# Complexité des algorithmes

Stéphane Grandcolas

stephane.grandcolas@univ-amu.fr

# Algorithmes

$\mathcal{P}$ : un problème

$\mathcal{M}$ : une méthode pour résoudre le problème $\mathcal{P}$

Algorithme : description de la méthode $\mathcal{M}$ dans un *langage algorithmique*

du nom du mathématicien perse *Al Khuwarizmi* (780 - 850)

# Structures algorithmiques

**Structures de contrôle**

- séquence
- embranchement (ou sélection)
- boucle (ou itération)

**Structures de données**

- constantes
- variables
- tableaux
- structures récursives (listes, arbres, graphes)

# Complexité des algorithmes

On veut

- Evaluer l'efficacité de la méthode $\mathcal{M}$
- Comparer $\mathcal{M}$ avec une autre méthode $\mathcal{M}'$

indépendamment de l'environnement (machine, système, compilateur, ...)

# Complexité des algorithmes

Evaluation du nombre d'opérations élémentaires en fonction

- de la taille des données,
- de la nature des données.

**Notations :**

- $n$ : taille des données,
- $T(n)$ : nombre d'opérations élémentaires

Configurations caractéristiques

- meilleur cas,
- **pire des cas**,
- cas moyen.

# Evaluation de $T(n)$ (séquence)

**Somme des coûts.**

$$\left.\begin{array}{ll} \text{Traitement1} & T_1(n) \\ & \\ \text{Traitement2} & T_2(n) \end{array}\right\} \qquad T(n) = T_1(n) + T_2(n)$$

# Evaluation de $T(n)$ (embranchement)

**Max des coûts.**

$$\left.\begin{array}{lr} \textbf{si} \ < \text{condition} > \ \textbf{alors} & \\ \quad \text{Traitement1} & T_1(n) \\ \textbf{sinon} & \\ \quad \text{Traitement2} & T_2(n) \end{array}\right\} max(T_1(n), T_2(n))$$

# Evaluation de $T(n)$ (boucle)

**Somme des coûts des passages successifs**

$$\left.\begin{array}{l} \textbf{tant que} \;\; < \text{condition} > \;\; \textbf{faire} \\ \qquad \text{Traitement} \qquad\qquad T_i(n) \\ \textbf{fin faire} \end{array}\right\} \qquad \sum_{i=1}^{k} T_i(n)$$

