

Solutions¹ Chapitre 2

*Question # 1

Utilisez la notation la plus appropriée parmi O , Ω et Θ pour indiquer la classe d'efficacité (par rapport au temps) de la recherche séquentielle ci-dessous dans les différents cas.

Algorithme 1 : RechercheSequentielle($A[0..n - 1], d$)

```

1 pour  $i = 0..n - 1$  faire
2   si  $A[i] = d$  alors
3     retourner  $i$ 
4 retourner  $-1$ 
```

A) Pire cas

Solution :

Puisque $C_{worst}(n) = n$, on a que $C_{worst}(n) \in \Theta(n)$.

B) Meilleur cas

Solution :

Puisque $C_{best}(n) = 1$, on a que $C_{best}(n) \in \Theta(1)$.

C) Cas moyen

Solution :

Aucune distribution de probabilité n'est donné sur les instances possibles. Nous allons donc en définir une. Soit p la probabilité que l'élément recherché d est dans le vecteur A . Dans le cas où l'élément recherché est dans le vecteur A , la probabilité que l'élément se retrouve à la position i est uniforme, soit $\frac{1}{n}$.

$$C_{avg}(n) = \sum_{x:|x|=n} P(x)C(x) \quad (1)$$

Il y a $n + 1$ instances de taille n . Une instance pour chaque position dans le vecteur où l'élément recherché peut se trouver, plus une instance où l'élément recherché n'est pas dans le vecteur.

$$= \sum_{i=0}^{n-1} P(A[i] = d)C(A[i] = d) + P(d \notin A)C(d \notin A) \quad (2)$$

$$= \sum_{i=0}^{n-1} \frac{p}{n}(i + 1) + (1 - p)n \quad (3)$$

$$= \frac{p}{n} \sum_{i=1}^n i + (1 - p)n \quad (4)$$

$$= \frac{p}{n} \frac{n(n + 1)}{2} + (1 - p)n \quad (5)$$

$$= p \frac{n + 1}{2} + (1 - p)n \quad (6)$$

1. Dans le but d'alléger les démonstrations, nous utilisons $f(n) \leq cg(n)$ comme définition de $f(n) \in O(g(n))$. Nous traitons implicitement le fait qu'il existe une certaine constante c et que l'inégalité est vraie pour tout $n \geq n_0$ pour un certain n_0 . Nous nous permettons de faire cette simplification car l'idée des preuves se situe rarement à l'intérieur des constantes c et n_0 .

***Question # 2**

En vous appuyant uniquement sur notre analyse de l'algorithme d'Euclide effectuée au chap 1, exprimez l'efficacité de l'algorithme d'Euclide en pire cas à l'aide de la notation asymptotique.

Solution :

Rappel : la taille s de l'instance est le nombre de bits utilisé par les deux entiers (m,n) à l'entrée de l'algorithme d'Euclide. Notre analyse du chap 1 nous a donné $C_{worst}(s) \in O(s)$. Alors $C(s) \in O(s)$. Le(s) pire cas n'a pas été identifié et nous n'avons pu conclure que $C_{worst}(s) \in \Theta(s)$. Ainsi notre analyse ne nous a pas permis de conclure que la borne asymptotique supérieure est optimale.

Question # 3

Démontrez la règle du maximum pour Ω .

Solution :

$$\begin{aligned}
& t_1(n) \in \Omega(g_1(n)) \wedge t_2(n) \in \Omega(g_2(n)) \\
\Leftrightarrow & \quad \langle \text{Définition de } \Omega \rangle \\
& t_1(n) \geq c_1 g_1(n) \wedge t_2(n) \geq c_2 g_2(n) \\
\Rightarrow & t_1(n) + t_2(n) \geq c_1 g_1(n) + c_2 g_2(n) \\
\Rightarrow & \quad \langle \text{Posons } c_3 = \min(c_1, c_2) \rangle \\
& t_1(n) + t_2(n) \geq c_3(g_1(n) + g_2(n)) \\
\Rightarrow & \quad \langle \text{pour tout } a, b \geq 0, \text{ on a } a + b \geq \max(a, b). \rangle \\
& t_1(n) + t_2(n) \geq c_3 \max(g_1(n), g_2(n)) \\
\Leftrightarrow & \quad \langle \text{Définition de } \Omega \rangle \\
& t_1(n) + t_2(n) \in \Omega(\max(g_1(n), g_2(n)))
\end{aligned}$$

Question # 4

Démontrez la règle du maximum pour Θ .

