Introduction : algorithme et complexité

1. Algorithme

« L'épreuve est principalement axée sur l'écriture de petits programmes (d'environ moins de 20 lignes) et l'apprentissage de la pensée algorithmique. »

a. Définition

Algorithme = suite finie d'instructions univoques (=ne pouvant pas être interprétées de plusieurs façons) **pouvant être exécutées de manière automatique** (= par une machine).

Exemple : une recette de cuisine…

b. Comment décrire un algorithme ?

i. **Principe** de l'algorithme, écrit en français.
ii. Pseudo-code (nous ne nous en servirons pas ici)
iii. **Programmation** en langage informatique (ici, implémentation en MAPLE) .

Dans une bonne pratique de l'algorithmique, ces trois étapes sont nécessaires. Cependant, les langages de « haut niveau » (c'est-à-dire proches du langage humain, par ex : MAPLE) permettent de se passer de l'étape ii, surtout pour des algorithmes courts. C'est beaucoup plus dangereux avec des langages de plus bas niveau (c'est-à-dire plus proche du langage machine).

c. Qu'est-ce qu'un bon algorithme ?

Un bon algorithme doit être :

i. **Juste**. L'algorithme doit répondre au problème posé. On peut faire la *preuve* d'un algorithme (non demandé au concours).
ii. **Simple**, à écrire et à comprendre. En particulier, un bon algorithme doit être commenté (i.e., on doit expliquer les différentes étapes). Ceci est évidemment valide pour l'épreuve du concours, mais également pour tout programme industriel, pour limiter les coûts de maintenance et de mise à jour.
iii. **Rapide** (en temps d'exécution) . C'est le problème de la **complexité** de l'algorithme (cf. §2). Dans la pratique, le cryptage de données ne repose pas sur le fait qu'il soit impossible de trouver un code donnée, mais plutôt que les algorithmes existant pour le « craquer » prennent en moyenne plusieurs milliers d'années et sont donc inutiles
iv. Econome de la **mémoire** : les capacités d'une machine sont grandes mais finies ! Attention : les algorithmes récursifs sont très gourmands.
v. Sophistiqué. L'algorithme prend en compte des cas particuliers, vérifie le type des données en entrée… Ceci est très important dans le cas d'algorithmes commerciaux.

Au concours, l'accent est mis sur **la justesse et la simplicité** de l'algorithme. Comme on ne demande pas de prouver les algorithmes proposés, il est évident qu'il faut que ces algorithmes soient simples pour que le correcteur puisse comprendre sans migraine les algorithmes des 100 candidats. La rapidité de l'algorithme peut être prise en compte mais ce n'est pas une priorité pour cette épreuve.

2. Complexité

« On se limitera à des raisonnements simples pour l'évaluation de la complexité des algorithmes : il s'agira d'être capable de compter le nombre d'opérations dans des boucles itératives. Il n'est pas question de faire des évaluations élaborées faisant intervenir la théorie de la complexité. »

a. Quelles sont les **opérations élémentaires** ?
   i. Accès mémoire (lire ou écrire une variable ou un élément d'un tableau)
   ii. Opération arithmétique (addition, soustraction, multiplication, division, division euclidienne)
   iii. Comparaisons (opérations booléennes)
   iv. Exemple

```
> a := 3; b := 4; c := a + b;
```

4 opérations élémentaires : 2 accès mémoire pour lire les valeurs de a et b, une addition, et un accès mémoire (affectation de c)

b. Qu'est-ce que l'ordre de grandeur d'une fonction ?
   i. Soient f et g deux fonctions définies au voisinage de l'infini. **L'ordre de grandeur** de f est inférieur ou égal à celui de g si f=O(g).

      Exemple : ln(n) est d'ordre de grandeur inférieur à n^2.

   ii. f et g ont le **même ordre de grandeur** si f=O(g) et g=O(f)

      Exemple : n^2 et 2n^2 ont le même ordre de grandeur (NB : ils ne sont pas équivalents pour autant !)

c. Qu'est-ce que la **complexité** d'un algorithme ?
   i. Le **nombre d'opérations élémentaire**s effectuées lors de l'exécution d'un algorithme dépend de manière croissante du **nombre N de données traitées** (exemple : tri d'un tableau contenant N entrées), mais également des données elles-mêmes (il est plus rapide de trier un tableau presque déjà trié qu'un tableau plus petit très désordonné).
   ii. Supposons qu'on puisse déterminer la **borne supérieure** (fonction de N) du nombre d'opérations élémentaires « dans le pire des cas ». La **complexité** de l'algorithme est un **ordre de grandeur**, lorsque N tend vers l'infini, de cette borne supérieure.

d. Exemple
   La recherche séquentielle d'un élément dans un tableau nécessite dans le pire des cas, N accès mémoire et N comparaisons, soit 2N opérations élémentaires. On dira que l'algorithme est en O(N), ou « en temps linéaire ».

e. Dans le *meilleur cas*, l'élément cherché est le premier terme du tableau et la complexité « dans le meilleur cas » est de 1. Sauf exception, on demande au concours d'estimer la complexité dans le pire des cas.