Complexité
Diviser pour regner
Master Théorème

Pascal Lafourcade

UNIVERSITÉ Clermont Auvergne

LIMOS

R5.A04 Qualité algorithmique
BUT 3, 2024-2025

Outline

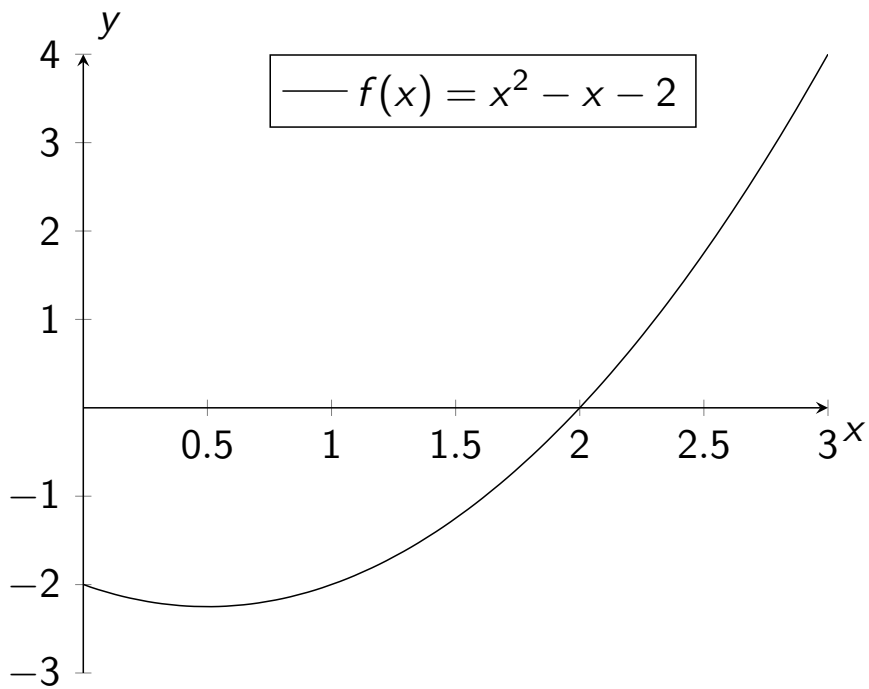
Outline

Trouver où s'annule une fonction

Problème

Soit f une fonction continue sur l'intervalle $[a, b]$, avec $b > a$, $f(a) < 0$ et $f(b) > 0$, trouver la valeur x telle que la fonction au point $|f(x)|$ soit plus petite que ϵ.

Soit $f(x) = x^2 - x - 2$, $a = 0$, $b = 3$,
$f(a) = f(0) = -2$ et $f(b) = f(3) = 4$



Balayage

- Parcourir avec un pas de 1 l'intervalle $[a, b]$, jusqu'à $f(a+m)$ et $f(a+n)$ soient de signes opposés.
- Parcourir avec un pas de 0.1 l'intervalle $[a+m, a+n]$, jusqu'à $f(a+m+o)$ et $f(a+n+p)$ soient de signes opposés.
- etc.

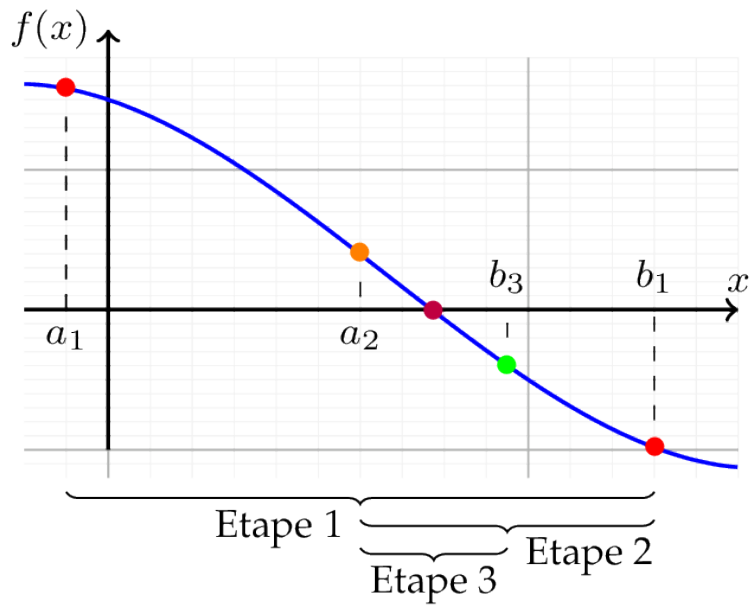
Cela peut être long au pire $|[a, b]|$.