Solution :

Résultat 1 :

$$\begin{aligned}
& t_1(n) \in \Theta(g_1(n)) \wedge t_2(n) \in \Theta(g_2(n)) \\
\Rightarrow & \quad \langle \text{car } \Theta(g(n)) = \Omega(g(n)) \cap O(g(n)) \rangle \\
& t_1(n) \in \Omega(g_1(n)) \wedge t_2(n) \in \Omega(g_2(n)) \\
\Rightarrow & \quad \langle \text{en vertu de la règle du maximum pour } \Omega \rangle \\
& t_1(n) + t_2(n) \in \Omega(\max(g_1(n), g_2(n)))
\end{aligned}$$

Résultat 2 :

$$\begin{aligned}
& t_1(n) \in \Theta(g_1(n)) \wedge t_2(n) \in \Theta(g_2(n)) \\
\Rightarrow & \quad \langle \text{car } \Theta(g(n)) = \Omega(g(n)) \cap O(g(n)) \rangle \\
& t_1(n) \in O(g_1(n)) \wedge t_2(n) \in O(g_2(n)) \\
\Rightarrow & \quad \langle \text{en vertu de la règle du maximum pour } O \rangle \\
& t_1(n) + t_2(n) \in O(\max(g_1(n), g_2(n)))
\end{aligned}$$

Ces deux résultats impliquent que :

$$t_1(n) + t_2(n) \in \Omega(\max(g_1(n), g_2(n))) \cap O(\max(g_1(n), g_2(n))) = \Theta(\max(g_1(n), g_2(n)))$$

Question # 5

Démontrez que $f(n) \in \Theta(g(n)) \Leftrightarrow \Theta(f(n)) = \Theta(g(n)) \Leftrightarrow O(f(n)) = O(g(n))$

Solution :

- $f(n) \in \Theta(g(n)) \Rightarrow \Theta(f(n)) = \Theta(g(n)) :$

$$\begin{aligned} & \text{Soit } x(n) \in \Theta(f(n)) \\ \Leftrightarrow & \quad \langle \text{Définition de } \Theta \rangle \\ & c_1 f(n) \leq x(n) \leq c_2 f(n) \\ \Rightarrow & \quad \langle f(n) \in \Theta(g(n)) \rangle \\ & d_1 g(n) \leq c_1 f(n) \leq x(n) \leq c_2 f(n) \leq d_2 g(n) \\ \Leftrightarrow & \quad \langle \text{Définition de } \Theta \rangle \\ & x(n) \in \Theta(g(n)) \end{aligned}$$

De la même façon (et en utilisant la symétrie de Θ), on peut montrer que $x(n) \in \Theta(g(n)) \Rightarrow x(n) \in \Theta(f(n))$ sous l'hypothèse que $f(n) \in \Theta(g(n))$. On obtient alors l'égalité entre $\Theta(f(n))$ et $\Theta(g(n))$

- $\Theta(f(n)) = \Theta(g(n)) \Rightarrow O(f(n)) = O(g(n)) :$

$$\begin{aligned} & \text{Soit } x(n) \in O(f(n)) \\ \Leftrightarrow & \quad \langle \text{Définition de } O \rangle \\ & x(n) \leq c f(n) \\ \Rightarrow & \quad \langle f(n) \in \Theta(g(n)) \text{ car } \Theta(f(n)) = \Theta(g(n)) \rangle \\ & x(n) \leq c f(n) \text{ et } d_1 g(n) \leq f(n) \leq d_2 g(n) \\ \Rightarrow & \\ & x(n) \leq c \times d_2 g(n) \\ \Leftrightarrow & \quad \langle \text{Définition de } O \rangle \\ & x(n) \in O(g(n)) \end{aligned}$$

De la même façon, on peut montrer que $x(n) \in O(g(n)) \Rightarrow x(n) \in O(f(n))$ sous l'hypothèse que $\Theta(f(n)) = \Theta(g(n))$. On obtient alors l'égalité entre $O(f(n))$ et $O(g(n))$

- $O(f(n)) = O(g(n)) \Rightarrow f(n) \in \Theta(g(n)) :$
En assumant $O(f(n)) = O(g(n))$ et par la réflexivité de O , on a que $f(n) \in O(g(n))$ et $g(n) \in O(f(n))$. On déduit donc que $f(n) \leq d_1 g(n)$ et $\frac{1}{d_2} g(n) \leq f(n)$ ce qui donne bien $f(n) \in \Theta(g(n))$.

