# 5 Analyse d'algorithmes — aspects pratiques

La notation asymptotique classique ($O$, $\Theta$, $\Omega$) est préferée dans la description théorique de l'efficacité d'algorithmes et de structures de données. Si un algorithme est de $O(n^2)$, on sait que même dans le pire cas, il va prendre un temps quadratique dans la taille du problème. Mais en pratique, on voudrait *prédire* le comportement de l'algorithme sur des *entrées typiques* (et non pas seulement le pire cas). On voudrait aussi savoir un peu plus sur les *constantes cachées* par $O$, pour pouvoir choisir entre deux algorithmes de $O(n^2)$. Par exemple, le tri rapide (quicksort) prend $O(n^2)$ dans le pire cas, mais il est en général préférable au tri par fusion (mergesort) qui est de $\Theta(n \log n)$ parce que quicksort prend $\Theta(n \log n)$ pour presque toute entrée, et les constantes cachées sont plus petites qu'en mergesort.

## 5.1 Temps de calcul comme coût

Afin de développer une caractérisation d'utilité pratique, on procède comme suit.

1. Développer un modèle de l'entrée et définir la notion de la «taille» de l'entrée. Le modèle doit permettre la génération de données à l'entrée pour mesurer la performance d'une implantation sur l'ordinateur.
2. Identifier la partie du code le plus fréquemment exécutée (qui domine la croissance du temps de calcul). Pour un algorithme itératif, cette partie est à l'intérieur de la boucle le plus profondement imbriquée.
3. Définir un modèle de coût pour le temps de calcul. En particulier, on veut écrire le temps de calcul avec le coût d'opérations typiques dans le contexte du problème. Exemples d'opération typique : comparaison de deux éléments lors du tri d'un fichier, accès à une cellule dans un tableau, ou une opération arithmétique (algorithme d'Euclid, exponentiation).
4. Déterminer la fréquence d'exécuter les opérations typiques en fonction de la taille de l'entrée.

**Exemple.** On prend l'exemple de chercher le maximum dans un tableau par itération sur les éléments (Exemple §4.1 de Note No. 4). [1] On mesure la taille de l'entrée par le nombre de ses éléments $n$ et on assume que les éléments sont dans un ordre quelconque (= permutation au hasard, uniformement distribuée). [2] La boucle intérieure contient la comparaison «**if** $x[i] > \mathsf{max}$ **then** $\mathsf{max} \leftarrow x[i]$». [3] On caractérise le temps de calcul par la fréquence de comparaisons ($x[i] > \mathsf{max}$). [4] La boucle s'exécute $(n-1)$ fois quand $n > 0$. En conclusion, l'algorithme fait $C(n) \sim n$ comparaisons arithmétiques pour un tableau de taille $n$.

### 5.1.1 Mesurer et prédire le temps de calcul

L'avantage de la notation tilde est qu'elle exprime un *hypothèse scientifique* sur le comportement de l'algorithme qu'on peut tester empiriquement. En particulier, on peut implanter l'algorithme, générer des entrées de tailles différentes, et mesurer le temps ou explicitement compter la fréquence d'exécution d'opération typiques.

**Mesurer le temps de calcul.** Le temps d'exécution peut être mesuré :

⋆ dans le code (Java) :

```
...
long T0 = System.currentTimeMillis(); // temps de début
...
long dT = System.currentTimeMillis()-T0; // temps (ms) dépassé
```

⋆ dans le shell (Linux/Unix)

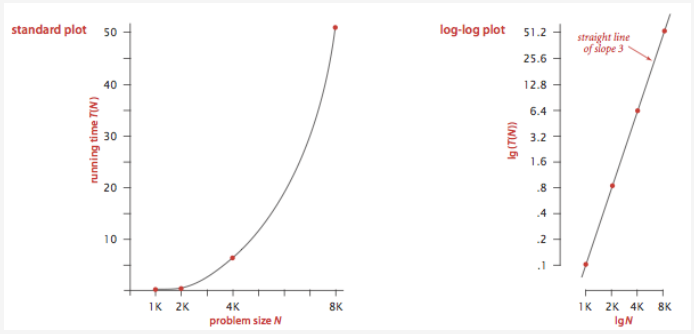
```
% time java -cp Monjar.jar mabelle.Application
0.283u 0.026s 0:00.35 85.7%  0+0k 0+53io 0pf+0w
```

temps CPU («user time»)
temps CPU («kernel time»)
temps d'exécution (seconds)
temps réel («wallclock time»)
utilisation du CPU

⋆ ou bien dans un environnement de développement de logiciel (Netbeans, Eclipse).

**Conception d'expériences.** Dans une étude empirique, on veut performer plusieurs mesures. (1) On répète l'expérience pour entrées différentes de même taille. La moyenne montre le comportement typique, et les répétitions permettent de comprendre la dispersion statistique du temps de calcul. (2) On répète l'expérience pour entrées de taille différente. Il est particulièrement utile de considérer des tailles multipliées par la même facteur (2 ou 10). Si on a l'hypothèse que le temps de calcul est $T(n) \sim a \cdot n^b$, avec des constantes quelconques $a, b > 0$, on a

$$\frac{T(2n)}{T(n)} \sim \frac{a(2n)^b}{an^b} = 2^b. \tag{5.1}$$




En fait, Equation (5.1) permet de déduire $b$ par régression linéaire même si on ne le sait pas. On a $\log T(n) \sim a' + b \log n$, et donc on peut déterminer $b$ par la pente dans un répère log-log. À gauche : temps d'éxécution d'un algorithme avec $T(N) \sim aN^3$.
Les mesures servent aussi à prédire (par extrapolation) le temps de calcul de l'implantation examinée pour des entrées de grande taille.

## 5.2 Usage de mémoire en Java