + 5x$ ---------- 1 addition
- $P(x)$ ---------- 1 addition

- En tout, on a besoin de 6 multiplications et 3 additions. On suppose que les autres opérations nécessaire pour effectuer l'algorithme sont négligeable en terme de temps d'exécution.

CSI 3505
Introduction

- En général, le calcul de $P(x)=a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0$ demande

  $n + (n-1) + ... + 2 + 1 = n(n+1)/2$ multiplications et

  n additions.

  Ceci fait un total de

  $n(n+3)/2 = 1/2\ n^2 + 3/2\ n$ opérations.

- Une méthode plus intelligente:

Horner 1819 (en fait ça remonte à Newton 1711).

$P(x)= 2x^3 + 4x^2 +5x +2 = ((2x +4)x +5)x +2$

6 opérations sont suffisante au lieu de 9 opérations.

$P(x)=a_n x^n + a_{n-1} x^{n-1} + ... + a_1 x + a_0 = (...((a_n x + a_{n-1})x + a_{n-2})x+ ...)x + a_1)x + a_0$.

On a besoin de 2n opérations.

Peut-on faire mieux?

CSI 3505
Introduction

Qu'est-ce qu'une opération élémentaire?

- Ceci dépend du problème étudié: opération arithmétique, comparaison de deux nombres, etc. C'est l'opération fondamentale qui permet d'exécuter l'algorithme.

Pourquoi chercher des algorithmes efficaces?

- Un algorithme qui n'est pas efficace ne peut pas résoudre des exemplaires de grande taille même si la rapidité des ordinateurs est multipliée par des milliards.

CSI 3505
Introduction

- **Pourquoi analyser les algorithmes?**
- Ça permet de prévoir le comportement de l'algorithme, et surtout le temps d'exécution et la validité, sans l'exécuter sur un ordinateur. Autrement on ne sera jamais sûr de la validité indépendamment des tests qu'on fait.
- De plus, il est plus simple d'avoir une mesure de la rapidité indépendamment du programme et de l'ordinateur.
- Ça permet de comparer deux algorithmes qui résolvent le même problème.
- En général il est difficile d'avoir une information exacte sur le comportement de l'algorithme, on cherche surtout une approximation..

CSI 3505
Introduction

Comment analyser un algorithme dans le pire cas?

Des exemples concrets d'algorithmes pour:

- Problème de recherche
  - Algorithme de recherche séquentielle
  - Algorithme de recherche binaire
- Le calcul de la suite de Fibonacci
- Le calcul du PGCD
- La multiplication matricielle
- Problème du commis voyageur
- La recherche binaire récursive

CSI 3505
Introduction

- Analyse: Pire Cas
  - Algorithme A pour résoudre un problème P
  - Taille de I 'exemplaire de P « des données »: n
  - D_n: ensemble de tous les exemplaires de taille n pour le problème P
  - t(I): nombre d'opérations élémentaires faites par l'algorithme A pour I 'exemplaire I
  - $W_A(n)$: performance dans le cas pire de l'algorithme A avec des exemplaires de taille n

$$W_A(n) = \max \{ t(I) \text{ tel que } I \in D_n \}$$

CSI 3505
Introduction

$$W_A(n) = k$$

Signifie que, pour n 'importe quel exemplaire de taille n, l'algorithme A utilise au plus k opérations élémentaires.

De plus il existe au moins un exemplaire pour lequel l'algorithme A utilise en effet k opérations élémentaires.

Borne supérieure

Recherche d'un élément x dans une liste. Algorithme 1.1 - recherche séquentielle

On traverse le tableau a partir du premier élément jusqu'a ce que l'on trouve x, Sinon on arrive a la fin du tableau.

taille du problème : n
opérations élémentaires: Comparaisons
Pire Input: x est le dernier élément de la liste ou x n 'est pas dans la liste

CSI 3505
Introduction

Algorithme 1.1

```
location := 1
 Tant que (location ≤ n) et (S[location] ≠ x)
     location := location+1
 si location > n alors
     location := 0
```

Pire cas: Si x est le dernier élément de la liste ou si x n'est pas dans la liste alors on a besoin de n comparaisons.

CSI 3505
Introduction

- recherche binaire
  - Ca ne marche que si la liste de départ est triée

- S[bas]----------S[haut]

  - comparer l'élément du milieu «position Mil=(haut+bas) /2 » avec x
    - si x=S[Mil] alors retourner Mil
    - Si x < S[Mil] alors diviser l'intervalle de recherche en deux parties et garder la moitié S[bas] ---S[Mil]-1
    - Sinon garder la moitié S[Mil]+1 ---S[haut]
  - continuer jusqu'a ce que x est trouvé ou l'intervalle est vide

CSI 3505
Introduction

Taille du problème: n (nombre d'éléments dans la liste)
Opérations élémentaires: Comparaisons

Pire Cas: x n'est pas dans la liste

Il y a une comparaison par itération
Soit $n = 2^k$

Itération	dimension de l'intervalle
1	n
2	n/2
3	n/4
4	n/8
…	…
i	$n/2^{i-1}$
…	…
k+1	$1 = n/2^k$

$$W(n) = (k+1) * 1 = k+1$$

Nombre d 'itérations

Nombre de comparaisons par itération

Puisque $1 = n/2^k$ alors $k = \log n$ et donc

$$W(n) = \log n + 1$$

Comparer les deux algorithmes de recherche:

Taille	Séquentielle	Binaire
128	128	8
1024	1024	11
1,048,576	1,048,576	21
4,294,967,296	4,294,967,296	33

La recherche binaire est beaucoup plus rapide mais il faut que la liste soit déjà TRIÉE

CSI 3505
Introduction

Peut on faire mieux que la recherche séquentielle en ordonnant la liste puis en faisant la recherche binaire?