Question # 6

Démontrez que :

$$\begin{aligned} f(n) \in O(g(n)) \wedge f(n) \notin \Theta(g(n)) &\Leftrightarrow f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n)) \\ &\Leftrightarrow f(n) \in O(g(n)) \wedge g(n) \notin O(f(n)) \\ &\Leftrightarrow O(f(n)) \subset O(g(n)) \end{aligned}$$

Solution :

- $f(n) \in O(g(n)) \wedge f(n) \notin \Theta(g(n)) \Rightarrow f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n)) :$
Assumons $f(n) \in O(g(n)) \wedge f(n) \notin \Theta(g(n))$ et supposons que $f(n) \in \Omega(g(n))$. On a donc que $f(n) \leq cg(n)$ et $f(n) \geq dg(n)$, ce qui donne que $dg(n) \leq f(n) \leq cg(n)$ et donc $f(n) \in \Theta(g(n))$. Ceci est une contradiction, on a donc $f(n) \notin \Omega(g(n))$.
- $f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n)) \Rightarrow f(n) \in O(g(n)) \wedge g(n) \notin O(f(n)) :$
Assumons $f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n))$ et supposons $g(n) \in O(f(n))$. Par la dualité entre O et Ω , on obtient que $f(n) \in \Omega(g(n))$. Ceci est une contradiction, on a donc $g(n) \notin O(f(n))$.
- $f(n) \in O(g(n)) \wedge g(n) \notin O(f(n)) \Rightarrow O(f(n)) \subset O(g(n)) :$
Comme $f(n) \in O(g(n))$, on a que $O(f(n)) \subseteq O(g(n))$ (pour tout $x(n) \in O(f(n))$, on a $x(n) \leq c(f(n)) \leq d(g(n))$ et donc $x(n) \in O(g(n))$). Si $O(f(n)) = O(g(n))$, alors forcément $g(n) \in O(f(n))$ (car $g(n) \in O(g(n))$). Ceci est une contradiction. Puisque $O(f(n)) \subseteq O(g(n))$, mais que $O(f(n)) \neq O(g(n))$, on obtient que $O(f(n)) \subset O(g(n))$.
- $O(f(n)) \subset O(g(n)) \Rightarrow f(n) \in O(g(n)) \wedge f(n) \notin \Theta(g(n)) :$
Comme $O(f(n)) \subset O(g(n))$, il est clair que $f(n) \in O(g(n))$. Par la question 5, on sait que si $f(n) \in \Theta(g(n))$, on obtient que $O(f(n)) = O(g(n))$, ce qui est une contradiction. On doit donc avoir $f(n) \notin \Theta(g(n))$.

Question # 7

Soit deux fonctions positives asymptotiquement $f(n)$ et $g(n)$. Les relations $=$, \neq , \subset et $\not\subset$ sont quatre relations intéressantes pour comparer les ensembles $O(f(n))$ et $O(g(n))$.

- A) Pourquoi n'est-il pas intéressant d'utiliser seulement $=$ ou bien \subset pour comparer les ensembles $\Theta(f(n))$ et $\Theta(g(n))$?

Solution :

Il n'est pas possible d'avoir $\Theta(f(n)) \subset \Theta(g(n))$. On peut facilement montrer que si il existe une fonction x telle que $x \in \Theta(f(n))$ et $x \in \Theta(g(n))$, alors on a $\Theta(f(n)) = \Theta(g(n))$.

- B) Que devrait-on utiliser pour comparer $\Theta(f(n))$ et $\Theta(g(n))$?

Solution :

On devrait utiliser $=$ ou bien \neq .

***Question # 8**

Pour chacune des fonctions suivantes, indiquez la classe $\Theta(g(n))$ à laquelle la fonction appartient (Utilisez la forme la plus simple possible pour $g(n)$). Démontrez vos affirmations.

A) $(n^2 + 1)^{10}$

Solution :

$$(n^2 + 1)^{10} \in \Theta(n^{20})$$

$$\lim_{n \rightarrow \infty} \frac{(n^2 + 1)^{10}}{n^{20}} = \lim_{n \rightarrow \infty} \frac{(n^2 + 1)^{10}}{(n^2)^{10}} = \lim_{n \rightarrow \infty} \left(\frac{n^2 + 1}{n^2}\right)^{10} = \lim_{n \rightarrow \infty} \left(1 + \frac{1}{n^2}\right)^{10} = 1$$

