Terminaison, Validité et Complexité d'un algorithme

Trois questions doivent être posées face à un programme :

Donne-t-il un résultat ? (ou bien ne s'arrête-t-il jamais ?)

Donne-t-il le résultat attendu ? (ou bien calcule-t-il n'importe quoi ?)

Donne-t-il le résultat en un temps raisonnable ? (ou bien faut-il attendre plusieurs siècles ?)

1. Terminaison d'un algorithme

On est sûr qu'un algorithme se termine lorsque le nombre d'instructions à effectuer est connu à l'avance (avec une boucle for par exemple).

On appelle **convergent** une quantité qui prend ses valeurs dans un ensemble minoré d'entiers et qui diminue strictement à chaque passage dans une boucle. On démontre que le convergent finira par atteindre la borne inférieure de l'ensemble minoré.

Si la borne minorée est une condition d'arrêt alors on est certain que le programme se termine. Cette variable est appelée « **variant** » de boucle.

2. Validité d'un algorithme

L'utilisation **d'invariants de boucle** aide à montrer la validité d'un algorithme itératif.

On peut contrôler la validité des algorithmes utilisés dans « Les tours de Hanoï » en contrôlant que les listes A, B, C sont croissantes, de longueur majorée par N et disjointes (elles n'ont pas d'élément en commun).

Un **invariant de boucle** est une propriété qui est vérifiée avant l'entrée dans une boucle et qui reste vraie après chaque passage dans la boucle.

Attention : un algorithme peut se terminer sans être valide… et être valide sans se terminer.

3. La notion de complexité

Le terme de complexité est trompeur. On ne parle pas d'une difficulté de compréhension, mais d'efficacité : « complexe » ne veut pas dire « compliqué ».

On peut avoir des algorithmes à très faible complexité qui sont extrêmement compliqués.

L'idée de la complexité algorithmique, c'est : si je donne à mon programme une entrée de taille N, quel est l'ordre de grandeur, en fonction de N, du nombre d'opérations qu'il va effectuer ?
Remarque : il faut aussi définir ce qu'est une opération.

Idée à retenir :
un algorithme de forte complexité est plus lent pour traiter un grand nombre de données.

4. Notation " grand O"

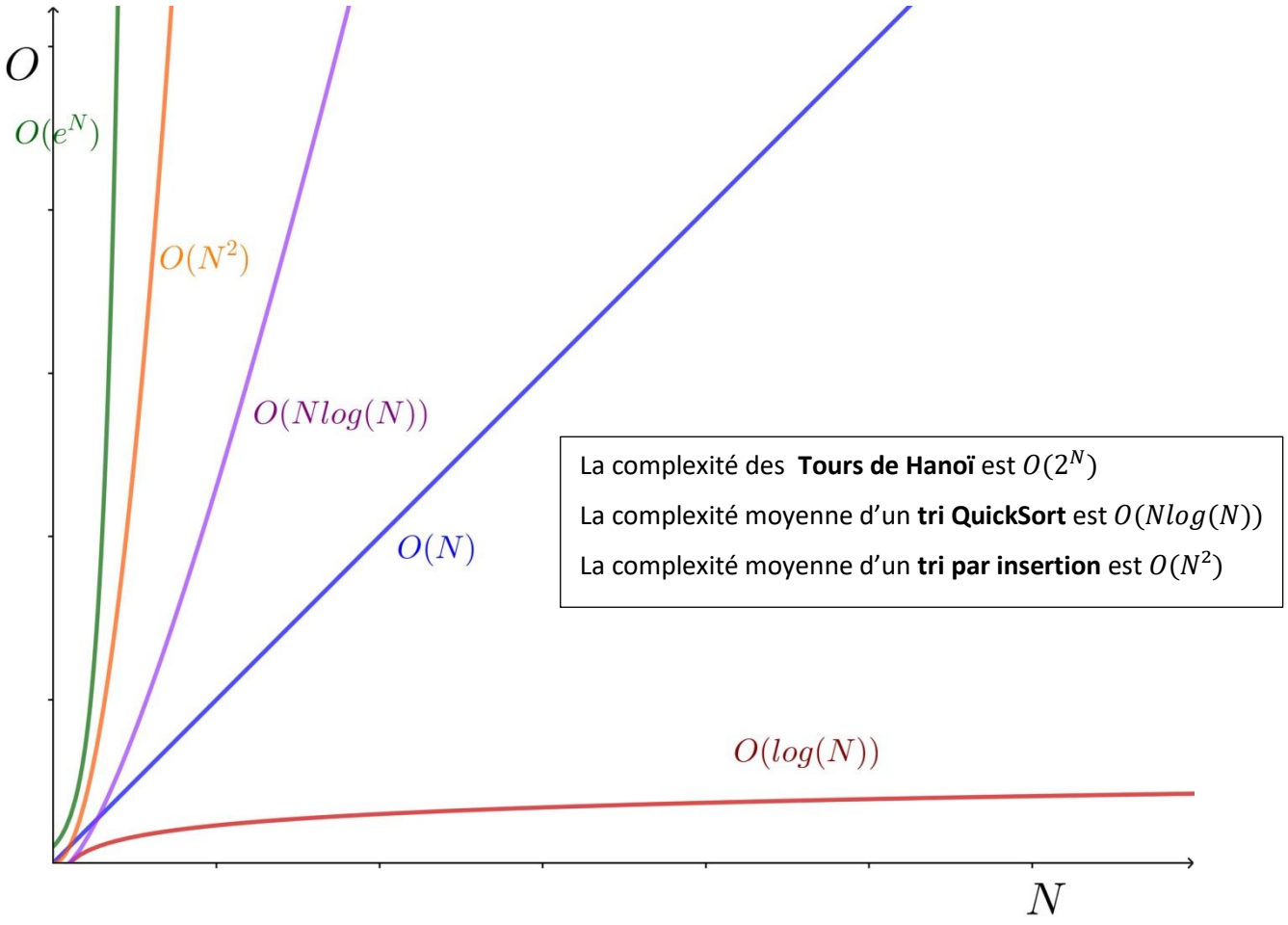
Pour représenter la complexité on utilise une notation spécifique, la notation O.

La notation O permet de classer ensemble des nombres d'opérations qui ont le même ordre de grandeur.

Exemples pour un programme ayant une entrée de taille N :

- Des algorithmes effectuant environ N opérations, $2N-1$ opérations ou $N/2$ opérations ont tous la même complexité : on la note $O(N)$ (lire « grand O de N »).
- Un algorithme effectuant $2N^2 + 3N + 5$ opérations aura une complexité de $O(N^2)$: on néglige les termes $3N$ et 5 qui sont de plus petits degrés que $2N^2$, et donc croissent moins vite.

5. Illustration graphique des différentes classes de complexité



La complexité des **Tours de Hanoï** est $O(2^N)$

La complexité moyenne d'un **tri QuickSort** est $O(Nlog(N))$

La complexité moyenne d'un **tri par insertion** est $O(N^2)$