Dichotomie

L'idée est alors d'évaluer ce que vaut f au milieu de $[a, b]$

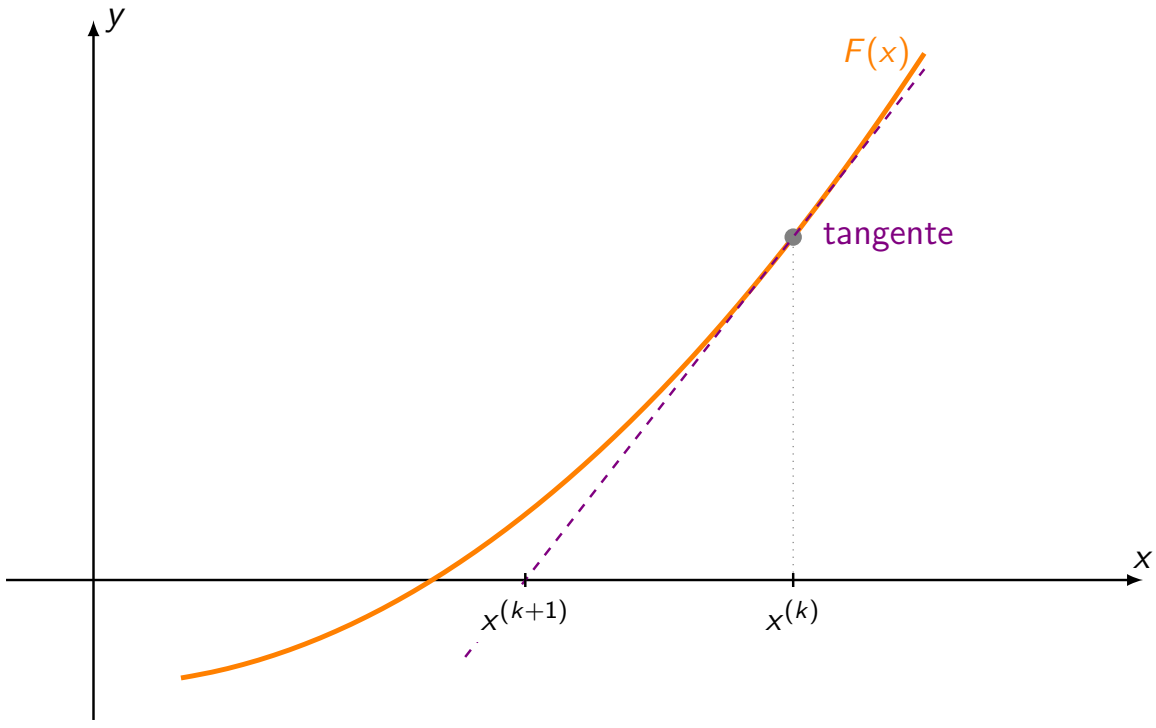
- si $f(\frac{a+b}{2}) \leq 0$, alors une racine est dans $[\frac{a+b}{2}, b]$
- sinon, une racine est dans $[a, \frac{a+b}{2}]$.

```
def dichotomie(a,b,prec):
    while b-a>prec:
          c = (a+b)/2
          if f(a)*f(c) <= 0:
             b = c
          else:
             a = c
    return a,b
```



Complexité en $log_2(|[a, b]|)$

Autres méthodes



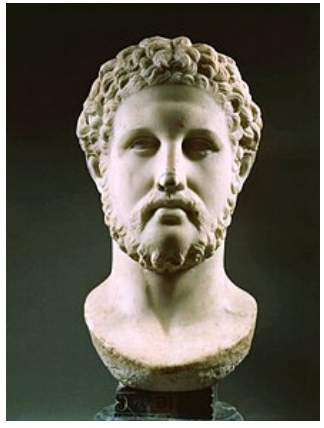
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

- Méthode d'Isaac Newton (convergence est quadratique)
- Méthode de Broyden (plus rapide)
- Méthode de Muller
- Méthode de Brent

Outline

Diviser pour régner

Divide ut regnes,
Philippe II de Macédoine père d'Alexandre Legrand.



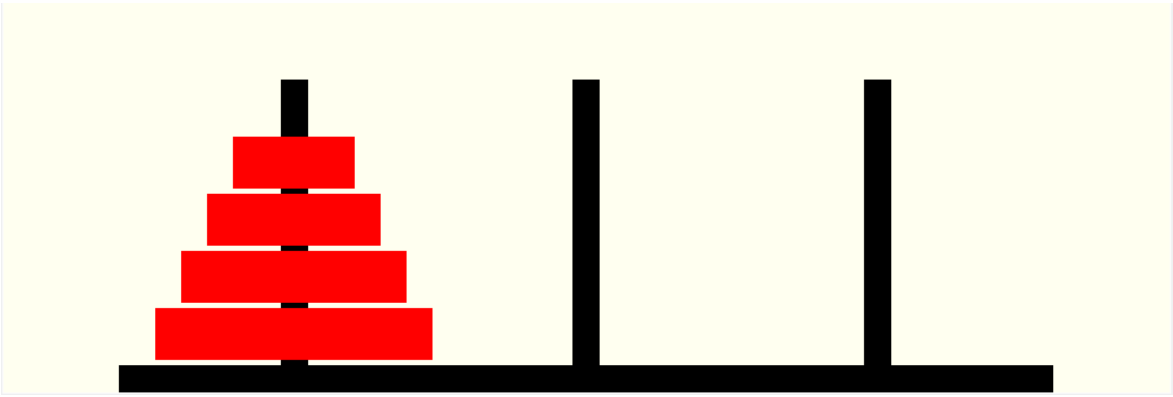
Divide et impera, Machiavel.