- Algorithme A:
  - Ordonner (trier) la liste
  - Faire une recherche binaire

→ Avec « Tri par échange» (Algo. 1.3)

Tri par échange:

```
Pour i=1 jusqu'à n-1
        Pour j=i+1 jusqu'à n
                si S[j]<S[i] alors
                        échanger S[i] et  S[j]
```

CSI 3505
Introduction

Taille du problème: n

Opérations élémentaires: comparaisons

Pire Input: Tous

i	comparaisons
1	n-1
2	n-2
…	…
…	…
n-1	1

Complexité du tri par échange:

$$W(n) = \sum_{i=1}^{n-1} k = \frac{n(n-1)}{2}$$

CSI 3505
Introduction

Complexité du tri par échange: $W(n) = \sum_{i=1}^{n-1} k = \frac{n(n-1)}{2}$

Donc la complexité de l'algorithme A:

- Ordonner (trier) la liste en utilisant Tri par échange
- Faire une recherche binaire

$$W(n) = \frac{n(n-1)}{2} + \log n + 1$$

Beaucoup plus mauvais que la recherche séquentielle !!!

CSI 3505
Introduction

Le calcul de la suite de Fibonacci

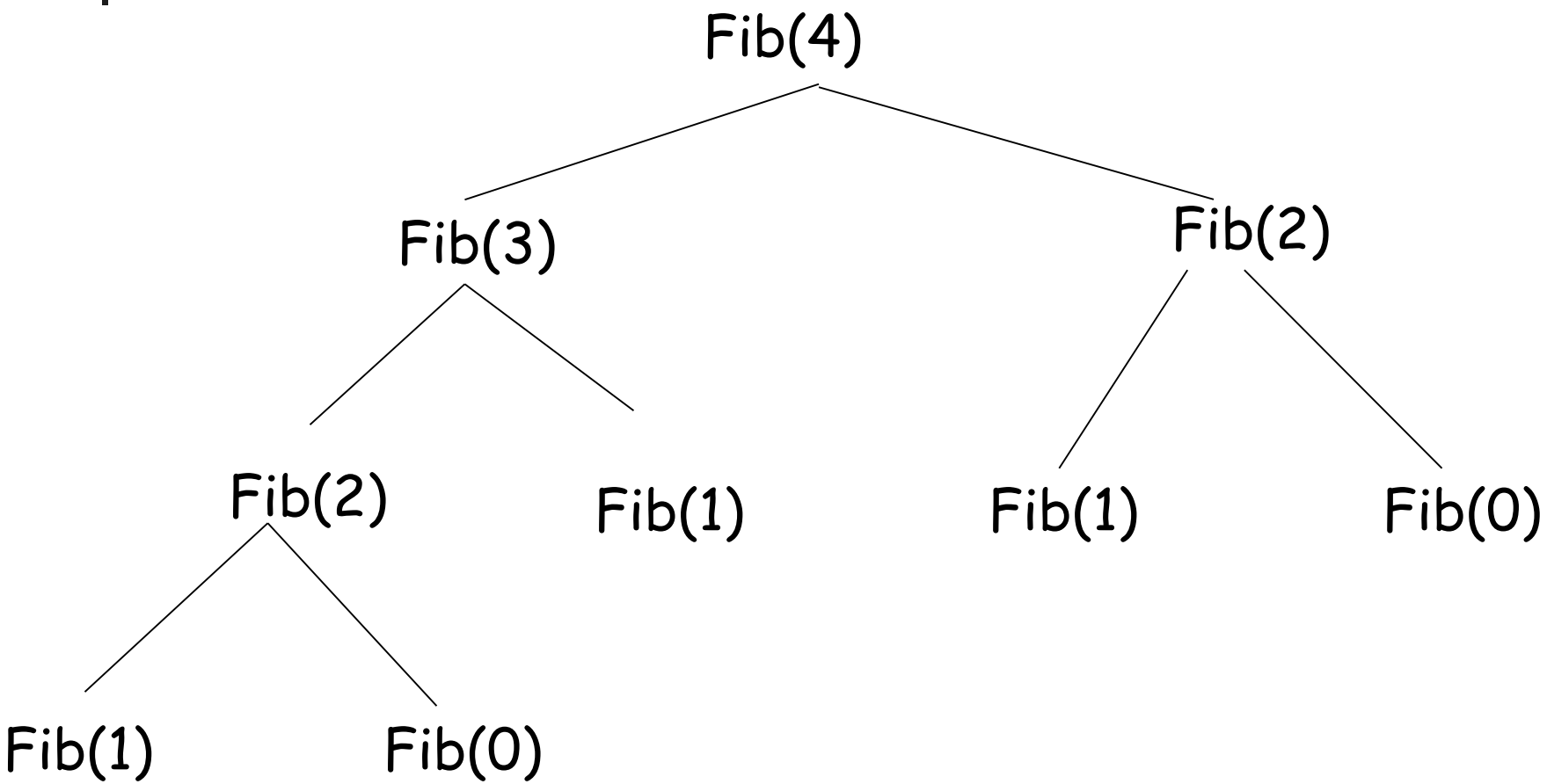
$f_0=0$; $f_1=1$ et $f_n = f_{n-1} + f_{n-2}$ pour $n \geq 2$.

