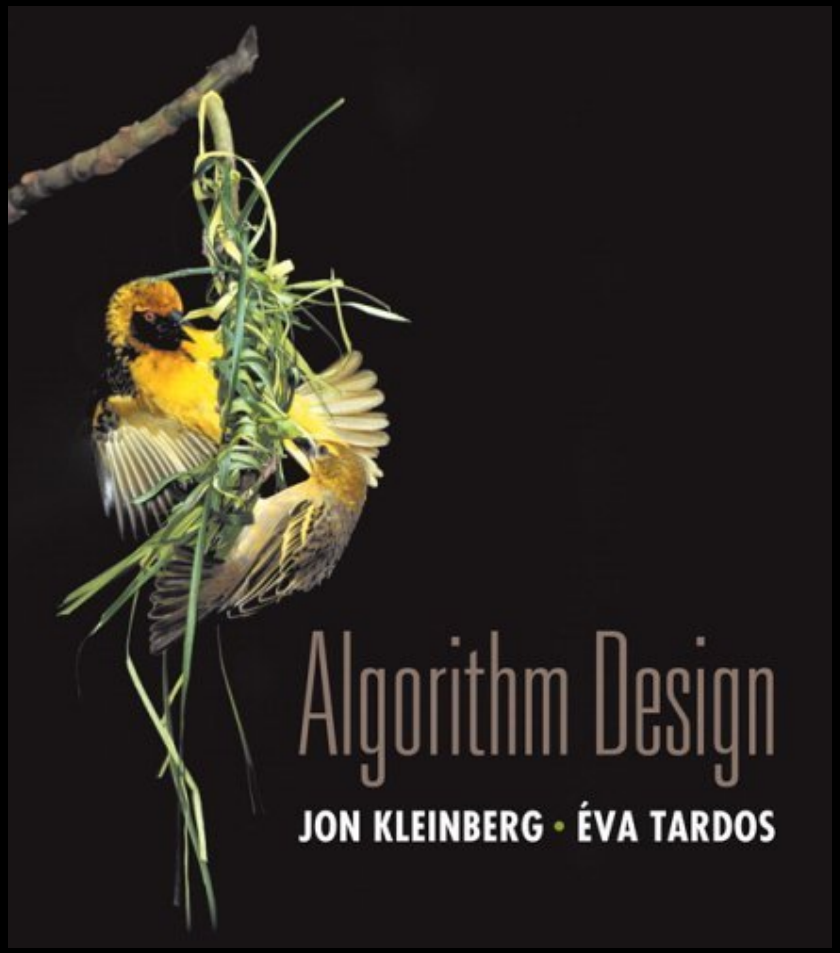


Chapitre 2

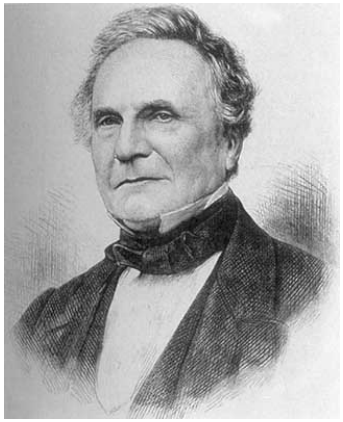
Les Bases de l'Analyse d'Algorithmes

“Faisabilité” calculatoire (Computational Tractability)

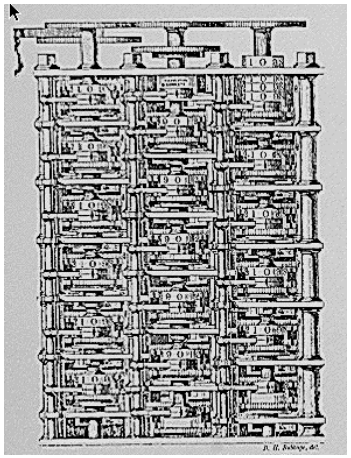
"For me, great algorithms are the poetry of computation. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some aspect of computing." - *Francis Sullivan*

Faisabilité calculatoire

As soon as an Analytic Engine exists, it will necessarily guide the future course of the science. Whenever any result is sought by its aid, the question will arise - By what course of calculation can these results be arrived at by the machine in the shortest time? - *Charles Babbage*



Charles Babbage (1864)



Machine analytique (schéma)

Temps polynomial

Force brute. Pour nombre de problèmes non triviaux, il existe un algorithme naturel de recherche brutale qui "teste" toutes les solutions possibles.