$T_i(n)$ : coût de la $i^{\text{ème}}$ itération

souvent défini par une équation récursive

# Evaluation de $T(n)$ (fonctions récursives)

**fonction** FunctionRecursive $(n)$

---

**si** $(n > 1)$ **alors**

    FunctionRecursive($n/2$),    coût $T(n/2)$

    Traitement$(n)$,    coût $C(n)$

    FunctionRecursive($n/2$),    coût $T(n/2)$

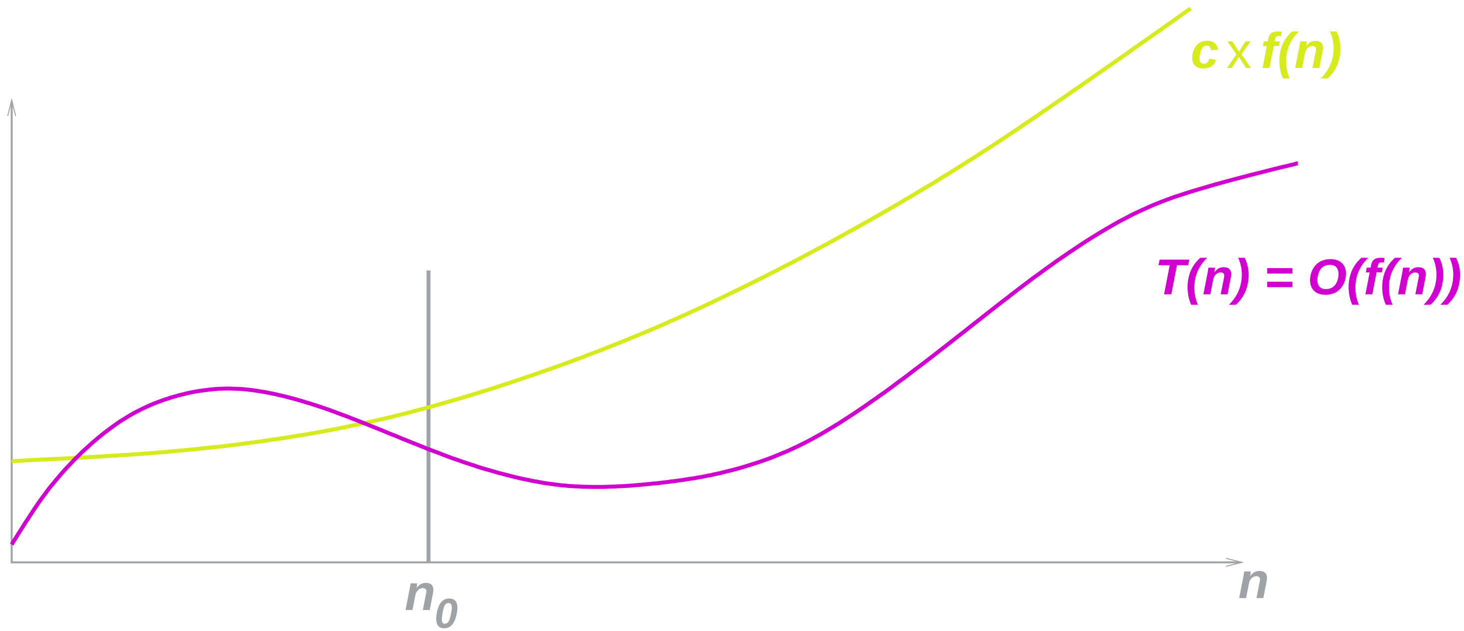
Equation récursive

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

si $C(n) = 1$ alors $T(n) = K \times n$

si $C(n) = n$ alors $T(n) = K \times n \times \log n$

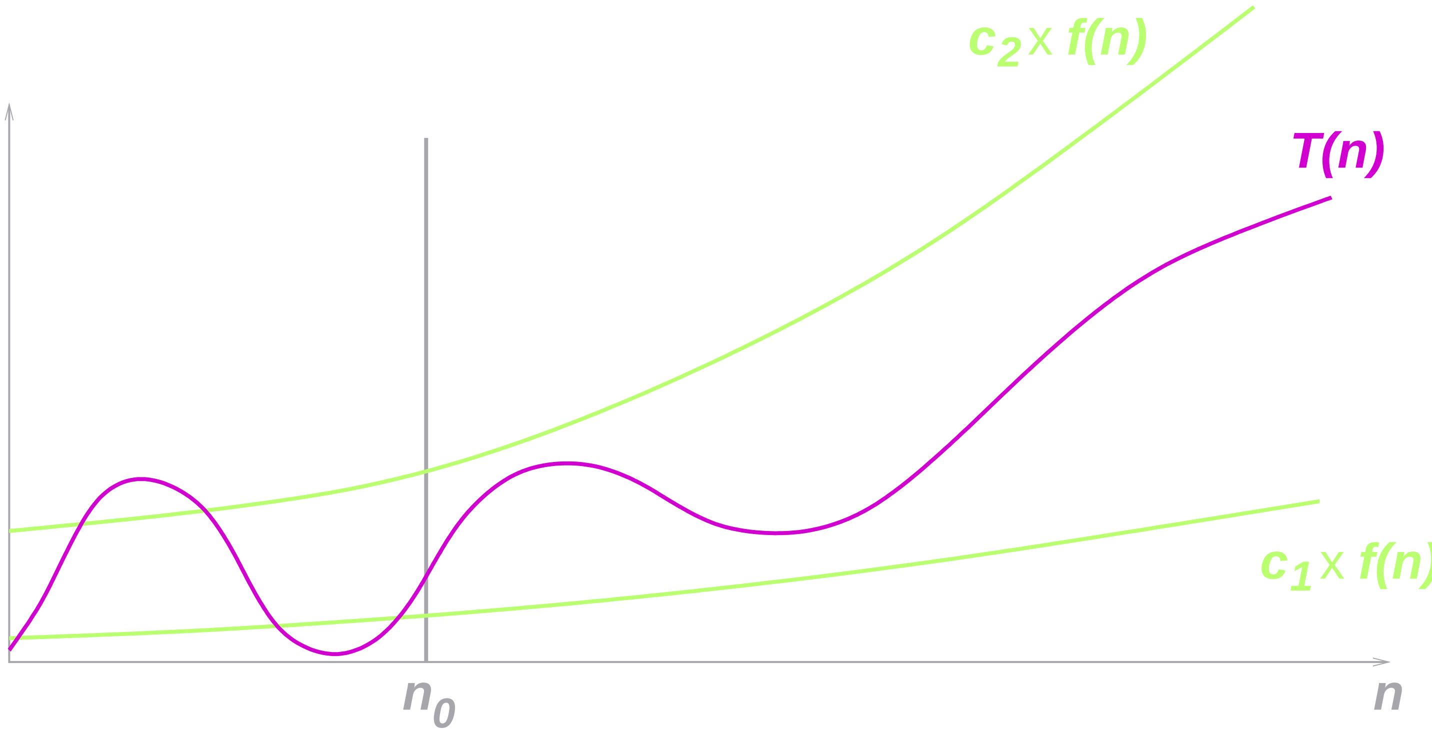
# Notation de Landau $O(f(n))$




Caractérise le comportement asymptotique (i.e. quand $n \to \infty$).

$$T(n) = O(f(n)) \ \text{ si } \exists c \ \exists n_0 \text{ tels que } \forall n > n_0, \ T(n) \leq c \times f(n)$$

# Notation $\Theta(f(n))$




$T(n) = \Theta(f(n))$ si $\exists c_1, c_2, n_0$ tels que

$$\forall n > n_0, \quad c_1 \times f(n) \leq T(n) \leq c_2 \times f(n)$$

# Exemples

$$f(n) = n^3 + 2\ n^2 + 4\ n + 2 = O(n^3)$$

(si $n \geq 1$ alors $f(n) \leq 8 \times n^3$)

# Exemples

$$f(n) = n^3 + 2\,n^2 + 4\,n + 2 = O(n^3)$$

(si $n \geq 1$ alors $f(n) \leq 8 \times n^3$)

$$f(n) = n\log n + 12\,n + 888 = O(n\log n)$$

# Exemples

$$f(n) = n^3 + 2\,n^2 + 4\,n + 2 = O(n^3)$$

(si $n \geq 1$ alors $f(n) \leq 8 \times n^3$)

$$f(n) = n \log n + 12\,n + 888 = O(n \log n)$$

$$f(n) = 1000\,n^{10} - n^7 + 12\,n^4 + \frac{2^n}{1000} = O(2^n)$$