$f_0=0$; $f_1=1$; $f_2=1$; $f_3=2$; $f_4=3$; $f_5=5$; $f_6=8$; $f_7=13$; $f_8=21$; $f_9=34$; …

```
fonction fib1( n)
        si n < 2 alors retourner n
        sinon  retourner fib1(n-1)  + fib1 (n-2).
```

Cet algorithme est très inefficace: il recalcule maintes fois les mêmes valeurs.

CSI 3505
Introduction



CSI 3505
Introduction

```
fonction fib2( n)
        i ← 1;  j← 0
        pour  k← 1 jusqu'à   n   faire
                    j← i+j
                    i← j-i
          retourner  j.
```

- Le temps pris pour calculer f_n est d'ordre n.
- Il existe d'autre algorithmes "fib3" plus compliqués et dont le temps de calcul de f_n est d'ordre log n.

CSI 3505
Introduction

n	10	20	30	50	100	10^4	10^6	10^8
fib1	8 ms	1s	2 min	21 jours				
fib2	1/6ms	1/3 ms	1/2 ms	3/4 ms	11/2 ms	150 ms	15 s	25 min
fib3	1/3ms	2/5 ms	1/2 ms	1/2 ms	1/2 ms	1 ms	11/2 ms	2 ms

CSI 3505
Introduction

- *Le calcul du déterminant d'une matrice.*
  - M=(ai,j) i≤n, j≤n
  - M[i, j] dénote la matrice (n-1) x (n-1) obtenue de M en supprimant sa colonne i et sa rangée j.
  - Une définition récursive du déterminant:

$$\det(M) = \sum_{1 \le j \le n} (-1)^{j+1} a_{1,j} \det(M[1, j])$$

- L'utilisation directe de cette formule conduit à un algorithme prenant un temps de l'ordre n!. Ceci est encore pire que 2^n.

CSI 3505
Introduction

- récursif Gauss-Jordan

5x5	20 sec	10x10	1/100 s
10x10	10 min	100x100	5 1/2 s
20x20	10 millions d'années		

- Il existe un autre algorithme récursif d'ordre $n^{2.81}$

CSI 3505
Introduction

- Peut être, l'algorithme le plus connu et celui d'Euclide (date de l'antiquité) qui permet de calculer le PGCD de deux nombres.

- fonction pgcd(m, n)
  i ← min (m, n) + 1
  répéter i ← i-1 jusqu'à ce que i divise m et n sans reste
  retourner i.

- Cet algorithme prend un temps dans l'ordre de la différence entre le petit de ses deux arguments et leur plus grand commun diviseur. Dans le cas où m et n sont du même ordre de grandeur et sont premier entre eux, il prend donc un temps dans l'ordre de n.

CSI 3505
Introduction

Le PGCD de *m* et *n (m≥n)* est égal au *pgcd* de *n* et du reste de la division entière de *m* par *n: pgcd(m, n) = pgcd(n, m mod n)*

n	*m*	*t*
102	30	12
30	12	6
12	6	0
6	0	

pgcd(102,30) = pgcd(30,12)

pgcd(30,12) = pgcd(12,6)

pgcd(12,6) = pgcd(6,0)

pgcd(6,0) = 6

CSI 3505
Introduction

- fonction Euclide(m, n)

  tantque m>0 faire

  t ← n mod m

  n ← m

  m ← t

  retourner n.

- Cet algorithme prend un temps dans l'ordre du logarithme des ses arguments, ce qui est beaucoup plus rapide que l'algorithme précédent.

CSI 3505
Introduction

Fusion de deux listes triées (Alg. 2.3): Soient U et V deux listes triées. Fusionner-les en une troisième liste S

Comparer U[i] et V[j] et mettre le plus petit nombre en S
Incrémenter l'index de la liste ou le plus petit se trouvait et incrémenter l'index de S
Lorsque l'on arrive a la fin d'une liste, copier ce qui reste de l'autre liste en S.

