VARI 2
ALGORITHMES ET COMPLEXITÉ

PLAN

- Définition d'un algorithme
- Un exemple
- Présentation des algorithmes
- Évaluation d'un algorithme
- Complexité

1.1 DÉFINITION D'UN ALGORITHME

- **Procédure de calcul bien définie qui prend en entrée un ensemble de valeurs et produit en sortie un ensemble de valeurs.** (Cormen, Leiserson, Rivert)
- **Ensemble d'opérations de calcul élémentaires, organisé selon des règles précises dans le but de résoudre un problème donné. Pour chaque donnée du problème, il retourne une réponse après un nombre fini d'opérations.** (Beauquier,Berstel,Chrétienne)

- Phase 1 **ENONCÉ DU PROBLÈME**

  Spécification
- Phase 2 **ÉLABORATION DE L'ALGORITHME**
- Phase 3 **VÉRIFICATION**
- Phase 4 **MESURE DE L'EFFICACITÉ**

  Complexité
- Phase 5 **MISE EN OEUVRE**

  Programmation

1.2 UN EXEMPLE

Le tri par sélection ordinaire

ÉNONCÉ DU PROBLÈME

- **Entrée:**

  suite de nombres de $\mathbb{N}$ (et $\leq$)

- **Sortie:**

  suite triée des nombres

L'ALGORITHME

Principe

tant que **il reste plus d'un élément non trié**

répéter

- chercher le plus petit parmi les non triés ;

- échanger le premier élément non trié avec le plus petit trouvé;

fait;

EXEMPLE (suite)

éléments triés éléments non triés				
7	8	15	**5**	10
5	8	15	**7**	10
5	7	15	**8**	10
5	7	8	15	**10**
5	7	8	10	**15**
5	7	8	10	15

1.3 PRÉSENTATION DES ALGORITHMES

EXEMPLE (suite)

```
void    tri-selec (entier tableau [ ]  A )
début
  pour  I = 0 à n-2   répéter
    // il y a n-I+1 éléments non triés placés de I à n-1
    J = I; // I = place où on doit mettre le suivant plus petit,   //
    recherche de la place du plus petit des non triés
          pour  K = I+1 à n-1   répéter
                si  A[K] < A[J]  alors
                    J = K ;
                finsi ;
          fait;  J est la place du plus petit des éléments non triés
    échanger (A[I] , A[J])  ;
  fait;
fin
```

en JAVA

```
public class tri_selection{
public static void main (String args []){
int i = 0;
int j;
int k;
while( i<=n-2 ){
  j = i;
  k = i;
  while( k<=n-1 ){
     if( t[k]<t[j] ) j = k;
     k = k+1;
  }
int temp=t[j];t[j]=t[k];t[i]=temp;
i = i+1;
}

}
```

1.4 ÉVALUATION D'UN ALGORITHME

EFFICACITÉ

- **TEMPS D'EXÉCUTION**
- **MÉMOIRE OCCUPÉE**

Non mesurable par des tests

EXEMPLE

n entier est-il premier ?

n = 1148 ? non, divisible par 2

n = 1147 ? ??? (31 * 37)

NOMBRE D'OPÉRATIONS EN FONCTION DU NOMBRE DES DONNÉES

- **PIRE DES CAS**
- **EN MOYENNE**

EXEMPLES

- **VOYAGEUR DE COMMERCE (N VILLES)**

  Données: distances entre chaque paire de villes

  Nombre de données = N^2

  Enumération et évaluation des (N-1)! tournées possibles et sélection de la plus économique

- **MIN DE M NOMBRES**

  Même nombre de données: $M=N^2$

Temps de calcul si 10^{-9} secondes par comparaison

N	10	20	30	40	50
(M=) N^2	100	400	900	1600	2500
Min de N^2 nombres	O,1 µs	0,4 µs	0,9 µs	1,6 µs	2,5 µs
Min de (N-1)! nombres	O,32 s	1 an	10^{16} ans	10^{30} ans	10^{45} ans

OPERATIONS "ÉLÉMENTAIRES"

- **Machine de Turing**
- **Opérations de base**
- **Macro-opérations**

  + - * / tests (> ≤) et ou ...