B) $\sqrt{10n^2 + 7n + 3}$

Solution :

$$\sqrt{10n^2 + 7n + 3} \in \Theta(n)$$

$$\lim_{n \rightarrow \infty} \frac{\sqrt{10n^2 + 7n + 3}}{n} = \lim_{n \rightarrow \infty} \sqrt{\frac{10n^2 + 7n + 3}{n^2}} = \lim_{n \rightarrow \infty} \sqrt{10 + \frac{7}{n} + \frac{3}{n^2}} = \sqrt{10}$$

C) $2n \lg(n+2)^2 + (n+2)^2 \lg(\frac{n}{2})$

Solution :

$$2n \lg(n+2)^2 + (n+2)^2 \lg(\frac{n}{2}) \in \Theta(n^2 \lg n)$$

$$\begin{aligned} & 2n \lg(n+2)^2 + (n+2)^2 \lg(\frac{n}{2}) \\ = & \\ & 2n 2 \lg(n+2) + (n+2)^2 (\lg n - 1) \\ \in & \quad \langle \text{par la règle du maximum} \rangle \\ & \Theta((n+2)^2 (\lg n - 1)) \\ = & \\ & \Theta((n^2 + 4n + 4)(\lg n - 1)) \\ = & \\ & \Theta(n^2 \lg n + 4n \lg n + 4 \lg n - n^2 - 4n - 4) \\ = & \\ & \quad \langle \lim_{n \rightarrow \infty} \frac{n^2 \lg n + 4n \lg n + 4 \lg n - n^2 - 4n - 4}{n^2 \lg n} = 1, \text{ voir plus bas.} \rangle \\ & \Theta(n^2 \lg n) \\ \\ & \lim_{n \rightarrow \infty} \frac{n^2 \lg n + 4n \lg n + 4 \lg n - n^2 - 4n - 4}{n^2 \lg n} \\ = & \\ & \lim_{n \rightarrow \infty} \frac{n^2 \lg n}{n^2 \lg n} + \frac{4n \lg n}{n^2 \lg n} + \frac{4 \lg n}{n^2 \lg n} - \frac{n^2}{n^2 \lg n} - \frac{4n}{n^2 \lg n} - \frac{4}{n^2 \lg n} \\ = & \\ & \lim_{n \rightarrow \infty} 1 + \frac{4}{n} + \frac{4}{n^2} - \frac{1}{\lg n} - \frac{4}{n \lg n} - \frac{4}{n^2 \lg n} \\ = & \\ & 1 + \frac{4}{\infty} + \frac{4}{\infty} - \frac{1}{\infty} - \frac{4}{\infty} - \frac{4}{\infty} \\ = & \\ & 1 + 0 + 0 - 0 - 0 - 0 = 1 \end{aligned}$$

$$D) 2^{n+1} + 3^{n-1}$$

Solution :

$$2^{n+1} + 3^{n-1} \in \Theta(3^n)$$

$$2^{n+1} + 3^{n-1}$$

=

$$2^n 2 + 3^n \frac{1}{3}$$

$$\in \quad \langle \text{par la règle du maximum} \rangle$$

$$\Theta(3^n)$$

E) $\lfloor \lg n \rfloor$

Solution :

Puisque $x - 1 < \lfloor x \rfloor \leq x$, on obtient que :

$$\begin{aligned}\lfloor \lg n \rfloor &\leq \lg n = \lg(a) \log_a(n) \Rightarrow \lfloor \lg n \rfloor \in O(\log n) \\ \lfloor \lg n \rfloor &> \lg n - 1 \\ &= \lg(a) \log_a(n) - 1 \\ &\geq \lg(a) \log_a(n) - \frac{1}{2} \lg(a) \log_a(n) && \forall n \geq 2 \\ &= \frac{1}{2} \lg(a) \log_a(n) \\ &\Rightarrow \lfloor \lg n \rfloor \in \Omega(\log n)\end{aligned}$$

Alors $\lfloor \lg n \rfloor \in \Theta(\log n)$.

*Question # 9

Placez les fonctions suivantes en ordre croissant de leur ordre de croissance.

$$(n-2)!, 5 \lg(n+100)^{10}, 2^{2n}, 0.001n^4 + 3n3 + 1, (\ln n)^2, \sqrt[3]{n}, 3^n$$

Solution :

$$5 \lg(n+100)^{10}, (\ln n)^2, \sqrt[3]{n}, 0.001n^4 + 3n3 + 1, 3^n, 2^{2n}, (n-2)!$$

Question # 10

Les valeurs contenus dans le tableau 2.1 (voir A. Levitin) suggère que les fonctions suivantes sont placées en ordre croissant de leur ordre de croissance :

$$\lg n, n, n \lg n, n^2, n^3, 2^n, n!$$

A) Est-ce que les valeurs du tableau forme une preuve que les fonctions sont placées en ordre croissant de leur ordre de croissance ?