2	4	7	8	9	25	30	32	34	35	42	47	54

S

4	7	8	25	30	32	35	47

U

2	9	34	42	54

V

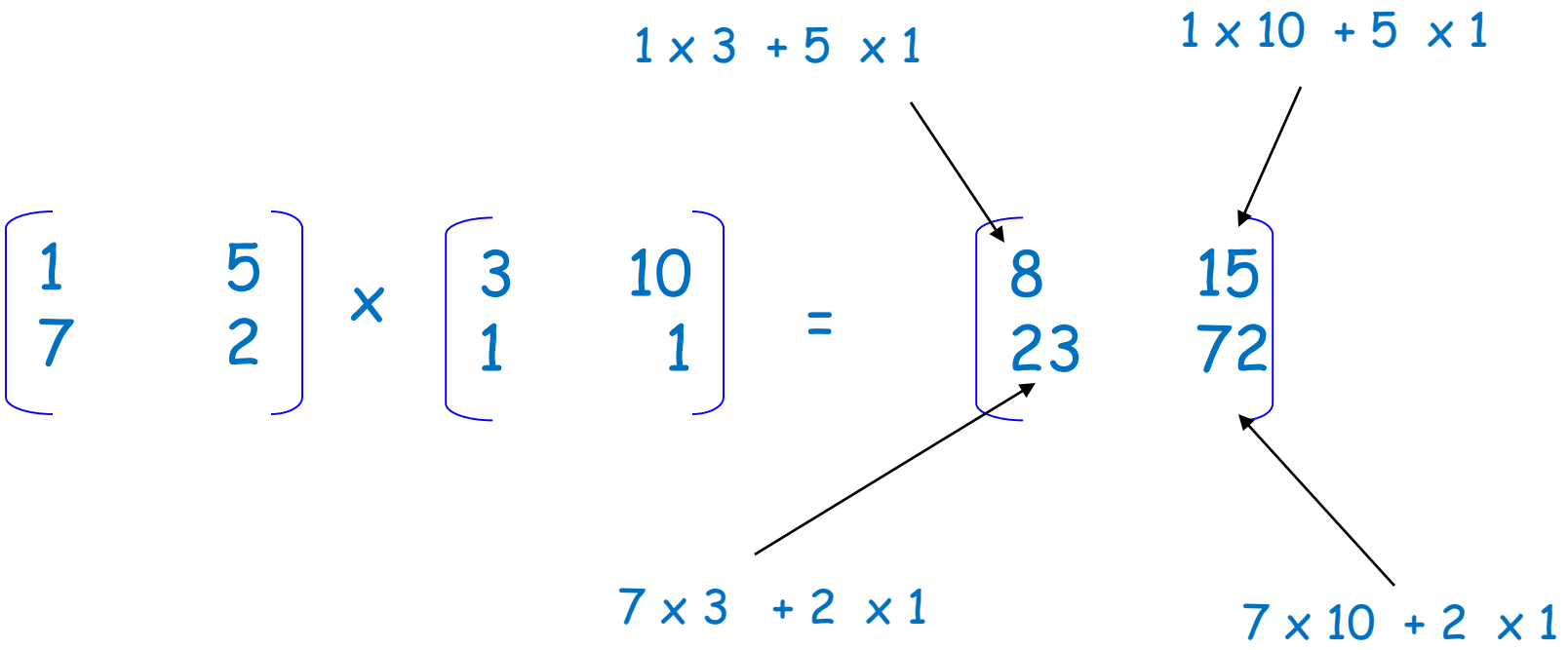
Taille du problème: n = h+m (Taille de U: h, Taille de V: m, Taille de S: h + m)
Opérations élémentaires: comparaisons
Pire input: quand les derniers éléments des listes sont le plus grand et le deuxième plus grand.

$W(n) = n-1$

CSI 3505
Introduction

Multiplication matricielle: Multiplier deux matrices A et B de taille nxn.

A * B = C



CSI 3505
Introduction

Algorithme 1.4

```
for i=1 to n do
  for j=1 to n do
      C[i,j]:=0
      for k=1 to n do
      C[i,j]:=  C[i,j]+A[i,k]*B[k,j]
      end
  end
end
```

CSI 3505
Introduction

A * B = C (les matrices A et B de taille nxn)

Taille de l'input: n
Opérations élémentaires: multiplications des nombres
Pire input: le même nombre d'opérations élémentaires pour tous les inputs de taille n.

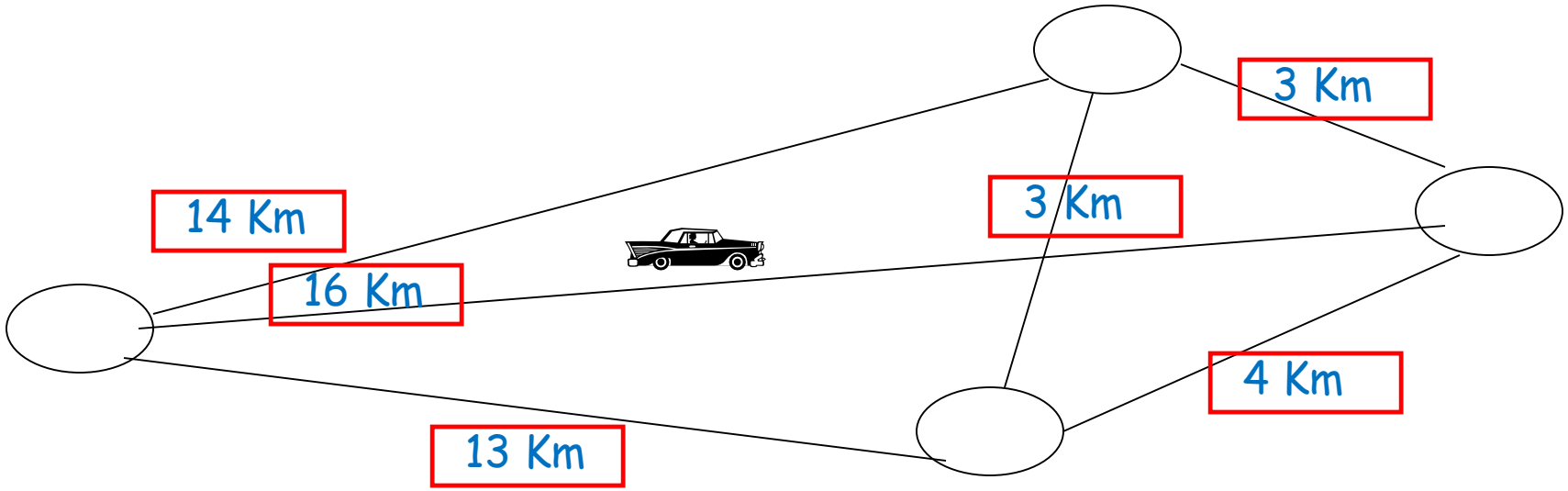
L'algorithme a besoin de n multiplications pour calculer chaque entrée de C. Il y a n x n entrées.