- Typiquement, cela demande un temps 2^N ou plus pour des entrées de taille N.
- Inacceptable en pratique.

n! pour le stable matching avec n hommes et n femmes

Croissance souhaitable. Quand la taille de l'entrée double, l'exécution de l'algorithme ne devrait être ralentie que d'un facteur constant.

> Il existe des constantes $c > 0$ et $d > 0$ telles que pour toute entrée de taille N, le temps de calcul (*i.e.* le nombres de "pas de calcul") est majoré par $c\,N^d$.

notion à préciser

Déf. Un algorithme est de coût (temporel) polynomial (poly-time) si son coût croît selon le schéma ci-dessus.

il suffit de choisir $c = 2^d$

Analyse dans le cas le plus défavorable (Worst-Case Analysis)

Coût d'exécution maximal. Majorer le temps de calcul sur l'entrée de taille N la plus coûteuse. Temps de calcul = fonction (N)

- En pratique, donne souvent une assez bonne idée de l'efficacité.
- Point de vue extrême, mais difficilement remplaçable.

Coût d'exécution moyen. Majorer le temps de calcul sur une entrée aléatoire de taille N. Temps moyen de calcul = fonction (N)

- Il est difficile (voire impossible) de modéliser précisément les fréquences d'apparition des entrées.
- Un algorithme mis au point pour être efficace avec une distribution particulière des entrées peut se comporter inefficacement sur des données distribuées différemment.

Temps polynomial

Déf. Un algorithme est efficace si son temps de calcul (coût temporel d'exécution) est un polynôme.

Justification : Cela marche en pratique !

- $6.02 \times 10^{23} \times N^{20}$ est un polynôme, mais un algorithme ayant ce coût temporel d'exécution ne sert (presque) à rien.
- Les algorithmes polynomiaux qu'on développe pour des applications concrètes ont des coefficients et un degré "petits".
- La possibilité de franchir la barrière exponentielle de la force brute révèle toujours quelque chose sur la structure du problème.

Exceptions.

- Certains algorithmes polynomiaux ont un degré et/ou des coefficients "grands", et sont inutilisables en pratique.
- Certains algorithmes exponentiels (ou plus coûteux) sont largement utilisés car les instances coûteuses semblent être rares.

simplex en PL, commande grep d'Unix, typage minimal en Caml

Tout cela est-il vraiment important ?

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1{,}000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10{,}000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100{,}000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1{,}000{,}000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Ordres de grandeur de croissance

Ne jamais oublier qu'en analyse d'algorithmes, c'est le comportement au voisinage de l'infini qui est important

Ordres de grandeur (notation dérivée de celle de Landau)

Les fonctions de coût sont toujours positives !

Upper bounds. T(n) est $O(f(n))$ s'il existe des constantes $c > 0$ et $n_0 \geq 0$ telles que pour tout $n \geq n_0$ on ait $T(n) \leq c \cdot f(n)$.

T(n) est dominé (asymptotiquement, en ordre de grandeur) par f(n)

Lower bounds. T(n) est $\Omega(f(n))$ s'il existe des constantes $c > 0$ et $n_0 \geq 0$ telles que pour tout $n \geq n_0$ on ait $T(n) \geq c \cdot f(n)$.

T(n) domine (asymptotiquement, en ordre de grandeur) f(n)

Tight bounds. T(n) est $\Theta(f(n))$ si T(n) est à la fois $O(f(n))$ et $\Omega(f(n))$.