En 3 étapes :

- DIVISER : le problème est divisé en sous-problèmes
- RÉGNER : les sous-problèmes sont plus faciles à résoudre
- COMBINER : les solutions des sous-problèmes sont combinées

Exemple : PGCD, Exponentielle, Tri fusion, Karatsuba, Strassen, Tour de Hanoï.

Tour de Hanoï (Edouard Lucas 1883)

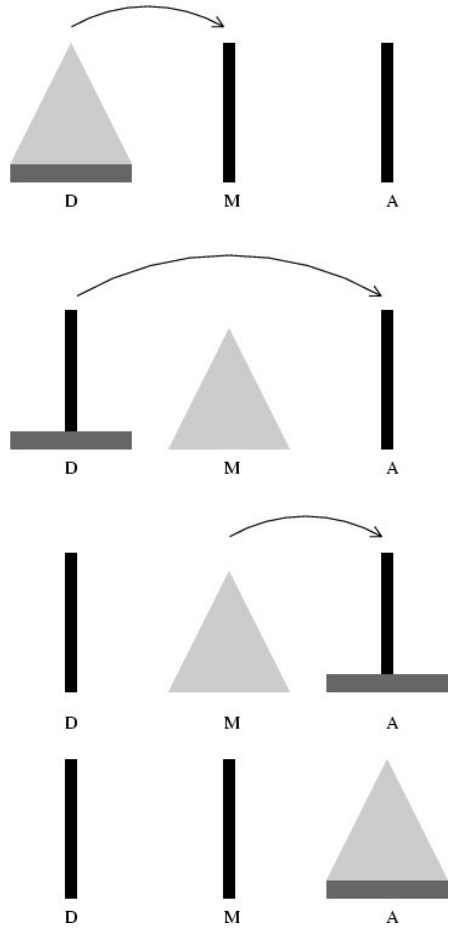


Règles

- Ne jamais poser un disque sur un disque plus petit.
- Ne déplacer qu'un disque à la fois.

BUT : Déplacer les 64 disques d'une colonne sur une autre.

Algorithme



Déplacer n disques de A vers C en passant par B :

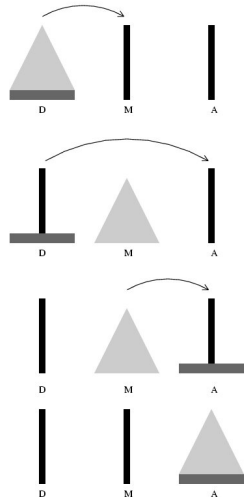
- Déplacer (n-1) disques de A vers B en passant par C;
- Déplacer 1 disque de A vers C ;
- Déplacer (n-1) disques de B vers C en passant par A.

Nombre de mouvements : Conjecture

Pour n disques, il faut $2^n - 1$ mouvements

Preuve par récurrence

- Cas de base : pour 0 (ou 1) disque il faut $0 = 2^0 - 1$ (ou $1 = 2^1 - 1$) mouvement

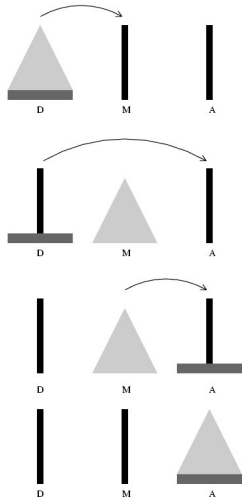


Nombre de mouvements : Conjecture

Pour n disques, il faut $2^n - 1$ mouvements

Preuve par récurrence

- Cas de base : pour 0 (ou 1) disque il faut $0 = 2^0 - 1$ (ou $1 = 2^1 - 1$) mouvement
- Induction : Supposons $P(n) = 2^n - 1$ vrai, montrons $P(n+1)$
  $P(n+1) = 2 * P(n) + 1$

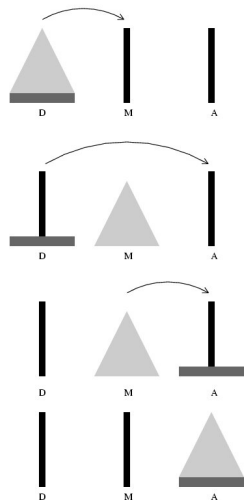


Nombre de mouvements : Conjecture

Pour n disques, il faut $2^n - 1$ mouvements

Preuve par récurrence

- Cas de base : pour 0 (ou 1) disque il faut $0 = 2^0 - 1$ (ou $1 = 2^1 - 1$) mouvement
- Induction : Supposons $P(n) = 2^n - 1$ vrai, montrons $P(n+1)$
  $P(n+1) = 2 * P(n) + 1 = 2 * (2^n - 1) + 1 = 2^{n+1} - 1$



PGCD : Plus Grand Commun Diviseur

$$PGCD(15, 6) = 3 \qquad PGCD(7, 11) = 1$$

$$PGCD(a, b)$$

- Décomposer en facteurs premiers a dans L_1 et b dans L_2.
- Calculer le $PGCD = L1 \cap L_2$.

