E.D. NFP 136 n°2

Thème : Algorithmique et Complexité

Exercice I.1

Rappel : soient 2 fonctions f et g de **N** *dans* **N**. *On dit que f = O(g) s'il existe n0 dans* **N** *et c constante>0, tels que f(n) ≤ c g(n) pour tout n≥n0 (n0 > 1 suffit, en général).*

Question 1

Soit un algorithme dont l'exécution nécessite exactement f(n) opérations dans le pire cas :

(a) Montrer que, si f(n) = 12n+7, alors f(n) = O(n) (algorithme dit « linéaire »).

(b) Montrer ensuite que, si $f(n) = a_0n^p+a_1n^{p-1}+a_2n^{p-2}+...+a_{p-1}n+a_p$, pour un nombre constant p+1 de réels a_i, alors $f(n) = O(n^p)$ (algorithme dit « polynomial »).

Question 2

Soient les deux algorithmes suivants permettant de calculer la valeur d'un polynôme

$$P(x) = a0 + a1x + a2x^2 + \ldots + anx^n$$

en un point x (c'est-à-dire pour une valeur x donnée).

```
algo 1 //algorithme « naïf »
début
      p := a0 ;
      pour i := 1  à  n  faire
            calculer pi = x^i ;   //x.x.x...x  i-1 fois
            p := p + ai pi ;
      fait;
fin;
```

```
algo 2      //utilise le fait que x^i = x^(i-1) * x
début
      p := a0;   q := 1;
      pour i := 1  à  n  faire
            q := q * x ;
            p := p  +  ai * q ;
      fait;
fin;
```

Comparer leur complexité au pire cas. (Note : la valeur x est donnée, donc connue.)

Exercice I.2 De l'intérêt d'améliorer la taille des ordinateurs

Question 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 (représenté par exemple par un tableau), 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, etc. ; en d'autres termes, c'est un moyen de dimensionner le problème.)

On vous demande d'évaluer le nombre d'opérations des deux algorithmes suivants, en considérant que l'affichage d'une valeur compte comme une opération élémentaire :

```
Algo 1
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(xi+xj+xk+xl+xm)
                fait
           fait
      fait
 fait
fait
```

Algo 2 : Affichage de toutes les valeurs possibles du vecteur x.

Exemple pour n=3 :

x = (0, 0, 0) x = (0, 0, 1) x = (0, 1, 0) x = (0, 1, 1)
x = (1, 0, 0) x = (1, 0, 1) x = (1, 1, 0) x = (1, 1, 1)

```
Algo enumeration (n,i : entier, x : vecteur)
Debut
     si  (i>n)
          alors afficher(x)
          sinon     xi := 0;
                    enumeration(n,i+1,x);
                    xi := 1;
                    enumeration(n,i+1,x);
     finsi
fin
```

Appel de l'algorithme :

```
enumeration(3,1,x);
```

Question 2

Compléter le tableau ci-dessous :

Nombre d'opérations en fonction de la taille n des Données	**Avec un ordinateur X**		**Avec un ordinateur Y 100 fois plus rapide que X**	
	Taille maximale (n max) des problèmes traités en 1h	Nombre d'opérations effectuées en 1h	Taille maximale (n max) des problèmes traités en 1h	Nombre d'opérations effectuées en 1h
N	N1	N1	N'1 = 100 N1	
n^2	N2		N'2	
n^5	N3		N'3	
2^n	N4		N'4	

Exercice I.3

Soit l'algorithme suivant qui effectue la recherche dichotomique du rang (place) d'un nombre v dans une suite triée (par ordre croissant) de n nombres mis dans un tableau à une dimension (tableau t[i], pour i=1,..,n). (Cet algorithme fonctionne sous l'hypothèse que v est effectivement présent dans la liste.)

```
Algorithme :
début
      place = 1 ; f = n ;
      tant que  place < f  faire
            milieu = (place + f) / 2 ;
            si  t[milieu] < v  alors
                  place = milieu + 1 ;
            sinon f = milieu ;
            finsi
      fait;
fin;
```

Question

Donner la complexité (en temps) au pire cas de cet algorithme, ainsi que sa complexité en espace, puis écrire sa version récursive.