T(n) a (asymptotiquement) le même ordre de grandeur que f(n)

Ex : $T(n) = 32n^2 + 17n + 32$.

- T(n) is $O(n^2)$, $O(n^3)$, $\Omega(n^2)$, $\Omega(n)$, and $\Theta(n^2)$.
- T(n) is not $O(n)$, $\Omega(n^3)$, $\Theta(n)$, or $\Theta(n^3)$.

Notation

Abus de notation. $T(n) = O(f(n))$.

- L'"égalité" dans l'expression ci-dessus n'est pas transitive :
  - $f(n) = 5n^3$; $g(n) = 3n^2$
  - $f(n) = O(n^3) = g(n)$
  - mais $f(n) \neq g(n)$.
- Meilleure notation: $T(n) \in O(f(n))$.

Le genre d'énoncé absurde à éviter. Tout algorithme de tri à base de comparaisons de clés nécessite $O(n \log n)$ comparaisons.

- Cet énoncé "ne compile pas" (incompatibilité de types.)
- C'est un Ω qu'on attend.

Propriétés

Transitivité.

- Si $f \in O(g)$ et $g \in O(h)$ alors $f \in O(h)$.
- Si $f \in \Omega(g)$ et $g \in \Omega(h)$ alors $f \in \Omega(h)$.
- Si $f \in \Theta(g)$ et $g \in \Theta(h)$ alors $f \in \Theta(h)$.

Additivité.

- Si $f \in O(h)$ et $g \in O(h)$ alors $f + g \in O(h)$.
- Si $f \in \Omega(h)$ et $g \in \Omega(h)$ alors $f + g \in \Omega(h)$.
- Si $f \in \Theta(h)$ et $g \in O(h)$ alors $f + g \in \Theta(h)$.

Ne jamais oublier que toutes les fonctions f,g,h ci-dessus sont positives !

Ordres de grandeurs de certaines fonctions "classiques"

Polynômes. $a_0 + a_1 n + \ldots + a_d n^d$ est $\Theta(n^d)$ si $a_d > 0$.

Temps polynomial. Le temps d'exécution est $O(n^d)$ pour une certaine constante d indépendante de la taille n de l'entrée.

Logarithmes. $\log_a n = \Theta(\log_b n)$ pour toutes constantes a, b > 0.

↑ on peut donc se dispenser de préciser la base

Logarithmes. Pour tout $x > 0$, $\log n \in O(n^x)$.

↑ log croît plus lentement que tout polynôme

Exponentielles. Pour tout $r > 1$ et tout $d > 0$, $n^d \in O(r^n)$.

↑ n'importe quelle exponentielle croît plus vite que tout polynôme

Un aperçu des temps de calcul “courants”

Temps linéaire : O(n)

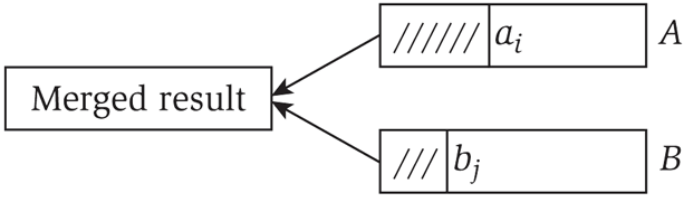
Temps linéaire. Le temps de calcul est au plus proportionnel (facteur constant) à la taille de l'entrée.

Calcul du maximum dans une liste de nombres. On cherche le plus grand nombre parmi $a_1, \ldots, a_n$.

```
max ← a_1
pour i = 2 à n {
    si (a_i > max)
        max ← a_i
}
```

Temps linéaire : O(n)

Fusion. Combiner deux listes triées $A = a_1, a_2, \ldots, a_n$ et $B = b_1, b_2, \ldots, b_m$ en une seule liste triée.




```
i ← 1, j ← 1, lo ← liste vide % lo = liste output
Tant que (une des deux listes n'est pas vide) {
   si (a_i ≤ b_j) ajouter a_i à la fin de lo et i ← i+1
   sinon          ajouter b_j à la fin de lo et j ← j+1
}
ajouter le reste de la liste non vide à la fin de lo
```

Prop. La fusion d'une liste de taille n et d'une liste de taille m coûte un temps O(n+m).