Solution :

L'ordre de croissance d'une fonction est étroitement lié au comportement asymptotique de cette fonction (lorsque n tends vers l'infini). Donc, en aucun cas, un ensemble fini de valeurs d'une fonction peut constituer une preuve de l'ordre de croissance de cette fonction.

B) Démontrez que les fonctions ci-haut sont listées en ordre croissant de leur ordre de croissance.

Solution :

$$\lim_{n \rightarrow \infty} \frac{\lg n}{n} = \lim_{n \rightarrow \infty} \frac{1}{n} \lg e = \lim_{n \rightarrow \infty} \frac{c}{n} = 0$$

$$\lim_{n \rightarrow \infty} \frac{n}{n \lg n} = \lim_{n \rightarrow \infty} \frac{1}{\lg n} = 0$$

$$\lim_{n \rightarrow \infty} \frac{n \lg n}{n^2} = \lim_{n \rightarrow \infty} \frac{\lg n}{n} = 0$$

Déjà démontré ci-dessus

$$\lim_{n \rightarrow \infty} \frac{n^2}{n^3} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0$$

$$\lim_{n \rightarrow \infty} \frac{n^3}{2^n} = \lim_{n \rightarrow \infty} \frac{3n^2}{2^n \ln 2} = \lim_{n \rightarrow \infty} \frac{6n}{2^n (\ln 2)^2}$$

Règle de L'Hôpital

Règle de L'Hôpital

$$= \lim_{n \rightarrow \infty} \frac{6}{2^n (\ln 2)^3} = 0$$

Règle de L'Hôpital

$$\lim_{n \rightarrow \infty} \frac{2^n}{n!} = 0$$

Voir la solution de #3B série 0

Question # 11

Démontrez que tout polynôme $p(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_0$ tel que $a_k > 0$ appartient à $\Theta(n^k)$.

Solution :

$$\lim_{n \rightarrow \infty} \frac{a_k n^k + a_{k-1} n^{k-1} + \dots + a_0}{n^k} = \lim_{n \rightarrow \infty} a_k + \frac{a_{k-1}}{n} + \dots + \frac{a_0}{n^k} = a_k > 0$$

Question # 12

Démontrez que deux fonctions exponentielle a^n et b^n ayant des bases différentes ($a \neq b$) ont des ordre de croissance différente.

Solution :

$$\lim_{n \rightarrow \infty} \frac{a^n}{b^n} = \lim_{n \rightarrow \infty} \left(\frac{a}{b}\right)^n = \begin{cases} 0 & \text{si } a < b \\ \infty & \text{si } a > b \end{cases}$$

Question # 13

Démontrez que, pour tout $a \geq 2$ et $b \geq 2$, on a : $\log_b(n) \in \Theta(\log_a(n))$ (et alors $\log_a(n) \in \Theta(\log_b(n))$).

Solution :

$$\lim_{n \rightarrow \infty} \frac{\log_a(n)}{\log_b(n)} = \lim_{n \rightarrow \infty} \frac{\log_a(b) \log_b(n)}{\log_b(n)} = \log_a(b)$$

Question # 14

1. Elvis, Jimmy et Janis analysent un même algorithme A et obtiennent les résultats $T_E(n)$, $T_i(n)$ et $T_a(n)$ respectivement. Elvis obtient $T_E(n) \in O(n^2)$ en pire cas. Jimmy obtient $T_i(n) \in \Omega(n^2)$ en meilleur cas. Janis affirme que $T_a(n) \in \theta(n^2)$ pour tous les cas. Si Elvis et Jimmy ont raison, le résultat de Janis est-il correct ? Justifiez.

2. Soient **A** et **B** des algorithmes prenant des temps, en meilleur cas, dans $\Omega(n^2)$ et $\Omega(n \log n)$ respectivement. Est-il possible qu'une implantation de l'algorithme **A** soit plus efficace qu'une implantation de l'algorithme **B** sur tous les exemplaires ? Justifiez.
3. Alice et Bob analysent un même algorithme **A** et obtiennent les résultats $T_A(n)$ et $T_B(n)$ respectivement. Alice obtient que $T_A(n) \in \theta(n^3)$ en pire cas. Bob obtient que $T_B(n) \in \Omega(n^2 \log n^2)$ pour tous les cas. Le résultat d'Alice implique-t-il celui de Bob ? Justifiez.
4. Énoncez le principe d'invariance en algorithmique.