$W(n) = n^3$

CSI 3505
Introduction

Le problème du commis voyageur: Travelling Salesman Problem (TSP)

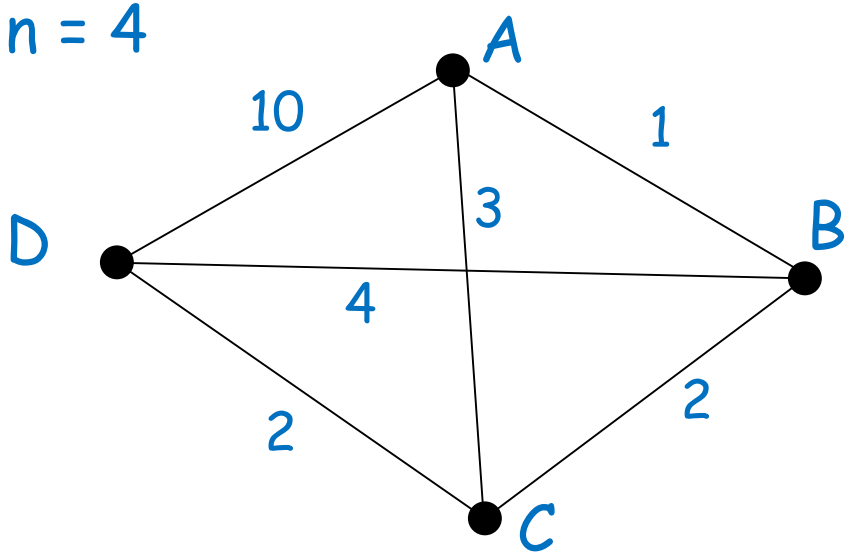
Nous connaissons les distances directes entre un certain nombre de villes. Le commis voyageur doit quitter l'une de ces villes, visiter chaque autre ville une et une seule fois, et revenir au point de départ en minimisant la distance totale parcourue.



CSI 3505
Introduction

TSP: Soit un graphe complet et pondéré avec n sommets, trouver un tour de poids minimal.

n = 4



	A	B	C	D
A	∞	1	3	10
B	1	∞	2	4
C	3	2	∞	2
D	10	4	2	∞

Les tours sont représenter avec les permutations cycliques de A,B,C,D.

A B C D = 1+2+2+10

B A C D = 1+3+2+4

CSI 3505
Introduction

Algorithme exhaustif: générer toutes les permutations cycliques
et choisir celle qui a la longueur minimale

Données: n ≥ 3 et une matrice H de taille n x n avec des poids non-négatifs
Résultat: un tour de longueur minimale

```
min := + ∞
pour chaque permutation cyclique π de 1,2 …n
    coût:= coût associe a π
    if coût < min then
        min := coût
        MeilleurTour:= π
```

CSI 3505
Introduction

Taille du problème: n le nombre de nœuds
Opérations élémentaires: le nombre d'itérations dans l'algorithme
Pire input: pour tous les exemplaires, on a besoin du même nombre d'itérations

Il y a n! permutations de (1, 2, …, n). Un tour est représenté par n permutations.

1,2,3,4
2,3,4,1
3,4,1,2
4,1,2,3

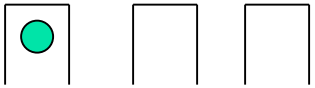
$$\frac{n!}{n}$$

De plus: 1,2,3,4 représente le même tour que 4,3,2,1

Donc le nombre de différents tours pour n nœuds

$$W(n) = \frac{1}{2}\, n!/n = (n-1)!\,/\,2$$

Analyse des algorithmes: Cas Moyen

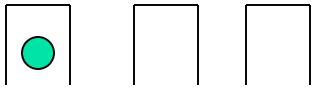


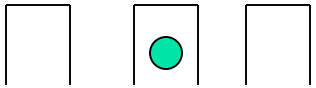
Sous quelle tasse se trouve ● ?

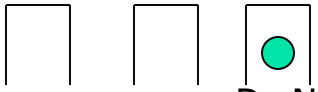
Algorithme:

regarder sous la première tasse
regarder sous la deuxième tasse
regarder sous la troisième tasse

Inputs possibles

I_1 

I_2 

I_3 

On suppose que chaque input a la même probabilité

CSI 3505
Introduction

$P(I_1) = 1/3$ Probabilité pour que la solution est tasse 1
$P(I_2) = 1/3$ Probabilité pour que la solution est tasse 2
$P(I_3) = 1/3$ Probabilité pour que la solution est tasse 3

$t(I_1) = 1$: temps demandé pour exemplaire de type 1
$t(I_2) = 2$ temps demandé pour exemplaire de type 2
$t(I_3) = 3$ temps demandé pour exemplaire de type 3

$$A(n) = \sum_{i=1}^{3} p(I_i)\, t(I_i) \quad = 1/3\ (1 + 2 + 3) = 2$$

CSI 3505
Introduction

A(n): temps moyen
D(n): ensemble de tous les inputs de taille n
P(I): probabilité d'avoir l'input I
t(I): temps demandé par l'input I (nombre d'opérations élémentaires)

$$A(n) = \sum_{I \in Dn} p(I)\, t(I)$$

$$(\Sigma\, p(I) = 1)$$

CSI 3505
Introduction

Grouper les input sur lesquels l'algorithme a le même comportement

De telle façon que t(I) est le même pour chaque $I \in I_i$

$$D_n = I_1 \cup I_2 \cup \ldots \cup I_k$$

$P(I_i)$: probabilité d'avoir un des inputs du groupe I_i
$t(I_i)$: temps nécessaire pour n'importe quel input du groupe I_i

$$A(n) = \sum_{i=1}^{k} p(I_i)\, t(I_i)$$

CSI 3505
Introduction

- Exemple: Recherche séquentielle (1.1)
  - Pire Cas: W(n) = n
  - Meilleur cas: W(n) = 1
- Cas moyen: x est dans la liste
  - On suppose que tous les éléments de la liste sont différents
  - Partition de $D_{n:}$

