Solution TD1

Correction 1-

d'apres l'ordre de grandeur generer la valeur de complexité de l'algorithme

- Pour l'algorithme en $\mathcal{O}(n^2)$, on a $100n_0^2 = n_1^2$, et donc $n_1 = \sqrt{100} \times n_0$
- Pour l'algorithme en $\mathcal{O}(2^n)$, on a $100 \cdot 2^{n_0} = 2^{n_1}$, et donc $n_1 = \log 100 + n_0$

Correction 2 -

- Vraies : 1 $(c = 1000, n_0 = 0)$; 3 $(c = 1, n_0 = 0)$; 4 $(c = 2, n_0 = 0)$; 5 $(c_1 = 1, c_2 = 3, n_0 = 1)$; 6 $(c = 5, n_0 = \left\lceil \sqrt[3]{2017} \right\rceil)$; 9 $(c_1 = \frac{1}{4}, c_2 = 1, n_0 = 12)$; 10 pour toute fonction $f(n) = \frac{1}{4}n^2 * \Omega(n)$, il existe c_1 t.q. $f(n) \geq \frac{c_1}{4}n^3$, on peut prouver $f(n) \in \Omega(n^3)$ en choisissant $c = \frac{c_1}{5}$ et $n_0 = 0$.
- Fausses : 2 $(cn^3 > n^2$ pour tout $n > \frac{1}{c})$; 7 $(n^5 > cn^3$ pour tout c si $n > \sqrt{c})$; 8 (passage au log : $2^n > \log(c) + n$ et on prend $n \geq \log(c)$)

Correction 3 -

- 1. est vrai :
  - $\Rightarrow$: $\exists c \exists n_0 \forall n \geq n_0\ f(n) \geq cg(n)$ donc $\forall n \geq n_0\ g(n) \leq \frac{1}{c}f(n)$;
  - $\Leftarrow$: similaire
- 2. est fausse : par exemple avec $f(n) = n$ et $g(n) = n * n^{\sin(n)}$ (il suffit de schématiser deux fonctions ayant ce type de comportement)

Correction 4 -

1. $f(n)=5n^4+3n^3+2n^2+4n+1 < (5+3+2+4+1)n^4=cn^4$

Donc $f(n) \in O(n^4)$ pour c=15 et $n \geq n0=1$

2. $5n^2+3n \log n+2n+5 < (5+3+2+5)n^2=cn^2$

Donc $f(n) \in O(n^2)$ pour c=15 et $n \geq n0=2$

3. $2^{n+2} = 2^n 2^2 = 4.2^n = c.2^n$

Donc $f(n) \in O(2^n)$ pour c=4 et $n \geq n0=1$

Correction 5 -

a.

$$y_n = x_n + \sum_{i=1}^{n} x_i = x_n + y_{n-1} \text{ avec } y_0 = 0$$

$$y_n = \prod_{i=1}^{n} x_i = x_n y_{n-1} \text{ avec } y_0 = 1$$

$$y_n = \sum_{i=0}^{n} \frac{(-1)^i x^i}{i!} = \frac{(-1)^n x^n}{n!} + y_{n-1} \text{ avec } y_0 = 1$$

b.

1. Solution simple Cette définition calcule,:

$$H(n) = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \ldots + \frac{1}{n}$$

2. La fonction n'est pas récursive terminale car la valeur de retour de l'appel récursif est utilisée dans une expression qui devient la valeur de retour de l'appel courant.

Formellement, une version récursive terminale de la fonction est:

$$H(n, a) = \begin{cases} a + 1 & \text{si } n = 1 \\ H\ (n - 1, a + \frac{1}{n}) & \text{si } n > 1 \end{cases}$$

Correction 6 –

Dans les deux cas, on peut remarquer que l'instruction la plus souvent exécutée (terme dominant) est "$r \leftarrow r + 1$" avec r initialisé à 0 et retourné par l'algorithme.

Donc la complexité de ces algorithmes est du même ordre de grandeur que la valeur retournée.