TAILLE D'UN PROBLÈME
=
NOMBRE DE DONNÉES

Codage raisonnable des données

EXEMPLE (ajout, sous condition, d'éléments du tableau B à des éléments du tableau A ; on suppose que les 2 tableaux ont plus de m+n éléments)

```
void    AjoutTab (….)
début
    pour  I = 0 à m-1   répéter
                A(I) =B(I) ;                                   1
m               pour  J = I à n+I-1   répéter
          n               si  A(I) < B(J)  alors               1
fois      fois                  A(I) =A(I)+B(J) ;              2
                          finsi ;
                fait;
    fait;
fin
```

nombre d'opérations

Pire des cas, test bleu toujours vrai

m(1+ 3n)= 3nm+m opérations

Meilleur cas, test bleu toujours faux

m(1+ n)= nm+m opérations

EXEMPLE (suite)

```
void    tri-selec...
début
  pour  I = 0 à n-2    répéter                          n-1 fois
    J = I ;                                  1
          pour  K = I+1 à n-1   répéter
            si  A[k] < A[j]  alors  1        n-I fois    1+2(n-I)+3
                J = K ;        1
            finsi ;
          fait;
          échanger (A[I] , A[J])  ;          3           opérations
  fait;
fin
```

nombre d'opérations *(pire des cas)*

$$4(n-1)+2\sum_{I=1}^{n-1}(n-I)= \quad 4(n-1)+2\sum_{I=1}^{n-1}I=$$

4 (n-1)+ 2n(n-1)/2 = n^2 +3n - 4

Rappel : somme des n-1 premiers entiers = n(n-1)/2

Algorithme "efficace"

=

Algorithme polynomial

EXEMPLE (suite): $\mathbf{n^2 + 3n - 4}$

Problèmes "intraitables"

→

Théorie de la complexité

EXEMPLE: **voyageur de commerce**

1.5 COMPLEXITE DES ALGORITHMES

D_n = {données de taille n}

$coût_A$ (d) = coût de l'algorithme A sur la donnée d

- **PIRE DES CAS**

$$Max_A(n) = \max_{d \in Dn} | coût_A(d)$$

- borne sup du coût

- assez facile à calculer

- souvent réaliste

p(d) = probabilité de la donnée d

- **COÛT MOYEN**

$$\mathbf{Moy}_A\ \mathbf{(n)} = \sum_{d \in D_n} p(d) * coût_A(d)$$

- connaître la distribution des données

- souvent difficile à calculer

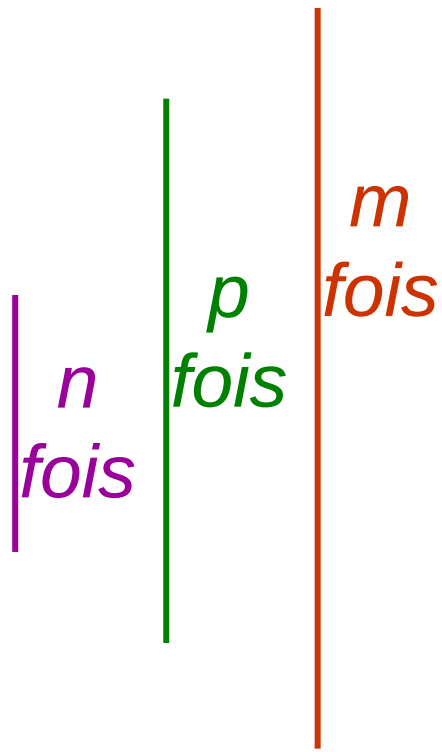
- intéressant si comportement usuel de l'algorithme éloigné du pire des cas

EXEMPLE (suite):

pire des cas ≈ cas moyen

EXEMPLE (voir première partie du cours)

```
float[][] produitMatrices( float[][] A,float[][]
  B ){  //A[m,n] et B[n,p] résultat C[m,p]
float[][] C = newfloat[A.length][B[0].length];
for(int i=0;i< A.length;i++){
  for(int j=0;j< B[0].length;j++){
     float s=0.0f;
     for(int k=0;k< A.length;k++){
          s=s+A[i][k]*B[k][j];
          C[i][j]=s;
     }
  }
}
return C;
}
```



pire cas = cas moyen=2mnp+1 opérations

NOTATIONS DE LANDAU

f , g fonctions $\mathbb{N} \rightarrow \mathbb{N}$

- **Notation O**

$f = O(g)$ ***lire : " f est en O de g "*** **$\Leftrightarrow$**

$\exists\, n_0 \in \mathbb{N}$ et c constante t.q. $f(n) \leq c\, g(n) \quad \forall\, n \geq n_0$

EXEMPLE

algo α : $f(n) = 4n^3+2n+1$ opérations

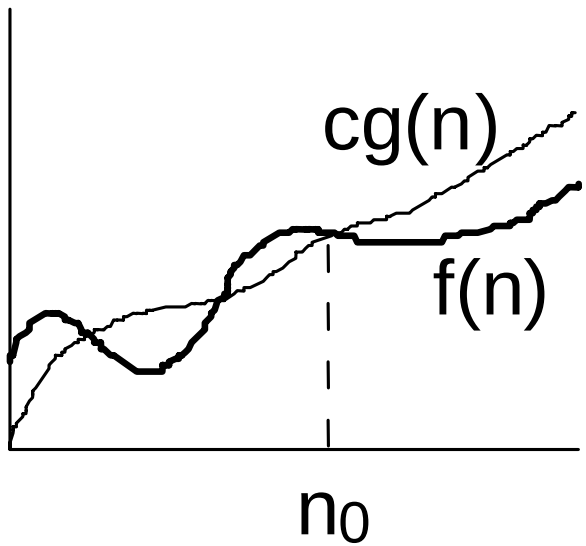
- O

pour tout $n \geq 1$: $f(n) \leq 7n^3$

$n_0=1$ $c=7$ $g(n)=n^3$

donc $f(n) = O(n^3)$

Notation O → majorant du nombre d'opérations d'un algorithme



f = O(g(n))