    $I_i = \{I \in D_n : x \text{ est dans la position i dans la liste}\}$

Les positions sont équiprobables

CSI 3505
Introduction

Les positions sont équiprobables

$P(I_1) = 1/n$
$P(I_2) = 1/n$
$\ldots \quad \ldots$
$P(I_n) = 1/n$

$t(I_1) = 1$
$t(I_2) = 2$
$\ldots \quad \ldots$
$t(I_n) = n$

$$A(n) = \sum_{i=1}^{n} p(I_i)\, t(I_i) = 1/n \sum_{i=1}^{n} i = 1/n\,(\, n\,(n+1)/2\,) = (n+1)/2$$

CSI 3505
Introduction

- Si x est dans la liste: $A(n) = (n+1)/2$
- Si x n'est pas nécessairement dans la liste:
  - q: probabilité que x est dans la liste
  - I_{n+1}: Inputs de D_n pour lesquels x n'est pas dans la liste

CSI 3505
Introduction

I_{n+1}: Input de D_n pour lequel x n'est pas dans la liste

- Donc $p(I_i) = q/n$ pour $i \leq n$ et $p(I_{n+1}) = 1-q$

- $A(n) = \sum_{1 \leq i \leq n+1} p(I_i)\, t(I_i)$

$= q/n[\sum_{1 \leq i \leq n} i\ ] + (1-q)n$

$A(n) = q(n+1)/2 + (1-q)n$

- **Ordre: Notations asymptotiques**

Problème P.	Problème Q	Problème R.
Algorithme A1 $W_{A1}(n) = 2n$	Algorithme A1 $W_{A1}(n) = 2000n$	Algorithme A1 $W_{A1}(n) = 1/100\ n$
Algorithme A2 $W_{A2}(n) = 4.5\ n$	Algorithme A2 $W_{A2}(n) = 2\ n^2$	Algorithme A2 $W_{A2}(n) = \log n$

Pour chacun de ces problèmes, quel est le meilleur algorithme ????

CSI 3505
Introduction

Ordre de complexité	10	20	30	40	50	60
n	.00001 secondes	.00002 secondes	.00003 secondes	.00004 secondes	.00005 secondes	.00006 secondes
n^2	.0001 secondes	.0004 secondes	.0009 secondes	.0016 secondes	.0025 secondes	.0036 secondes
n^3	.001 secondes	.008 secondes	.027 secondes	.064 secondes	.125 secondes	.216 secondes
n^5	.1 secondes	3.2 secondes	24.3 secondes	1.7 minutes	5.2 minutes	13.0 minutes
2^n	.001 secondes	1.0 seconde	17.9 minutes	12.7 jours	35.7 années	366 siècles
3^n	.059 secondes	58 minutes	6.5 années	3855 siècles	$2x10^8$ siècles	$1.3x10^{13}$ siècles

CSI 3505
Introduction

- **Ordre: Notations asymptotiques**
- L'analyse théorique de l'efficacité d'un algorithme se fait à une constante multiplicative pré.
- Le but est de ne pas tenir compte des variations introduites par un changement d'implantation, de langage de programmation ou d'ordinateur.

- **Pour un algorithme qui résout un problème de taille n dans 100n nombre d'étapes, on dira que la durée est approximativement n.**
- **Si ça étais $3n^3 + 25$, on dira approximativement n^3**

Ce genre d'approximation est très valable lorsque la valeur de n est grande.

On a besoin de quelques notations mathématiques pour simplifier l'étude de l'efficacité des algorithmes.

La notation "l'ordre de "

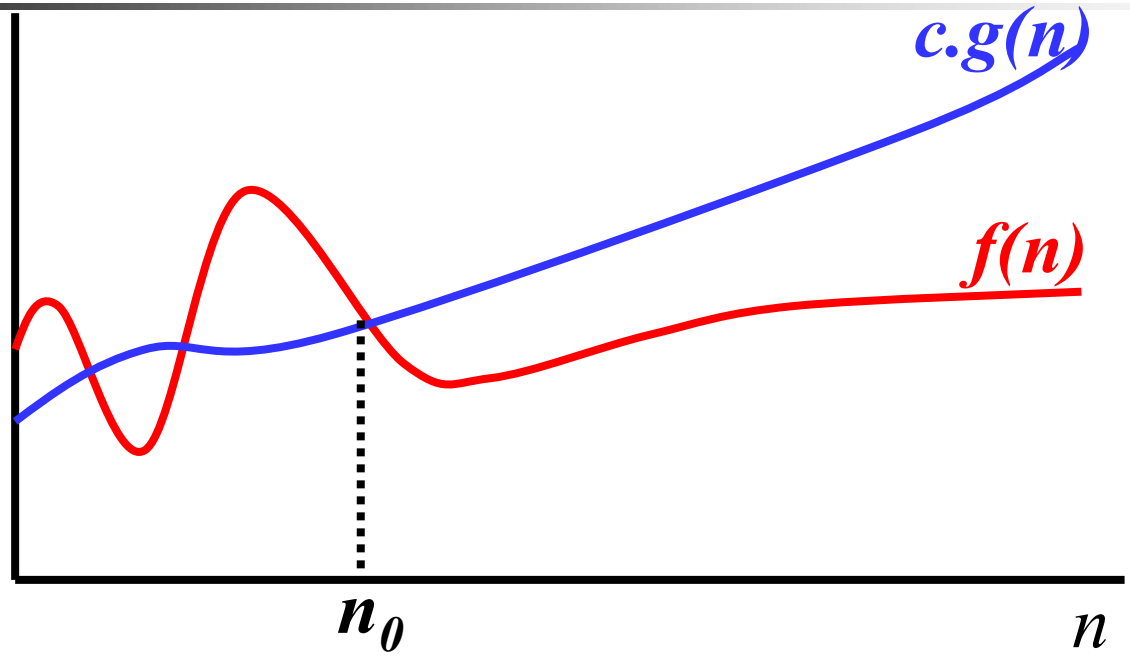
- N l'ensemble des entiers naturels,
- R l'ensemble des nombres réels, et R^* = {x dans R telque x>0}.

