

# Chapitre 5

## Complexité des algorithmes

# Complexité des algorithmes

- En général la solution d'un problème n'est pas unique
- Les critères de choix d'une solution répondent aux exigences souvent contradictoires :
  - L'algorithme doit être simple à mettre en œuvre et à mettre au point.
  - L'algorithme doit mettre intelligemment à contribution les ressources de l'ordinateur et doit s'exécuter le plus rapidement possible.
- Le temps d'exécution d'un programme dépend :
  - Des données entrant dans le programme.
  - De la qualité du code généré par le compilateur.
  - De la nature et de la vitesse d'exécution des instructions du microprocesseur utilisé.
  - De la complexité de l'algorithme mis en œuvre.

- **Quel est le temps d'exécution du programme?**
  - **Complexité temporelle**
- **De combien de mémoire le programme a-t-il besoin?**
  - **Complexité de mémoire (ou Complexité spatiale)**
- **Comment déterminer ces complexités ?**
  - **méthode empirique**
  - **méthode mathématique**

## Méthode empirique

- Avec une montre et un logiciel d'analyse de mémoire
- **Problème** : dépend des facteurs suivants
  - de la machine utilisée;
  - du jeu d'instructions utilisées
  - de l'habilité du programmeur
  - du jeu de données générées
  - du compilateur choisi
  - ...

**BIEN SUR :** Le temps d'exécution dépend de la longueur de l'entrée .

- Ce temps est une fonction  $T(n)$  où  $n$  est la longueur des données d'entrée.

# Méthode mathématique

- **Théorie de la complexité**
  - **Temporelle**
  - **Spatiale**
- **Basée sur une machine abstraite**
  - **Random-Access Memory (RAM)**
  - **Instructions de base (affectation, boucle, appel de fonctions ...)**

## exemple 1

**Pb: Trouver le maximum d'un tableau.**

```
i := 1; max:=T[1];  
tant que i<=n  
    i := i + 1;  
    si max< T[i] alors  
        max := T[i] ;  
    fin si  
fin tant que
```

**Affectations :  $2 + 2n$**

**Comparaisons :  $2n$**

**Appel de fonctions : 0**

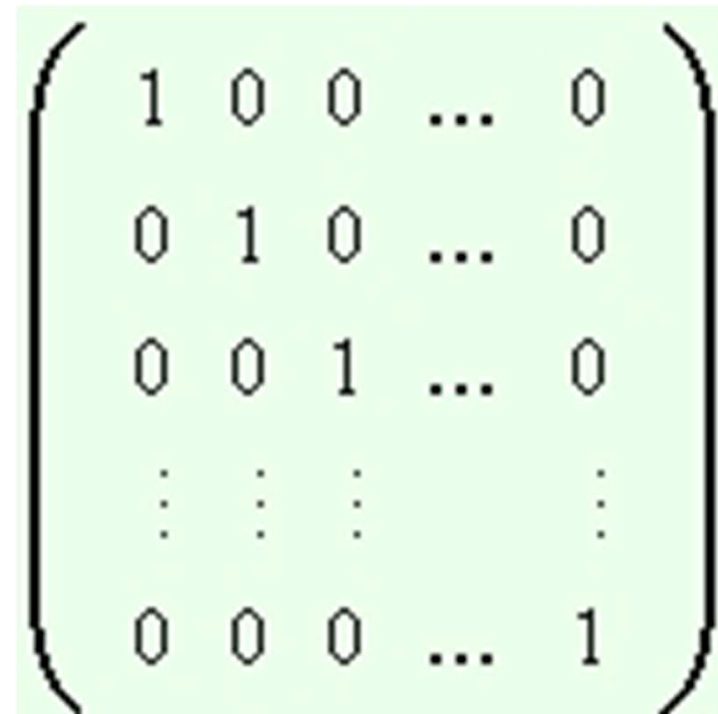
**(Attention : considérer le “worst case”, c-à-d le pire des cas)**

- **$T(n) = 4n+2$**

## Exemple 2

### Remplir une matrice identité

```
i := 1;  
tant que i<=n faire  
  j := 1;  
  tant que j<=n faire  
    si i=j ALORS  
      a[i][j] := 1;  
    sinon  
      a[i][j] := 0;  
    fin si  
    j := j + 1;  
  fin tant que  
  i := i + 1;  
fin tant que
```


$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$$

**Affectations :  $1 + 2n + 2n * n$**

**Comparaisons :  $n + 2n * n$**

$$T(n) = 4n^2 + 3n + 1$$



# Théorie de la complexité – Diminuer les constantes

Comment peut-on optimiser  $T(n)$ ?

$$T(n) = an^2 + bn + c$$

**1ère solution :** diminuer les constantes  $a, b, c$

Exemple :

–  $T_1(n) = 4n^2 + 3n + 1$

–  $T_2(n) = 4n^2$

–  $T_3(n) = n^2$

n	1	2	3	10	20	100	1000
$T_1(n)$	8	25	46	431	1661	40301	4003001
$T_2(n)$	4	16	36	400	1600	40000	4000000
$T_3(n)$	1	4	9	100	400	10000	1000000
$T_1/T_2$	2	1,56	1,28	1,08	1,04	1,01	1
$T_1/T_3$	8	6,25	5,11	4,31	4,15	4,04	4