PGCD : Algorithme des soustractions

- $pgcd(a, a) = a$ et $pgcd(a, 0) = a$
- $pgcd(a, b) = pgcd(a, b - a)$ pour $b \geq a > 0$
- $pgcd(a, b) = pgcd(b, a)$ pour $a > b > 0$ ou $a = 0$

Si les entiers sont égaux, alors leur PGCD est leur valeur commune.
Sinon :

- remplacer le plus grand des deux nombres par la valeur de la différence le plus grand et le plus petit;
- recommencer sur le nouveau couple de deux nombres, jusqu'à obtenir deux nombres égaux : la valeur commune est le PGCD des deux nombres initiaux.

PGCD : Euclide

- $pgcd(a, b) = pgcd(b, reste(a, b))$ pour $b \neq 0$
- $pgcd(a, 0) = a$

Calculer le reste de la division euclidienne du plus grand par le plus petit.
Si le reste est nul, alors le PGCD des deux nombres A et B est le plus petit des deux nombres.
Sinon :

- recommencer, en divisant le dernier diviseur par le dernier reste obtenu, jusqu'à obtenir un reste nul.
- Le PGCD des deux nombres A et B est le dernier reste non nul obtenu.

On divise au moins par 2 à chaque fois $2log_2(n) + 2$

Calcul naïf de l'exponentielle

```
def expo(n,p,x):
    res = 1
    for i in range(n) :
        res = res * x % p
   return res
```

Exponentielle rapide O(log n)

L'algorithme d'**exponentielle rapide** calcule plus vite $x^n \bmod p$

$$\text{expo}(x, n, p) =$$

$$\begin{cases} 1, & \text{si } n = 0 \\ \text{expo}(x^2 \bmod p,\ n/2, p), & \text{si } n \text{ est pair} \\ x \times \text{expo}(x^2 \bmod p,\ (n-1)/2, p) \bmod p, & \text{si n est impair} \end{cases}$$