# Les principales classes de complexité

| | |
|---|---|
| $O(1)$ | temps constant |
| $O(\log n)$ | logarithmique |
| $O(n)$ | linéaire |
| $O(n \times \log n)$ | tris (par échanges) |
| $O(n^2)$ | quadratique, polynomial |
| $O(2^n)$ | exponentiel (problèmes très difficiles) |

# Exemple : permutation

**fonction** permutation $(S, i, j)$

| | |
|---|---|
| $tmp := S[i]$, | coût $c_1$ |
| $S[i] := S[j]$, | coût $c_2$ |
| $S[j] := tmp$, | coût $c_3$ |
| **renvoyer** $S$ | coût $c_4$ |

Coût total

$$T(n) = c_1 + c_2 + c_3 + c_4 = O(1)$$

# Exemple : recherche séquentielle

**fonction** recherche $(x, S, n)$

---

$i := 1$,

**tant que** $((i < n)$ **et** $(S[i] \neq x))$ **faire** $\quad$ ($n$ fois)

$\quad\quad i := i + 1$,

**renvoyer** $(S[i] = x)$

Pire des cas : $n$ fois la boucle

$$T(n) = 1 + \sum_{i=1}^{n} 1 + 1 = O(n)$$

# Exemple : tri à bulle

**fonction** Tri $(S, n)$

**pour** $i := n$ **à** $2$ **faire** ($n - 1$ fois)

  **pour** $j := 1$ **à** $i - 1$ **faire** ($i - 1$ fois)

    **si** $(S[j] > S[j+1])$ **alors**

      permuter $S[j]$ et $S[j+1]$ dans $S$,

$$T(n) = C_{perm} \times \sum_{i=1}^{n-1} i = \frac{C_{perm} \times n \times (n-1)}{2} = O(n^2)$$

# Equations récursives

Boucles itératives, fonctions récursives (approches de type diviser pour régner notamment)

## Cas général

$$T(n) = a \times T(n/b) + f(n)$$

- méthode par substitution,
- méthode par développement itératif,
- méthode générale.