Soit f: $N \rightarrow R^*$ et g: $N \rightarrow R^*$ deux fonctions quelconques. On dira que g est dans O(f(n)) , ou aussi «g(n) est dans l'ordre de f(n)»,

s'il existe une constante c et un entier naturel n_0 tels que

$g(n) \leq c\, f(n)$ pour tout $n \geq n_0$.

CSI 3505
Introduction



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

Il existent deux constantes c et n_0 tel que
$0 \le f(n) \le c.g(n)$ pour $n \ge n_0$

CSI 3505
Introduction

- O(f(n)) est l'ensemble de toutes les fonctions g(n) bornées supérieurement par un multiple réel positif de f(n) a partir d'un certain seuil n_0.

- Pour des valeurs de n suffisamment grandes, le taux de croissance de g(n) est inférieur ou égal à celui de f(n)

CSI 3505
Introduction

Exemples:

- $f(0)=1, f(1)=4$, $f(n) = (n+1)^2$

  $(n+1)^2 \leq 4n^2$ pour tout $n \geq 1$, donc $f(n) \in O(n^2)$.

- $f(n) = 3n^3 + 2n^2$

  $f(n) \leq 5\, n^3$ pour tout $n \geq 1$, donc $f(n) \in O(n^3)$

- $3^n \notin O(2^n)$

Supposons que $3^n \leq c\, 2^n$ pour tout $n \geq n_0$. Alors $c \geq (3/2)n \rightarrow$ infini.

Impossible

CSI 3505
Introduction

- $f(n)=a_k n^k + a_{k-1}\, n^{k-1} + \ldots + a_1 n + a_0$

$f(n) \in O(n^k)$

$$f(n) \leq (|a_k| + |a_{k-1} / n | + \ldots + |a_0 / n^k |)\ n^k$$
$$\leq (|a_k| + |a_{k-1} | + \ldots + |a_0 |)\ n^k \qquad \text{pour tout } n \geq 1.$$

Un algorithme avec k étapes dont les durées sont
$c_1 n^{m1}$, $c_2 n^{m2}$, , ... , $c_k n^{mk}$
a une complexité $O(n^m)$ où $m=\max\{m_1, \ldots , m_k\}$.

CSI 3505
Introduction

O(1)	temps constant
O(log n)	temps logarithmique
O(n)	temps linéaire
O(n log n)	
$O(n^2)$	temps quadratique
$O(n^3)$	temps cubique
$O(2^n)$	temps exponentiel

CSI 3505
Introduction

Une fonction $g(n) \in \Omega(f(n))$ (est grand omega de f(n)) si et seulement si il existe une constante c et un entier naturel n_0 tels que $g(n) \geq c\, f(n)$ pour tout $n \geq n_0$.

$g(n) \in \mathbf{\Omega}(f(n))$: pour les valeurs de n suffisamment grandes, le taux de croissance de g(n) est plus grand ou égal à celui de f(n)

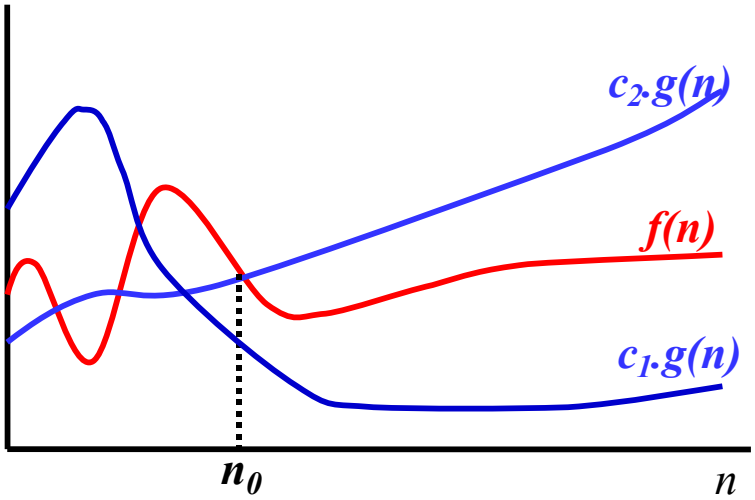
CSI 3505
Introduction

Une fonction $g(n) \in \Theta(f(n))$ (g(n) est Theta de f(n)) si elle vérifie à la fois

$$g(n) \in O(f(n)) \text{ et } g(n) \in \Omega(f(n))$$

$g(n) \in \Theta(f(n))$: Le taux de croissance asymptotique de g(n) est identique à celui de f(n)