Tris en $O(n^2)$

- Insertion
- Sélection
- Bulles

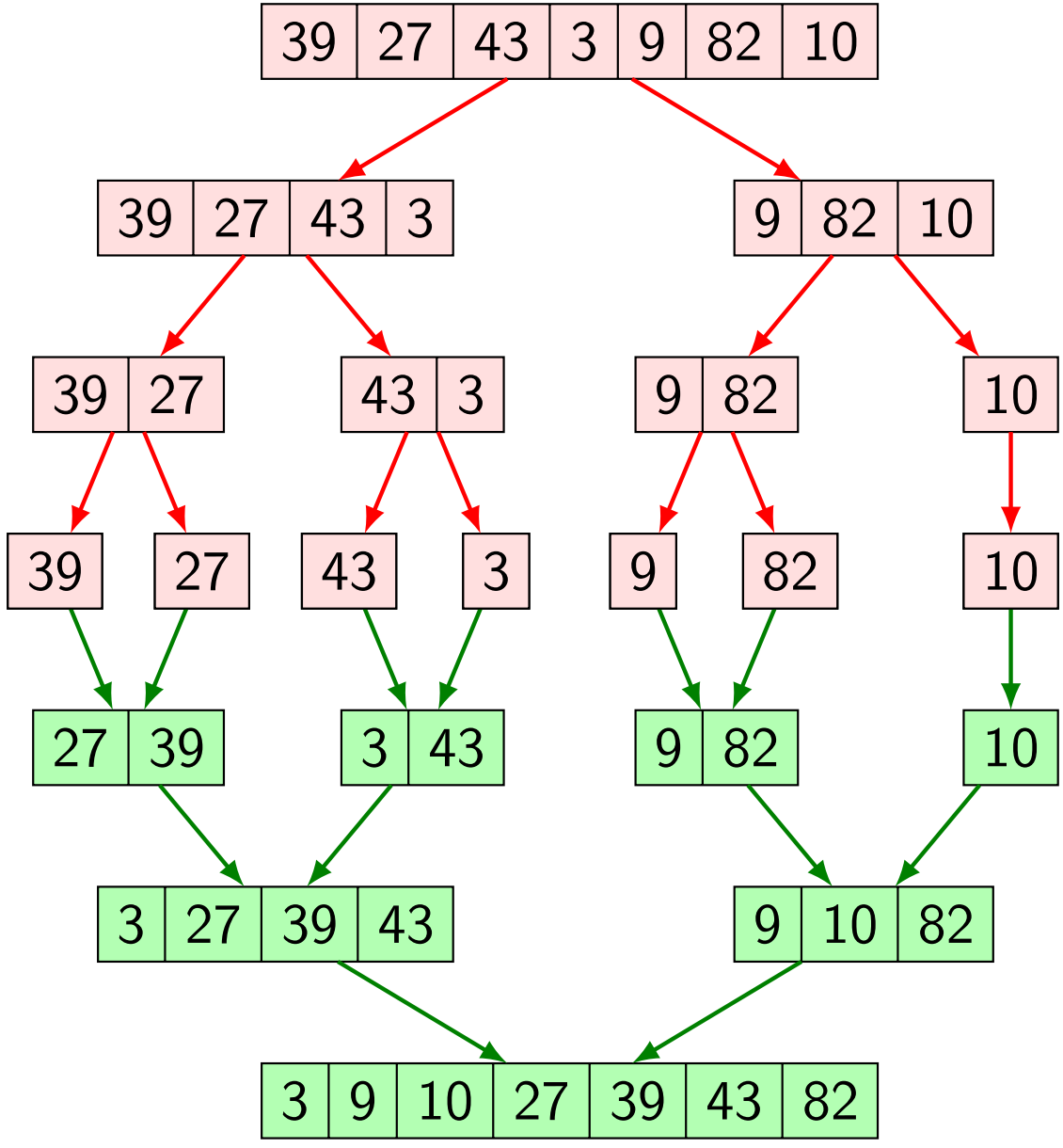
Tri Fusion $O(nlog(n))$

Principe :

- Découper la liste initiale en deux listes de taille égale
- Trier récursivement ces deux listes
- Fusionner les listes triées

Fusionner deux listes l_1 et l_2 déjà triées en une liste triée est rapide linéaire $O(|l_1| + |l_2| - 1)$.

Exemple : Tri Fusion ($O(n \log(n))$)



Tri Fusion ($O(n \log(n))$)

```
def mergesort(list):
    if len(list) < 2:
        return list
    else:
        middle = len(list) / 2
        left = mergesort(list[:middle])
        right = mergesort(list[middle:])
        return merge(left, right)
```

Tri fusion : merge

```
def merge(left, right):
    result = []
    i ,j = 0, 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            result.append(left[i])
            i += 1
        else:
            result.append(right[j])
            j += 1
    return result
```

Tri Fusion ($O(n \log(n))$)

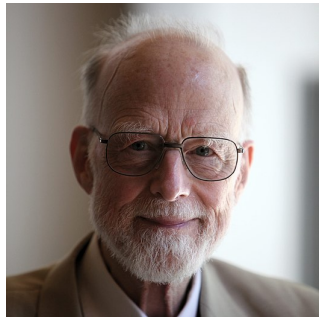
Complexité :

- $T(0) = 0$
- $T(1) = 1$
- $T(n) \approx 2T(n/2) + n - 1$

Tri Rapide ($O(n \log(n))$)

Idée :

Découper la liste initiale en deux morceaux en fonction d'un pivot.
Trier récursivement puis joindre les deux morceaux triés.



```
from random import randrange
def qsort(list):
    if list == []:
        return []
    else:
        pivot = list.pop(randrange(len(list)))
        lesser = qsort([l for l in list if l < pivot])
        greater = qsort([l for l in list if l >= pivot])
        return lesser + [pivot] + greater
    return qsort(list[:])
```

Algorithme de multiplication de 2 nombres entiers

$$(a \times 10^k + b)(c \times 10^k + d) = ac \times 10^{2k} + (ad + bc) \times 10^k + bd$$

nécessite 4 produits ac, ad, bc et bd.
Si on note $K(n)$ le nombre de multiplications nécessaires pour calculer le produit de 2 nombres à n chiffres, on obtient :
$K(n) < 4K(n/2)$
Complexité $O(n^{log_2(4)}) \approx O(n^2)$

Karatsuba, 1960

$$
\begin{aligned}
& ac \times 10^{2k} + (ac + bd - (a-b)(c-d)) \times 10^k + bd \\
= \quad & ac \times 10^{2k} + (ac + bd - ac + bc + ad - bd) \times 10^k + bd \\
= \quad & ac \times 10^{2k} + (ad + bc) \times 10^k + bd \\
= \quad & (a \times 10^k + b)(c \times 10^k + d)
\end{aligned}
$$

3 produits ac, bd et $(a–b)(c–d)$
Si on note $K(n)$ le nombre de multiplications nécessaires pour calculer le produit de 2 nombres à n chiffres, on obtient :
$K(n) \leq 3K(\lceil n/2 \rceil)$
Complexité $O(n^{log_2(3)}) \approx O(n^{1,585})$ au lieu de $O(n^2)$

Multiplication de matrices

Définition

Si $A = (a_{ij})$ est une matrice (m, n) et $B = (b_{ij})$ est une matrice (n, p), alors leur produit, noté $AB = (c_{ij})$ est une matrice (m, p) telle que :

$$\forall i, j : c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{in} b_{nj}$$

Applications : Jeux vidéo, imagerie, GPS, résolution systèmes, PageRank (probabilité), théorie des codes, Fibonnacci ...