- L'algorithme f_1 retourne $\sum_{i=1}^{n} \sum_{j=i+1}^{n} 1 = \sum_{i=1}^{n} (n - i) = n^2 - \sum_{i=1}^{n} i = n^2 - \frac{1}{2}n(n+1) = \frac{1}{2}n(n-1)$: la somme des $n-1$ premiers entiers positifs
- L'algorithme f_2 passe $\lceil \log_2 n \rceil$ fois dans la boucle puisque i est divisé par 2 à chaque itération. La valeur retournée est donc $r = \lceil \log_2 n \rceil$.

2. **Prouver que la complexité** de l'algorithme de tri par sélection= $O(n^2)$:

Algorithme TRI-SELECTION(T : tableau [1..N] d'entiers, entier N)

w(A)=w(1)

$$= \sum_{i=1}^{n-1} (w2 + w3 + w6) \quad = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (w4 + w5)$$

$$= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} (O(1))$$

$$= \sum_{i=1}^{n-1} (\text{n} - \text{i})$$

$$= n(n-1)/2$$

=> la complexité est $\Theta(n^2)$

- **Prouver que la complexité** de l'algorithme de tri par fusion = O(nlog n) :

Algorithme TRI-FUSION (T : tableau [1..N] d'entiers ; N , l, r : entiers)
Var
m: entier;
Dédut
si (l < r) alors m ← [(l + r)/2] ; finsi ;
TRI-FUSION(T ; l ; m) ;
TRI-FUSION(T; m+1, r) ;
fusion(T; l; r; m) ;
fin.

- La complexité de l'algorithme récursif est donnée par son équation,
Le coût de la récursivité est donné par T(m) tel que :

g(m), le coût d'un appel récursif) => tri par fusion (le cout de l'utilisation des registres temporaire est linéaire) => g(m)=O(n)
=> T(m) = 2.T([n/2]) + O(n) d'après les relations de récurrences (Cours) :
T(n) = c.T(n/d) + O(n^k)
c=2, d=2 et k=1 => $c=d^k$ ، alors T(n) = O(nlog n)

Correction 7 –
le nombre d'instruction est égale à:

$$\sum_{i=1}^{n-1}\sum_{j=i+1}^{n}\sum_{k=1}^{j}\Theta(1)$$

=> ordre de grandeur est O(n^3)

Correction 8 –

1. la complexité des algorithmes itératifs suivants, en fonction de deux paramètres, où m et n sont deux entiers positifs.

A. par rapport au nombre d'itérations effectuées
Le corps des différents algorithmes ne contient que des opérations en O(1). En comptant le nombre d'itérations effectuées par chacun des algorithmes, on obtient immédiatement que :

- l'algorithme A est en $O(\min(m, n))$;
- l'algorithme B est en $O(\max(m, n))$;
- l'algorithme C est en $O(m + n)$;
- l'algorithme D est en $O(mn)$.

B. par rapport aux opérations arithmétiques

- l'algorithme E : w(A) = w(1) + w(2) + w(3) + w(7)

$$= 0 + 0 + 0 + \sum_{i=1}^{n/2}(w(4) + w(6))$$

$$= \sum_{i=1}^{\lceil \frac{n}{2} \rceil}\sum_{j=5}^{n^2} w(5) = \sum_{i=1}^{\lceil \frac{n}{2} \rceil}\sum_{j=5}^{n^2} 1$$

$$= \sum_{i=1}^{\lceil \frac{n}{2} \rceil} (n^2 - 5 + 1) = \sum_{i=1}^{\lceil \frac{n}{2} \rceil} (n^2 - 4)$$

$$= \lceil \frac{n}{2} \rceil \cdot (n^2 - 4) = O(n^3)$$

- l'algorithme F :

$$\sum_{i=1}^{n-1}\left(4 + 2\cdot\sum_{j=i+1}^{n} 1\right) = 2\cdot\left(\sum_{i=1}^{n}(n-i)\right) + 4\cdot(n-1)$$

$$= 2\cdot\left(\sum_{i=1}^{n} n\right) - 2\cdot\left(\sum_{i=1}^{n} i\right) + 4\cdot(n-1)$$

$$= 2n(n-1) - 2\cdot\frac{n\cdot(n-1)}{2} + 4\cdot(n-1) = n(n-1) + 4(n-1)$$

$$= O(n^2)$$