CSI 3505
Introduction



$$f(n) = \Theta(g(n))$$

Il existent trois constantes c_1, c_2 et n_0 tel que

$$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad pour\ n \geq n_0$$

CSI 3505
Introduction

Utiliser la limite pour déterminer l'ordre.

$$\lim_{n \to \infty} g(n)/\, f(n) = \begin{cases} 0 \quad \text{alors } g(n) \in O(f(n)) \\ \qquad [\text{mais } g(n) \notin \Theta(f(n)] \\ c > 0 \quad \text{alors } g(n) \in \Theta(f(n)) \\ \infty \quad \text{alors } g(n) \in \Omega(f(n)). \\ \qquad [\text{mais } g(n) \notin \Theta(f(n)] \end{cases}$$

CSI 3505
Introduction

Théorème : Règle de L'Hôpital

Si f(x) et g(x) sont différentiable avec des dérivées f'(x) et g'(x) et si: $\lim_{x \to \infty} f(x) = \lim_{x \to \infty} g(x) = \infty$

alors

$$\lim_{x \to \infty} f(x)/g(x) = \lim_{x \to \infty} f'(x)/g'(x)$$

(Cf. Théorème 1.4 du texte.)

- Exemples:

$f(n) = n^3/2$ $\quad\quad$ $g(n) = 37\ n^2 + 120\ n + 17$

Montrer que $g(n) \in O(f(n))$ et que $f(n) \notin O(g(n))$

Règle de L'Hôpital

$$\lim_{n \to \infty} g(n)/f(n) = \lim_{n \to \infty} \frac{37\ n^2 + 120\ n + 17}{n^3/2}$$

$$= \lim_{n \to \infty} \frac{74\ n + 120}{3\ n^2/2} = \lim_{n \to \infty} \frac{74}{3\ n} = 0$$

CSI 3505
Introduction

Sans utiliser la règle de l'Hôpital

$$\lim_{n \to \infty} \frac{37\,n^2 + 120\,n + 17}{n^3/2} =$$

$$= \lim_{n \to \infty} \frac{37\,n^2}{n^3/2} + \frac{120\,n}{n^3/2} + \frac{17}{n^3/2}$$

$$= \lim_{n \to \infty} 74/n + 240/n^2 + 34/n^3 = 0$$

Par contre $f(n) \notin Og(n)$

$$\lim_{n \to \infty} f(n)/g(n) = \lim_{n \to \infty} \frac{n^3/2}{37\, n^2 + 120\, n + 17}$$

$$= \lim_{n \to \infty} \frac{3/2\, n^2}{74\, n + 120} \qquad = \lim_{n \to \infty} \frac{3n}{74} = \infty$$

$f(n) = n^2 \quad g(n) = n \log n \; : \; g(n) \in O(f(n))$

Rappel: $\log_2 n = \log_e n \log_2 e$

Rappel: $(\ln x)' = 1/x$

$$\lim_{n \to \infty} g(n)/f(n) = \lim_{n \to \infty} \frac{n \log n}{n^2} = \lim_{n \to \infty} \frac{\log n}{n}$$

$$= \lim_{n \to \infty} \frac{(\ln n)(\log e)}{n} = \lim_{n \to \infty} \frac{1/n \log e}{1}$$

$$= \lim_{n \to \infty} \frac{\log e}{n} = 0$$

CSI 3505
Introduction

Recherche séquentielle (alg 1.1)	$W(n) = n$	$O(n)$
Recherche binaire (alg 2.1)	$W(n) = \lfloor \log n \rfloor + 1$	$O(\log n)$
Bubblesort (algo 1.3)	$W(n) = n(n-1)/2$	$O(n^2)$
Fusionner 2 listes triées «tailles h+m=n »	$W(n) = n-1$	$O(n)$
Fibonacci (alg 1.6) récursif	$W(n) > 2^n$	$\Omega(2^n)$
Fibonacci (alg 1.7) itératif	$W(n) = n-1$	$O(n)$
Multiplication matrices (alg1.4)	$W(n) = n^3$	$O(n^3)$
Commis voyageur	$W(n) = (n-1)! / 2$	$O(n !)$

CSI 3505
Introduction

Prouvez que

$\log_a n \in \Theta(\log_b n)$, par contre $2^{\log_a n} \notin \Theta(2^{\log_b n})$.

$\sum_{1 \le i \le n} i^k \in \Theta(n^{k+1})$ pour tout entier $k \ge 0$ fixé.

$\log(n!) \in \Theta(n \log n)$

$\sum_{1 \le i \le n} i^{-1} \in \Theta(\log n)$.

Si f est une fonction croissante alors

$\int_{a-1}^{b} f(x)\,dx \le \sum_{a \le i \le b} f(i) \le \int_{a}^{b+1} f(x)\,dx$.

$\log(n!) = \sum_{2 \le i \le n} \log(i) \ge \int_1^n \log(x)\,dx = \log e \int_1^n \ln(x)\,dx = \log e\ [x \ln x - x]_1^n$

Donc $\log(n!) \ge n \log n - n \log e + \log e \ge n \log n - n \log e \ge n \log n - 1.5\, n$.

{$\log e \ln n = \log n$ et $\log e = 1.44$}.

Moreover $\log(n!) = \sum_{2 \le i \le n} \log(i) \le n \log n$. Donc $\log(n!) \in \Theta(n \log n)$.