- **Si on considère deux fonctions**

- $T1(n) = a_1n^2 + b_1n + c_1$

- $T2(n) = a_2n^2 + b_2n + c_2$

**alors :**

$$\lim_{n \rightarrow \infty} \frac{T1(n)}{T2(n)} = \frac{a_1}{a_2}$$

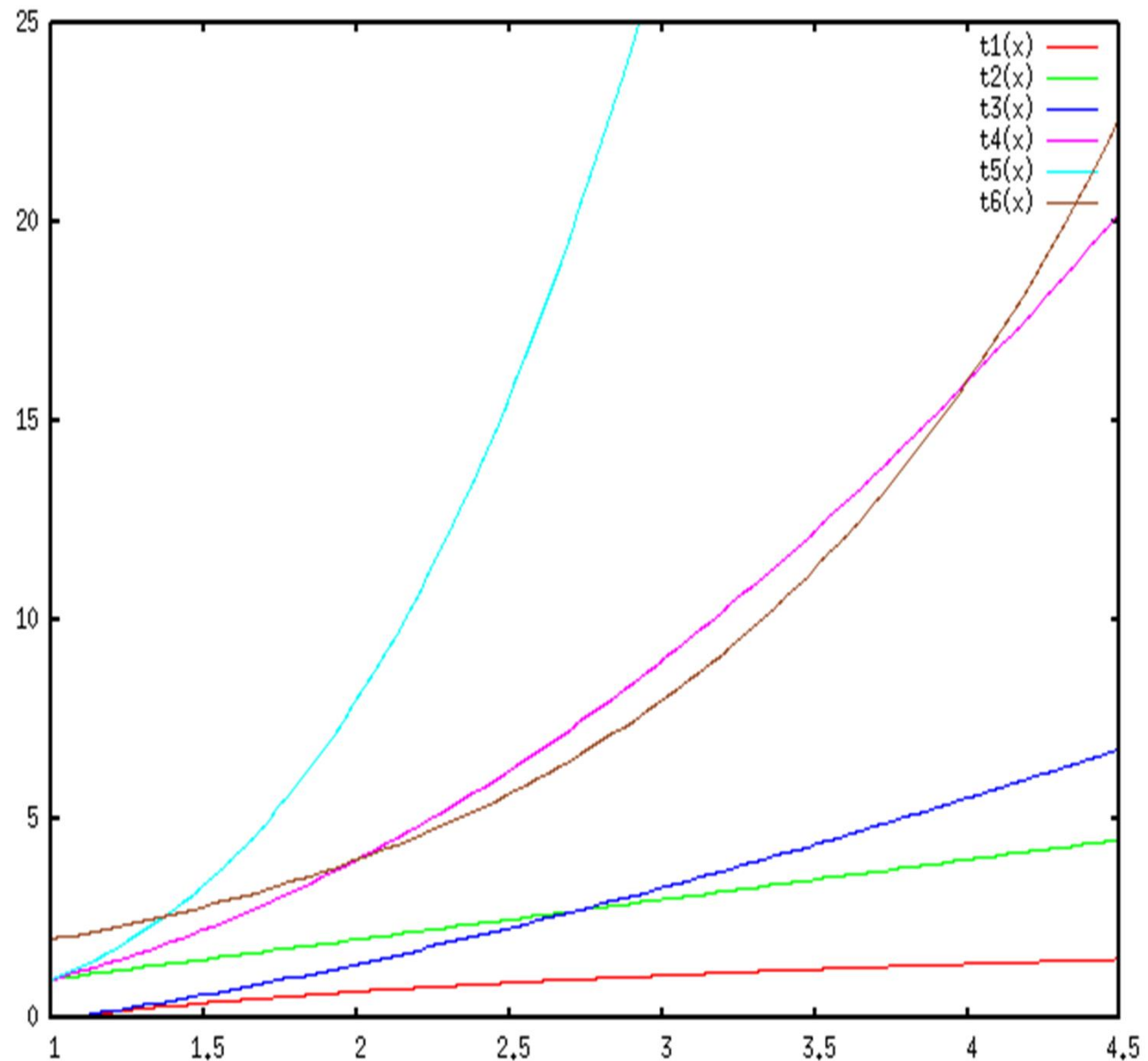
- **Pour des grands n, seule la constante du plus grand degré est significative**

# Théorie de la complexité – Changement de la fonction

## 2ème solution

changer la fonction

- $T1(n) = \log_2(n)$ 
  - logarithmique
- $T2(n) = n$ 
  - linéaire
- $T3(n) = n \log_2(n)$ 
  - Quasi-linéaire
- $T4(n) = n^2$ 
  - quadratique
- $T5(n) = n^3$ 
  - cubique
- $T6(n) = 2^n$ 
  - exponentiel



Quel taille de  $n$  peut être résolue en 1 seconde, une minute, une heure ?

– Avec une machine de 1000 instructions par seconde

$T_i(n)$	1 sec	1 min	1 heure
$\log_2 n$	$2^{1000}$	$2^{60000}$	$2^{3600000}$
$N$	1000	60000	36000000
$n \log_2 n$	140	4893	20000
$n^2$	31	244	1897
$n^3$	10	39	153
$2^n$	9	15	21