CORRIGÉ de l'ED1
À PROPOS DE COMPLEXITÉ

On rappelle les définitions suivantes :
Soient f et g sont deux fonctions de N dans N.
On dit que **f = O(g)** s'il existe n0 ∈ N et c constante, tels que f(n) ≤ c * g(n) pour tout n ≥ n0.
De même, on dit que **f = Θ (g)** s'il existe n0 ∈ N et c1, c2 constantes, tels que c1 * g(n) ≤ f(n) ≤ c2 * g(n) pour tout n ≥ n0.

A- *Expliquer les graphes de la figure 1, et placer la valeur n0.*

La fig. de gauche correspond au cas où f(n) est "dominée" par c*g(n) à partir d'une certaine valeur n0 de n : la fonction f est donc "en O(g)", ou : f = O(g)

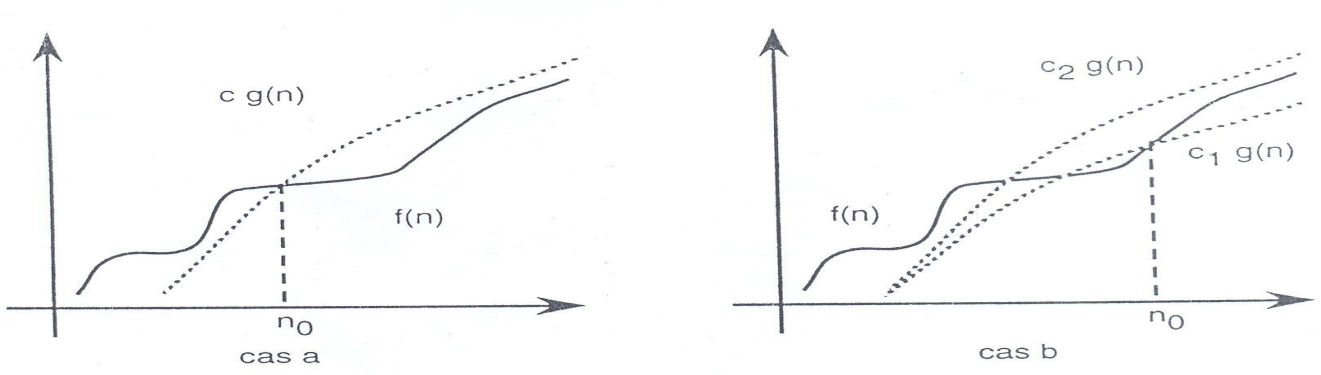

Figure — Graphes de complexité (solution).

La fig. de droite correspond au cas où, à partir d'une certaine valeur n0 de n, la fonction f est "encadrée" par c1*g(n) et c2*g(n) : f est alors du même ordre de grandeur que g, ou : **f = Θ (g)**

B.1- *Soit un algorithme demandant exactement f(n)=12n+7 opérations; montrer que cet algorithme est en O(n) et en Θ (n).*

a. La droite y = **13*n** "monte plus vite" que f(n) : elle apparaîtra donc "au-dessus" à partir d'un certain rang. Plus précisément, 12n+7 ≤ 13n *ssi* 7 ≤ n
   En prenant : c = 13 et n0 = 7, on aura donc : pour tout n ≥ n0, f(n) ≤ c * n, soit : f = O(n). Ici, g est la fonction Identité (g(n) = n).

b. Et, quel que soit n : **12n** ≤ 12n+7.
   Donc, en prenant : c1 = 12, c2 = 13, n0 = 7 : pour tout n ≥ n0, 12n ≤ f(n) ≤ 13n, soit : **f = Θ (n)**, avec ici encore la même fonction Identité pour g.

B.2- *De même, montrer que si f(n) = a0*n^p+a1*n^{p-1} +a2 * n^{p-2} + **...** + a_{p-1} * n + a_p, avec a0 positif, alors f = O(n^p) et f = Θ (n^p).* [=> seul importe le terme de "+ fort degré" en ***n***...]