Multiplication de matrices

$M = \left(\begin{smallmatrix} A & B \\ C & D \end{smallmatrix}\right)$ et $N = \left(\begin{smallmatrix} A' & B' \\ C' & D' \end{smallmatrix}\right)$ alors :

$$MN = \begin{pmatrix} AA' + BC' & AB' + BD' \\ CA' + DC' & CB' + DD' \end{pmatrix}$$

8 Multiplcations et 4 additions.

Exemple

$$N = \begin{pmatrix} 4 & 3 \\ 2 & 1 \end{pmatrix}$$

$$M = \begin{pmatrix} 1 & 2 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 8 & 5 \\ 2 & 1 \end{pmatrix}$$

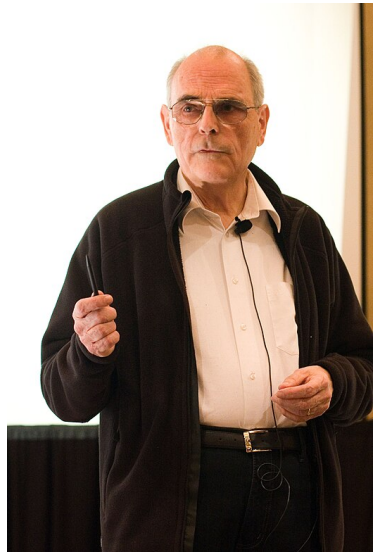
Algorithme de Strassen

$A, B \in \mathbb{R}^{d \times d}$ où $d := 2^k$ et $k \in \mathbb{N}^*$.

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{1,1} & \mathbf{A}_{1,2} \\ \mathbf{A}_{2,1} & \mathbf{A}_{2,2} \end{bmatrix}, \mathbf{B} = \begin{bmatrix} \mathbf{B}_{1,1} & \mathbf{B}_{1,2} \\ \mathbf{B}_{2,1} & \mathbf{B}_{2,2} \end{bmatrix}, \mathbf{C} = \begin{bmatrix} \mathbf{C}_{1,1} & \mathbf{C}_{1,2} \\ \mathbf{C}_{2,1} & \mathbf{C}_{2,2} \end{bmatrix}$$

$$\mathbf{M}_1 := (\mathbf{A}_{1,1} + \mathbf{A}_{2,2})(\mathbf{B}_{1,1} + \mathbf{B}_{2,2})$$
$$\mathbf{M}_2 := (\mathbf{A}_{2,1} + \mathbf{A}_{2,2})\mathbf{B}_{1,1}$$
$$\mathbf{M}_3 := \mathbf{A}_{1,1}(\mathbf{B}_{1,2} - \mathbf{B}_{2,2})$$
$$\mathbf{M}_4 := \mathbf{A}_{2,2}(\mathbf{B}_{2,1} - \mathbf{B}_{1,1})$$
$$\mathbf{M}_5 := (\mathbf{A}_{1,1} + \mathbf{A}_{1,2})\mathbf{B}_{2,2}$$
$$\mathbf{M}_6 := (\mathbf{A}_{2,1} - \mathbf{A}_{1,1})(\mathbf{B}_{1,1} + \mathbf{B}_{1,2})$$
$$\mathbf{M}_7 := (\mathbf{A}_{1,2} - \mathbf{A}_{2,2})(\mathbf{B}_{2,1} + \mathbf{B}_{2,2})$$
$$\mathbf{C}_{1,1} = \mathbf{M}_1 + \mathbf{M}_4 - \mathbf{M}_5 + \mathbf{M}_7$$
$$\mathbf{C}_{1,2} = \mathbf{M}_3 + \mathbf{M}_5$$
$$\mathbf{C}_{2,1} = \mathbf{M}_2 + \mathbf{M}_4$$
$$\mathbf{C}_{2,2} = \mathbf{M}_1 - \mathbf{M}_2 + \mathbf{M}_3 + \mathbf{M}_6$$

7 multiplications 18 additions !



Outline

Principe

Remplacer successivement les valeurs récursivement.

Exemple :

$$
\begin{aligned}
T(n) &= b + T(n-1) \\
&= b + (b + T(n-2)) \\
&= 2b + (b + T(n-3)) \\
&\vdots \\
&= k \times b + T(n-k) \\
&\vdots \\
&= n \times b + T(0)
\end{aligned}
$$

Complexité $O(n)$.

Exemple : $T(n) = 4T(n/2) + n$

Guess : $O(n^3)$ i.e., $T(k) \leq ck^3$

$$\begin{aligned} T(k) &\leq 4c(\frac{k}{2})^3 + k \\ &= \frac{c}{2}k^3 + k \\ &= ck^3 - (\frac{1}{2}ck^3 - k) \\ &\leq ck^3 \end{aligned}$$

Car, pour $c \geq 2$, $\frac{1}{2}ck^3 - k \geq 0$
Complexité est de $O(n^3)$

Exemple : $T(n) = 4T(n/2) + n$

Guess : $O(n^2)$ i.e., $T(k) \leq ck^2$

$$\begin{aligned} T(k) &\leq 4c(\frac{k}{2})^2 + k \\ &= ck^2 + k \\ &= ck^3 - (-k) \end{aligned}$$

$-k \geq 0$ seulemnt pour $k = 0$, donc cela ne fonctionne pas.

