# Rappels sur la complexité des algorithmes

**Exercice 1** A titre d'illustration des notions de taille de problèmes et de nombre d'opérations, on considère un vecteur $x = (x_1, x_2, \ldots, x_n)$ de $n$ composantes, où chaque composante $x_i$ peut prendre les valeurs 0 ou 1. Dans tous les exemples qui suivent la taille du problème est la taille du vecteur $x$, c'est à dire $n$. (Ici, la taille du problème est donc le nombre de composantes du vecteur $x$, mais pour d'autres problèmes, cela pourrait être un nombre de clients à traiter, un nombre de villes à relier entre elles par des lignes téléphoniques, bref c'est un moyen de dimensionner le problème).

Pour chacun des algorithmes suivants évaluer le nombre d'opérations :

```
Algo 1 :
Debut
    Pour i allant de 1 à n faire
       afficher(x[i])
    fait
Fin

Algo 2 :
Debut
    Pour i allant de 1 à n faire
       Pour j allant de 1 à n faire
         afficher(x[i]+x[j])
       fait
    fait
Fin

Algo 3:
Debut
    Pour i allant de 1 à n faire
       Pour j allant de 1 à n faire
         Pour k allant de 1 à n faire
             Pour l allant de 1 à n faire
                  Pour m allant de 1 à n faire
                      afficher(x[i]+x[j]+x[k]+x[l]+x[m])
                  fait
             fait
         fait
      fait
    fait
Fin
```

Affichage de toutes les valeurs possibles du vecteur x.
Exemple un appel de l'algorithme enumeration(3,1,x) :

$$x = (0,0,0) \quad x = (0,0,1) \quad x = (0,1,0) \quad x = (0,1,1)$$
$$x = (1,0,0) \quad x = (1,0,1) \quad x = (1,1,0) \quad x = (1,1,1)$$

```
Algo 4 :
enumeration (n,i : entier, x : vecteur)
Debut
         si  (i>n)
            alors afficher_vecteur(x)
         sinon x[i] := 0;
```

```
            enumeration(n,i+1,x);
            x[i] := 1;
            enumeration(n,i+1,x);
         finsi
fin
```

**Exercice 2** Soient les 3 algorithmes suivants permettant de calculer la valeur d'un polynôme

$$p(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n$$

en un point $x$. Comparer leur complexité. (La valeur de $x$ est donnée).

Algo 1 :
```
début
    p := a[0] ;
    pour i := 1  à  n  faire
      calculer p[i] = x[i] ;   -- x*x*x*..*x  i-1 fois
      p := p + a[i] * p[i] ;
    fait;
fin;
```

Algo 2 : – il utilise le fait que $x^i = x^{i-1} * x$
```
Début
    p := a[0];   q := 1;
    pour i := 1  à  n  faire
      q := q * x ;
      p := p  +  a[i] * q ;
    fait;
fin;
```

Algo 3 : –algorithme de Horner ; il calcule à reculons $a_n, a_nx+a_{n-1}$, puis $(a_nx+a_{n-1})x+a_{n-2}\ldots$
```
début
    p := a[n] ;
    pour i := n  à  1, pas -1, faire
      p := p*x + a[i-1] ;
    fait;
fin;
```

**Exercice 3** Rappel : soient 2 fonctions $f$ et $g$ de $\mathbb{N}$ dans $\mathbb{N}$ : $f = O(g)$ si il existe $n_0 \in \mathbb{N}$ et $c$ constante $> 0$, tels que $f(n) \leqslant cg(n)$ pour tout $n \geqslant n_0$

(1) Soit un algorithme demandant exactement $f(n) = 12n + 7$ opérations, montrer que cet algorithme est en $O(n)$.

(2) De même, montrer que si $f(n) = a_0n^p + a_1n^{p-1} + a_2n^{p-2} + \ldots + a_{p-1}n + a_p$, et $a_0 > 0$ alors $f = O(n^p)$.

**Exercice 4** On considère deux manières de représenter ce que l'on appelle des "matrices creuses", c'est-à-dire des matrices d'entiers contenant un grand nombre d'éléments nuls :

a) La matrice est représentée par un tableau à deux dimensions dont les cases contiennent les éléments.

b) La matrice est représentée par un tableau à une dimension. On ne s'intéresse qu'aux $m$ éléments de la matrice qui ne sont pas nuls. Chaque case du tableau contient un triplet $(i, j, v)$ correspondant à l'indice de ligne $i$, l'indice de colonne $j$, et la valeur d'un élément non nul.

Le problème considéré consiste à calculer la somme des éléments d'une matrice. On demande d'écrire un algorithme permettant de calculer cette somme, pour chacune des deux représentations, puis de comparer leur complexité. Que peut-on conclure de cette comparaison ?