Complexité →

- **comportement des algorithmes quand le nombre de données augmente**
- **comparaison entre algorithmes**

Algorithme polynomial

$$f(n) = a_0 n^p + a_1 n^{p-1} + \ldots + a_{p-1} n + a_p$$

$$f = O(n^p)$$

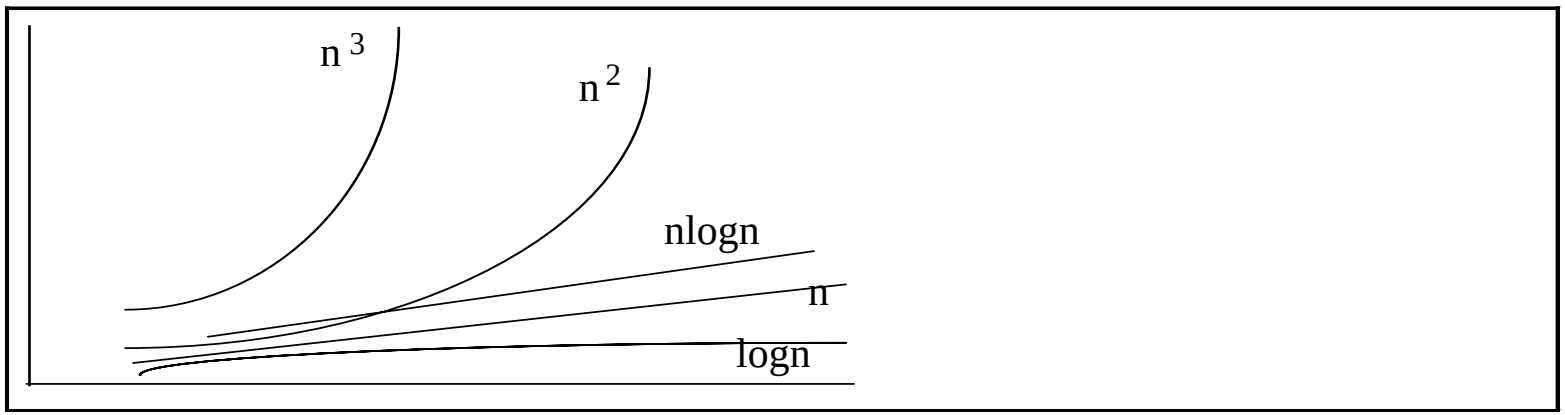
EXEMPLE (fin) $f(n) = n^2 + 3n - 4$

Le tri-selection est un algorithme polynomial en $O(n^2)$

“Bonne complexité”

O(log n) ou O(n) ou O(n log n)

Remarque: $\log^k n = O(n)$ $\forall$ k



Allure de quelques courbes

n = 10^6 1 μs par opérations

log2n	n	nlog2n	n^2	n^3
20μs	1 s	20 s	12 j	32 Ka

De l'effet des progrès de la technologie sur la résolution des problèmes

	taille max des problèmes traitables en 1 heure avec un ordinateur		
nombre d' opérations	actuel (le plus rapide)	100 fois plus rapide	1000 fois plus rapide
N	N_1	$100\ N_1$	$1000\ N_1$
n^2	N_2 (soit N_2^2 opérations)	$10\ N_2$	$31{,}6\ N_2$
n^5	N_3 (soit N_3^5 opérations)	$2{,}5\ N_3$	$3{,}98\ N_3$
2^n	N_4 (soit 2^{N4} opérations)	$N_4 + 6{,}64$	$N_4 + 9{,}97$
3^n	N_5 (soit 3^{N5} opérations)	$N_5 + 4{,}19$	$N_5 + 6{,}29$

Même si la puissance des ordinateurs augmente, certains "gros" problèmes ne pourront sans doute jamais être résolus

REMARQUES PRATIQUES

- **comptage grossier des opérations**

(+ - / * tests) mais attention aux boucles

- **ce qui est à l'extérieur des boucles est souvent négligeable**

- **on écrit O(logn) au lieu de $O(\log_2 n)$**

- **hiérarchie (n > 1)**

$\log n < n < n\log n < n^2 < n^3 << 2^n$

LE COMPROMIS ESPACE - TEMPS

Complexité en mémoire =

place mémoire nécessaire à l'exécution de l'algorithme en fonction de la taille du problème (sans le stockage du programme)

Temps diminue $\rightarrow$ espace mémoire augmente

(car stockage de données intermédiaires)