Preuve. Après chaque comparaison, la longueur de lo croît d'une unité.

Temps O(n log n)

Temps O(n log n). Apparaît souvent dans l'analyse d'algorithmes de type "diviser pour régner".

Parfois appelé en anglais linearithmic time

Tri. Mergesort et heapsort sont des algorithmes de tri qui exécutent O(n log n) comparaisons.

Calcul de la latence maximale. Etant donnés n tops d'horloge correspondant aux instants d'arrivée de copies d'un fichier à un serveur, x_1, …, x_n, quelle est la longueur maximale d'un intervalle sans arrivée de fichier ?

solution en O(n log n). Trier les tops d'horloge. Parcourir une fois la liste triée pour identifier le plus grands intervalle d'inactivité entre l'arrivée de deux copies du fichier.

Temps quadratique : $O(n^2)$

Temps quadratique. Enumérer toutes les paires d'éléments d'un ensemble.

Paire de points les plus proches. Etant donnée une liste de n points $(x_1, y_1), \ldots, (x_n, y_n)$ dans le plan euclidien, trouver la distance minimale entre deux points.

Solution en $O(n^2)$. Tester toutes les paires de points.

```
min ← (x1 - x2)² + (y1 - y2)²
pour i = 1 à n {
   pour j = i+1 à n {
      d ← (xi - xj)² + (yi - yj)²
      si (d < min)
         min ← d
   }
}
```

← Il est inutile de calculer des racines carrées

Remarque. $\Omega(n^2)$ semble inévitable, mais c'est trompeur. ← voir chapitre 5 du Kleinberg-Tardos

Temps cubique : $O(n^3)$

Temps cubique. Enumérer tous les triplets d'éléments.

Sous-ensembles disjoints. Etant donnés n sous-ensembles S_1, …, S_n de {1, 2, …, n}, y a-t-il deux sous-ensembles disjoints ?

solution en $O(n^3)$. Pour chaque paire de sous-ensembles, "décider" s'ils sont disjoints.

```
Pour tout ensemble Si {
   pour tout ensemble Sj ≠ Si {
      pour tout élément p de Si {
         déterminer si p appartient aussi à Sj
      }
      si (aucun élément de Si n'appartient à Sj)
         retourner : Si et Sj sont disjoints
   }
}
```

Temps polynomial : temps $O(n^k)$

Ensemble indépendant de taille k. Etant donné un graphe, peut-on trouver k sommets deux à deux déconnectés ?

k est une constante

solution en $O(n^k)$. Enumérer tous les sous-ensembles de k sommets.

```
Pour tout sous-ensemble S de k sommets {
   pour toute paire {x,y} de sommets de S {
      tester l'absence d'arête entre x et y
   }
   si (il n'existe pas d'arête entre deux sommets de S)
      retourner : S est un ensemble indépendant
}
```

- Vérifier que S est un ensemble indépendant $O(k^2)$.
- Nombre de sous-ensembles de k éléments =
- $O(k^2 n^k / k!) = O(n^k)$.

$$\binom{n}{k} = \frac{n(n-1)(n-2)L(n-k+1)}{k(k-1)(k-2)L(2)(1)} \le \frac{n^k}{k!}$$

Temps polynomial,
mais pas toujours acceptable (ex : k=17)

Temps exponentiel

Ensemble indépendant. Etant donné un graphe, quel est la taille maximale d'un ensemble indépendant ?

solution en $O(n^2 2^n)$. Enumérer tous les sous-ensembles.

```
S* ← ϕ
Pour tout sous-ensemble S de sommets {
   tester si S est un ensemble indépendant
   si (S est indépendant) et (S a plus de sommets que S*)
      S* ← S
   }
Retourner : le nombre d'éléments de S*
```