Exemple : $T(n) = 4T(n/2) + n$

Guess : $O(n^2)$ i.e., $T(k) \leq c_1 k^2 - c_2 k$

$$\begin{aligned} T(k) &= 4T(k/2) + k \\ &\leq 4c_1(\frac{k}{2})^2 - 4c_2 k/2 + k \\ &= ck^2 + (1 - c_2)k \\ &= ck^2 - c_2 k - (-1 + c_2)k \\ &\leq ck^2 - c_2 k \end{aligned}$$

$-1 + c_2 \geq 0$ seulemnt si $c_2 \geq 1$.
Rappel $T(1) \leq c_1 - c_2$ implique $c_1 \geq c_2$.

Outline

Recherche dichotomique dans un tableau

$f(a) < 0$ et $f(b) > 0$, trouver la valeur x telle que la fonction au point $|f(x)|$ soit plus petite que ϵ.

- Entrée : Tableau d'entiers triés
- Question : Trouver la position de x ou dire s'il n'est pas présent.

$$T(n) = T(n/2)$$

Donc $O(log_2(n))$

Tri fusion

$$T(n) = T(n/2) + n$$

Donc $O(nlog_2(n))$

Série Géométrique

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \ldots = 2$$

Rappel

Si $q < 1$:

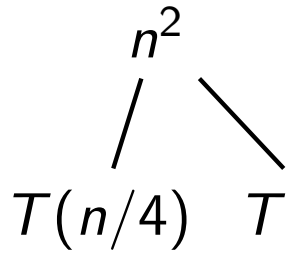
$$\sum_{i=0}^{i=k} q^i = \frac{1 - q^k}{1 - q}$$

$$\sum_{i=0}^{\infty} q^i = \frac{1}{1 - q}$$

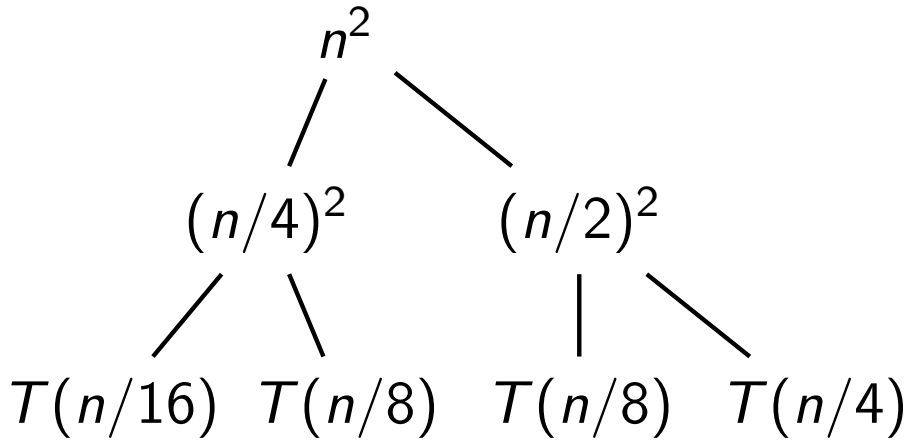
$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} \ldots = \frac{1}{1 - \frac{1}{2}} = \frac{1}{\frac{1}{2}} = 2$$

Exemple

$$T(n) = T(n/4) + T(n/2) + n^2$$



$$T(n) = T(n/16) + T(n/8) + (n/4)^2 + T(n/8) + T(n/4) + (n/2)^2 + n^2$$



Nombre de feuilles $< n$,

$n^2 + \frac{5}{16}n^2 + \frac{25}{256}n^2 \ldots = (1 + \frac{5}{16} + \frac{5^2}{16^2} \ldots + \frac{5^k}{16^k} \ldots)n^2 \leq 2n^2$

Outline

Master Theorem

$$T(n) = a\ T\left(\frac{n}{b}\right) + f(n), \quad a \geq 1, b > 1$$

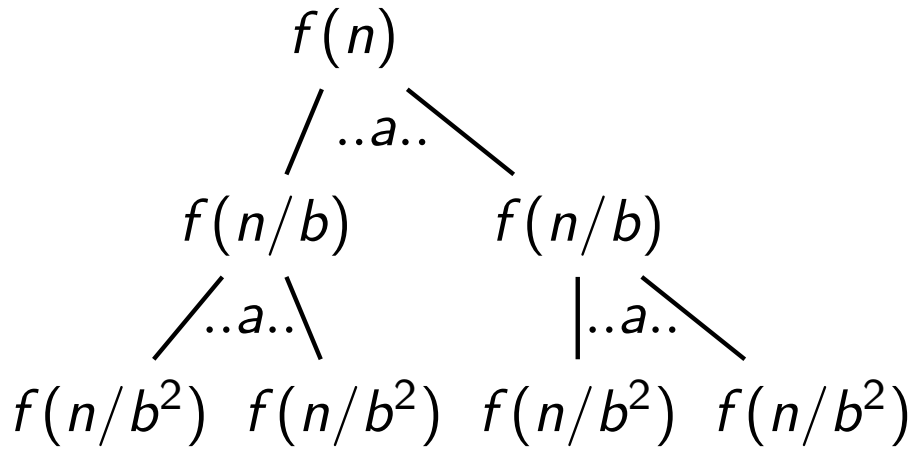
1. Si $f(n) = O\left(n^c\right)$ avec $c < \log_b a$, alors $T(n) = \Theta\left(n^{\log_b a}\right)$.
2. Si $f(n) = \Theta\left(n^c \log^k n\right)$ avec $c = \log_b a$ et une constante $k \geq 0$, alors $T(n) = \Theta\left(n^c \log^{k+1} n\right)$
3. Si $f(n) = \Omega\left(n^c\right)$ avec $c > \log_b a$ et s'il existe une constante $k < 1$ telle que, pour n assez grand, on a : $af\left(\frac{n}{b}\right) \leq kf(n)$ alors on a : $T(n) = \Theta\left(f(n)\right)$