Posons : g(n) = n^p
Quand n → ∞ , lim (f(n) / g(n)) = a0 > 0
Donc : quel que soit ε > 0, ∃ n0 ∈ N tel que : a0 - ε ≤ { f(n) / g(n) } ≤ a0 + ε
Prenons : ε = a0 / 2; il vient alors : a0 / 2 ≤ { f(n) / g(n) } ≤ 3*a0 / 2
Posant : c1 = a0 / 2 et c2 = 3*a0 / 2, le résultat est dès lors acquis (f = **Θ (g)** impliquant toujours **f = O(g)**)

C- *On considère trois algorithmes de calcul de la valeur d'un polynôme en un point, soit* $p(x) = a0 + a1 * x + a2 * x^2 + \ ... \ + an * x^n$. *Comparer leur complexité.*

début - - premier algorithme
p := a0;
pour i de 1 à n faire
  calculer pi = x^i*;* *- - x . x . x .... x (x multiplié i fois)*
  *p := p + ai * pi;*
fait;
fin;

début - - deuxième algorithme
p := a0; q := 1;
pour i de 1 à n faire
  *q := q * x;* *- - fondé sur la propriété :* $x^i = x^{i-1} * x$
  *p := p + ai * q;*
fait;
fin;

début - - troisième algorithme (Horner)
p := an;
pour i inverse de 1 à n faire
  p := $p * x + a_{i-1}$*;*
fait;
fin;

Le paramètre de taille du problème est ici **n**, le degré du polynôme traité.

Les opérations fondamentales en jeu dans ces algorithmes sont :
la multiplication, l'addition et l'affectation de réels.
Nous allons donc dénombrer les opérations de ces 3 types, sachant que les instructions du corps de boucle sont exécutées n fois; on en déduira l'ordre de la complexité d'après le résultat de B-2.

Algorithme	**Nb. de ***		**Nb. de +**		**Nb d'affect.**		**Complexité résultante**
	pas i	**Total**	***pas i***	**Total**	***pas i***	**Total**	
1	(i-1) + 1 = i	n*(n+1)/2	1	n	1	1 + n	**Θ (n²)**
2	2	2n	1	n	2	2 + 2n	**Θ (n)**
3 *(Horner)*	1	n	1	n	1	1 + n	**Θ (n)**

L'algorithme 1 est à proscrire (il est "en $\mathbf{n^2}$ "). Il recalcule inutilement chaque monôme sans exploiter les résultats antérieurs.
Les 2 et 3 sont de même ordre (linéaire). Horner sera cependant plus efficace, exécutant environ 2 fois moins d'opérations que (2) (de plus, il fournit numériquement de "meilleurs" résultats...)

D- *On dispose de quatre algorithmes, dont le nombre d'opérations en fonction de la taille des données est respectivement :* n, n^2, n^5 *et* 2^n.

On note : N1, N2, N3 et N4 la taille maximale du problème pouvant être traité en 1 heure, pour chacun des 4 algorithmes, par la machine la plus performante actuelle.
Déterminer quelles seront ces tailles lorsque l'on disposera d'une machine 100 fois plus rapide ?

Soit t le temps d'exécution d'une opération sur la *machine la plus performante actuelle*.
Dire qu'une nouvelle machine est *100 fois + rapide* revient à dire que ce temps élémentaire devient : **t / 100**
Notons alors N' les tailles maximales admissibles sur la nouvelle machine pour 1 h de calcul.
Pour un algorithme de complexité f(n), l'équation déterminant N' s'écrit alors :
1 heure = (t/100) * f(**N'**) = t * f(N) => f(**N'**) = 100 * f(N) => **N'** = f^{-1} (100 * f(N))

D'où le tableau suivant :

	COMPLEXITÉS *(f(n))*				
	n	**n^2**	**n^5**	**2^n**	
Équations	N'1 = 100 * N1	$N'_2{}^2 = 100 * N_2{}^2$	$N'_3{}^5 = 100 * N_3{}^5$	$2^{N'_4} = 100 * 2^{N_4}$	
f^{-1} (y)	y	SQRT (y)	$\sqrt[5]{(y)}$	$\log_2$ (y)	
Résultats (**N'**)	N'1 = **100 *** N1	$N'_2 =$ **10 *** N_2	$N'_3 = \sqrt[5]{(100)} * N_3$ = **2,5 *** N_3	$N'_4 = \log_2 (100) + N_4$ = **6,6** + N4	

Dans le cas linéaire, on exploite pleinement le gain de puissance. Il reste bon dans le cas quadratique, mais s'affaiblit considérablement dans le cas (3). Dans le cas exponentiel, le gain est purement symbolique...