# Méthode par substitution [Equations récursives]

**Principe** : on vérifie une **intuition**

$$T(n) = a \times T(n/b) + f(n) \text{ et } T(1) = c$$

**Hypothèse**

$$T(n) = g(n) \quad \text{(intuition)}$$

**Conclusion**

$$a \times g(n/b) + f(n) = g(n) \text{ et } g(1) = c$$

à démontrer en fixant les constantes

# Méthode par substitution [Equations récursives]

**fonction** RechercheDichotomique $(x, \langle S_1, \ldots, S_n \rangle)$

| | |
|---|---|
| $g := 0$, $d := n+1$, | $g < position \leq d$ |
| **tant que** $(g < d-1)$ **faire** | $stoppe\ quand\ g = d-1$ |
|     **si** $(x > S_{((g+d)/2)})$ **alors** | $g < (g+d)/2 < d$ |
|         $g := (g+d)/2$, | $et\ donc\ g < position \leq d$ |
|     **sinon** | |
|         $d := (g+d)/2$, | $et\ donc\ g < position \leq d$ |
| **renvoyer** $d$, | $g < position \leq d\ et\ g = d-1$ |

Nombre d'itérations $T(n) = 1 + T(n/2)$

# Méthode par substitution [Equations récursives]

$$\boxed{T(n) = 1 + T(n/2) \text{ et } T(1) = 1^{\star}}$$

intuition $\quad T(n) = O(\log_2 n)$

Hypothèse $\quad T(n) = a \times \log_2 n + c$

donc $T(n/2) = a \times \log_2 n - a + c$

en substituant on a $T(n) = 1 + a \times \log_2 n - a + c$

donc $1 - a + c = c$ et $a = 1$, et puisque $T(1) = 1$ donc $c = 1$

Conclusion $\quad T(n) = \log_2 n + 1$

$^{\star}$ s'il y a un élément on fera une itération

# Méthode itérative [rappel sommations]

$$\sum_{i=1}^{n-1} i = \frac{n \times (n-1)}{2} = O(n^2)$$

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$$

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

# Méthode itérative [Equations récursives]

**fonction** TriParFusion $(S, n)$

**si** $(n \leq 1)$ **alors**

  **renvoyer** $S$

décomposer $S$ en $S_1$ et $S_2$, $(n)$

$S_1$ :=TriParFusion$(S_1)$, $(T(\lceil n/2 \rceil))$

$S_2$ :=TriParFusion$(S_2)$, $(T(\lfloor n/2 \rfloor))$

$S$ :=fusion$(S_1, S_2)$, $(n)$

**renvoyer** $S$

$$T(n) = 1 + n + 2 \times T(n/2) + n \quad \text{et} \quad T(1) = 1$$

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\ n + 1 + 2\ T(n/2)$$

donc $T(n/2) = n + 1 + 2\ T(n/4)$

$$T(n) = (2\ n + 1) + (2\ n + 2) + 4\ T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\ T(n/8)$

$$T(n) = (2\ n + 1) + (2\ n + 2) + (2\ n + 4) + 8\ T(n/8)$$

...

$$T(n) = \sum_{i=0}^{\log\ n-1} (2\ n + 2^i) + 2^{\log\ n}$$

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

$$T(n) = \sum_{i=0}^{\log\,n-1}(2\,n + 2^i) + 2^{\log\,n}$$

$$T(n) = 2n\,\log\,n + \sum_{i=0}^{\log\,n-1} 2^i + n$$

# Méthode itérative [Equations récursives]

$$T(1) = 1$$
$$T(n) = 2\,n + 1 + 2\,T(n/2)$$

donc $T(n/2) = n + 1 + 2\,T(n/4)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + 4\,T(n/4)$$

or $T(n/4) = n/2 + 1 + 2\,T(n/8)$

$$T(n) = (2\,n + 1) + (2\,n + 2) + (2\,n + 4) + 8\,T(n/8)$$

...