Master Théorème, si $f(n)$ est un polynôme

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

1. Si $d < \log_b a$ alors $T(n) = O(n^{\log_b a})$
2. Si $d = \log_b a$ alors $T(n) = O(n^d \log_b n)$
3. Si $d > \log_b a$ alors $T(n) = O(n^d)$

Intuition de la preuve



Hauteur de l'arbre : $h = log_b(n)$
$a^h = a^{log_b(n)} = n^{log_b(a)}$

- Racine : $f(n)$
- Niveau 1 : $af(n/b)$
- Niveau 2 : $af(n/b^2)$

Cas 1 : Série géométrique croissante
Cas 2 : $f(n) * h$
Cas 3 : Série géométrique décroissante

Exemple 1 du Cas 1.

Rappel.

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d < \log_b a$ alors $T(n) = O(n^{\log_b a})$

$$T(n) = 4T\left(\frac{n}{2}\right) + n$$

$a = 4$, $b = 2$, $f(n) = n, d = 1$, et l'hypothèse sur c est bien vérifiée puisque $d = 1 < 2 = \log_b a = \log_2 4$.
On a donc
$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^2\right)$

Exemple 2 du Cas 1.

Rappel.

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d < \log_b a$ alors $T(n) = O(n^{\log_b a})$

$$T(n) = 8T\left(\frac{n}{2}\right) + 1000n^2$$

$a = 8$, $b = 2$, $c = 2$, et l'hypothèse sur c est bien vérifiée puisque $\log_b a = \log_2 8 = 3 > c$.
On a donc
$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^3\right)$
Si $T(1) = 1$, alors
$T(n) = 1001n^3 - 1000n^2$

Exemple 1 du Cas 2.

Rappel

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d = \log_b a$ alors $T(n) = O(n^d \log_b n)$

$$T(n) = 4\left(\frac{n}{2}\right) + n^2$$

$a = 4$, $b = 2$, $f(n) = n^2, d = 2$, et l'hypothèse sur c est bien vérifiée puisque $d = 2 = \log_b a = \log_2 4$.
On a donc
$T(n) = \Theta\left(n^{\log_b a}\right) = \Theta\left(n^{2log(n)}\right)$

Exemple 2 du Cas 2.

Rappel

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d = \log_b a$ alors $T(n) = O(n^d \log_b n)$

$T(n) = 2T\left(\frac{n}{2}\right) + 10n$
$a = 2,\ b = 2,\ c = 1,\ k = 0,\ f(n) = 10n = \Theta\left(n^c \log^k n\right)$. Comme $\log_b a = \log_2 2 = 1$, on a bien $c = \log_b a$ et par l'énoncé, on a $T(n) = \Theta\left(n^{\log_b a} \log^{k+1} n\right) = \Theta\left(n^1 \log^1 n\right) = \Theta(n \log n)$
Avec $T(1) = 1$ on obtient $T(n) = n + 10n \log_2 n$

Exemple 1 du Cas 3.

Rappel

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d > \log_b a$ alors $T(n) = O(n^d)$

$$T(n) = 4\left(\frac{n}{2}\right) + n^3$$

$a = 4$, $b = 2$, $f(n) = n^2, d = 3$, et l'hypothèse sur c est bien vérifiée puisque $d = 3 > 2 = \log_b a = \log_2 4$.
On a donc
$T(n) = \Theta(n^d) = \Theta(n^3)$

Exemple 2 du Cas 3.

Rappel

$$T(n) = a\ T\left(\frac{n}{b}\right) + O(n^d)$$

Si $d > \log_b a$ alors $T(n) = O(n^d)$

$T(n) = 2T\left(\frac{n}{2}\right) + n^2$
$a = 2$, $b = 2$, $c = 2$, $f(n) = n^2 = \Omega\left(n^c\right)$. On a bien
$2 = c > \log_b a = \log_2 2 = 1$ et, en choisissant $k = 1/2$, on a
$2\frac{n^2}{4} \leq kn^2$ ainsi $T(n) = \Theta(f(n)) = \Theta(n^2)$
$T(n) = 2n^2 - n$ avec $T(1) = 1$.

Exemple.

$$T(n) = 4T(\frac{n}{2}) + \frac{n^2}{log(n)}$$

Exemple.

$$T(n) = 4T(\frac{n}{2}) + \frac{n^2}{log(n)}$$

Le Master Theorem ne s'applique pas, il faut faire à la main (substitution ou autre).

Conclusion

Today

- Diviser pour régner
- Substitution
- Arbre de récursion
- Master Theorem

Questions?