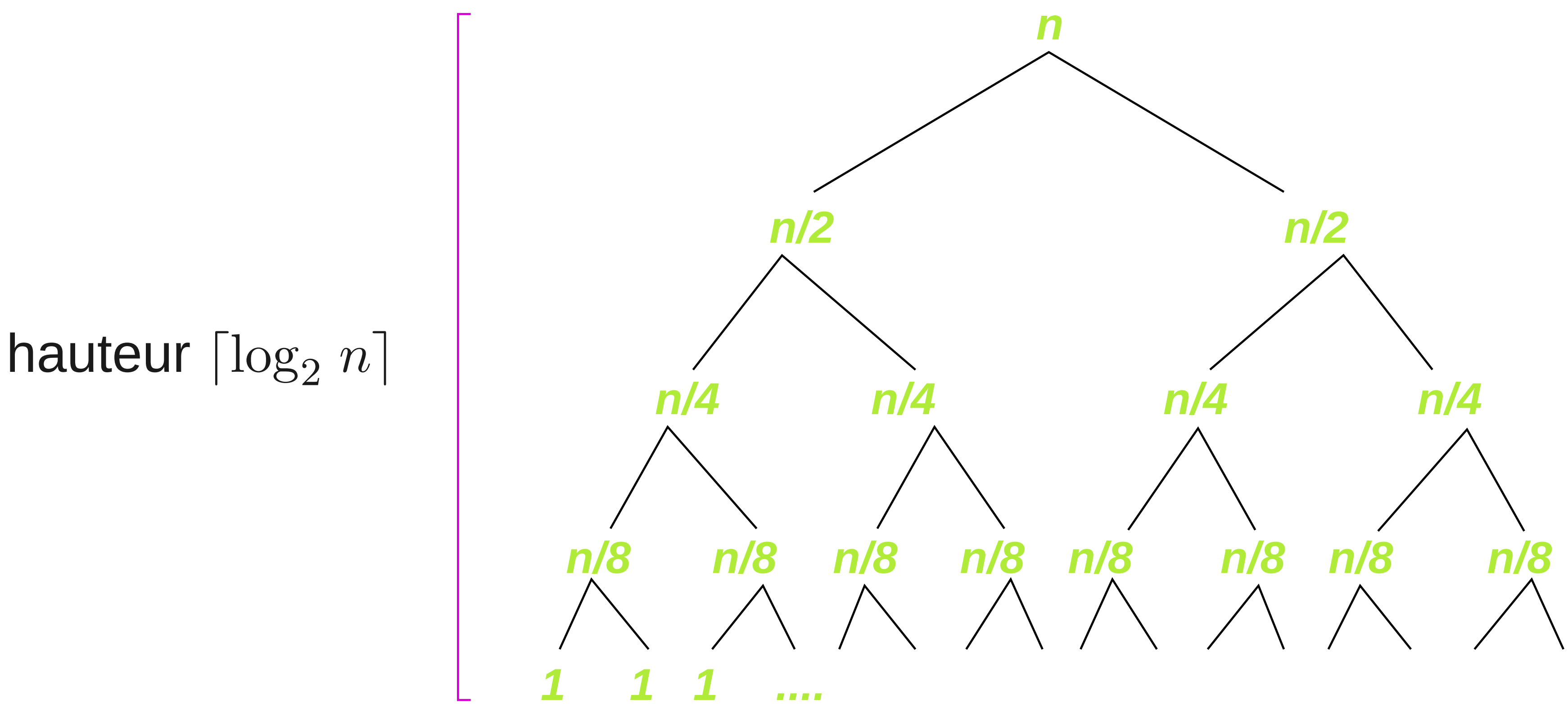
$$T(n) = \sum_{i=0}^{\log\,n-1} (2\,n + 2^i) + 2^{\log\,n}$$

$$T(n) = 2n\,\log\,n + \sum_{i=0}^{\log\,n-1} 2^i + n$$

et comme $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$, on a $\sum_{i=0}^{\log\,n-1} 2^i = 2^{\log\,n} - 1$

$$T(n) = 2n\,\log\,n + 2n - 1 = O(n\,\log\,n)$$

# Méthode itérative [Equations récursives]




$O(k)$ traitements pour décomposer et fusionner une suite de longueur $k$

$$T(n) = 2 \times n \times \lceil \log_2 n \rceil$$

# Méthode générale [Equations récursives]

$$T(n) = a \times T(n/b) + f(n)$$

avec $a \geq 1$, $b > 1$ et $f(n)$ est positive asymptotiquement.

- si $\exists \epsilon > 0,\ f(n) = O(n^{\log_b\ a-\epsilon})$ alors $T(n) = \Theta(n^{\log_b\ a})$,
- si $f(n) = \Theta(n^{\log_b\ a})$ alors $T(n) = \Theta(n^{\log_b\ a} \times \lg\ n)$,
- si $\exists \epsilon > 0,\ f(n) = \Omega(n^{\log_b\ a+\epsilon})$ et
  si $\exists c < 1,\ \exists n_0,\ \forall n > n_0,\ a \times f(n/b) \leq c \times f(n)$ alors $T(n) = \Theta(f(n))$

# Méthode générale [Equations récursives]

$$T(n) = a \times T(n/b) + f(n)$$

avec $a \geq 1$, $b > 1$ et $f(n)$ est positive asymptotiquement.

- si $\exists \epsilon > 0,\ f(n) = O(n^{\log_b\ a - \epsilon})$ alors $T(n) = \Theta(n^{\log_b\ a})$,
- si $f(n) = \Theta(n^{\log_b\ a})$ alors $T(n) = \Theta(n^{\log_b\ a} \times \lg\ n)$,
- si $\exists \epsilon > 0,\ f(n) = \Omega(n^{\log_b\ a + \epsilon})$ et
  si $\exists c < 1,\ \exists n_0,\ \forall n > n_0,\ a \times f(n/b) \leq c \times f(n)$ alors $T(n) = \Theta(f(n))$

Le tri par fusion ($T(n) = 2n + 1 + 2\, T(n/2)$) est dans le cas 2 :

$a = b = 2$ et $f(n) = 2n + 1 = \Theta(n)$ et $T(n) = \Theta(n\ \log\ n)$

# Méthode par substitution [Equations récursives]

$$T(n) = 2n + 1 + 2\,T(n/2) \text{ et } T(1) = 1$$

Hypothèse : $T(n) = O(n \log n) = an \log n + bn + c$

et donc $T(n/2) = a/2\; n \log n + (b - a)n/2 + c$

$$\begin{aligned} T(n) &= 2n + 1 + 2T(n/2) = 2n + 1 + an \log n + (b - a)n + 2c \\ &= an \log n + (b - a + 2)n + 2c + 1 \end{aligned}$$

(1) $b = b - a + 2$ et $a = 2$

(2) $c = 2c + 1$ et $c = -1$

(3) $T(1) = b - a + 2 + 2c + 1 = 1$ donc $b = 2$

et finalement $T(n) = 2n \log n + 2n - 1 = O(n \log n)$

# Temps de calcul [simulation]

| $Taille$ | $\log_2 n$ | $n$ | $n \log_2 n$ | $n^2$ | $2^n$ |
|---|---|---|---|---|---|
| $10$ | $0.003\,ms$ | $0.01\,ms$ | $0.03\,ms$ | $0.1\,ms$ | $1\,ms$ |
| $100$ | $0.006\,ms$ | $0.1\,ms$ | $0.6\,ms$ | $10\,ms$ | $10^{14}\,siecles$ |
| $1000$ | $0.01\,ms$ | $1\,ms$ | $10\,ms$ | $1\,s$ | |
| $10^4$ | $0.013\,ms$ | $10\,ms$ | $0.1\,s$ | $100\,s$ | |
| $10^5$ | $0.016\,ms$ | $100\,ms$ | $1.6\,s$ | $3\,heures$ | |
| $10^6$ | $0.02\,ms$ | $1\,s$ | $20\,s$ | $10\,jours$ | |

# Temps de calcul [simulation]

$nTs =$ nombre d'instructions par seconde

| $nTs$ | $2^n$ | $n^2$ | $n \log_2 n$ | $n$ | $\log_2 n$ |
|---|---|---|---|---|---|
| $10^6$ | 20 | 1000 | 63000 | $10^6$ | $10^{300000}$ |
| $10^7$ | 23 | 3162 | 600000 | $10^7$ | $10^{3000000}$ |
| $10^9$ | 30 | 31000 | $4.10^7$ | $10^9$ | |
| $10^{12}$ | 40 | $10^6$ | $3.10^{10}$ | | |