Solutions :

1. **Oui.** Elvis $\Rightarrow T(n) \in O(n^2)$ dans tous les cas et Jimmy $\Rightarrow T(n) \in \Omega(n^2)$ dans tous les cas, d'où le résultat.
2. **Oui.** Exemple : $T_A(n) = n^2$ et $T_B(n) = 5n^3 + 5$
3. **Non.** On pourrait avoir, par exemple, $T(n) = n^2$ (en meilleur cas).
4. Les **temps d'exécution de deux implantations d'un même algorithme** ne diffèrent qu'à une **constante multiplicative** près (voir notes de cours chapitre 2).

***Question # 15**

Soient **A**, **B**, **C**, **D** et **E** cinq algorithmes qui résolvent le même problème **P** et pour lesquels nous avons obtenu les résultats d'analyse suivants respectivement :

1. $T_A(n) \in O(n^2)$ en pire cas ;
 2. $T_B(n) \in \Omega(n^2)$ dans tous les cas ;
 3. $T_C(n) \in \Theta(n)$ en meilleur cas ;
 4. $T_D(n) \in \Theta(n^2 \lg n)$ dans tous les cas ;
 5. $T_E(n) \in \Theta(n^2)$ en pire cas.
- a) Vous êtes à concevoir un algorithme **Z** qui résout un problème **P'** ayant des instances de taille n . Votre algorithme **Z** ne doit jamais prendre plus de $n^4 \lg n$ unités de temps (à une constante près et pour des entrées de taille assez grande). Dans **Z**, il y a $n^2 \lg n$ appels à Résoudre le problème **P** (tous sur des instances de taille n). Donnez, parmi les cinq algorithmes ci-haut, tous ceux qui peuvent être utilisés dans votre algorithme **Z**, sans dépasser la limite de temps.
 - b) Vous êtes à élaborer un système qui doit fonctionner en temps réel (le facteur temps est primordial). Pour ce faire, vous devez choisir un (et un seul) algorithme parmi les 5 ci-haut. Quel est, selon vous, le choix le plus prometteur ? Justifiez.
 - c) Pour chacun des 5 algorithmes ci-haut, dites, avec justifications, si on est assuré que son temps d'exécution **en pire cas** appartient à l'ensemble suivant :

$$S = (O(n^3) \cap \Omega(n^2)) \cup \Theta(n).$$

- d) En supposant que les analyses 2 et 3 soient exactes, est-il possible que les algorithmes **B** et **C** soient en fait un seul et même algorithme ? Justifiez.

Solutions :

- a) Les algorithmes **A** et **E** peuvent être utilisés.

- b) L'analyse des algorithmes B et C ne donne aucune informations sur le maximum de temps requis pour ces algorithmes. Ils peuvent donc possiblement être extrêmement lent. Clairement, le choix de l'algorithme D est moins bon que A et E . Puisque E nous assure un temps de n^2 en pire cas, alors que A nous assure un temps inférieur ou égal à n^2 en pire cas, l'algorithme E ne peut être meilleur que A . A est donc le choix le plus prometteur.
- c) A : non, le temps en PC est au plus n^2 mais pourrait être inférieur, $n \lg n$ par exemple et dans ce cas il n'est pas dans l'ensemble car $n \lg n \notin \Omega(n^2)$ et $n \lg n \notin \Theta(n)$.
 B : non, aucune borne supérieure sur le temps en PC, ce dernier peut donc être n^{10} qui n'appartient clairement pas à S .
 C : non, même explication que B .
 D : oui, comme $T_D(n) \in \Theta(n^2 \lg n)$, on sait que $T_D(n) \in \Omega(n^2 \lg n) \subset \Omega(n^2)$ et $T_D(n) \in O(n^2 \lg n) \subset O(n^3)$.
 E : oui, comme $T_E(n) \in \Theta(n^2)$, on sait que $T_E(n) \in \Omega(n^2)$ et $T_E(n) \in O(n^2) \subset O(n^3)$.
- d) Non car il y a une contradiction. L'analyse 2 nous dit que $T(n)$ est borné inférieurement par n^2 dans tous les cas (en particulier pour le meilleur cas). Par contre, l'analyse 3 nous dit que $T(n)$ est exactement n en meilleur cas. Mais comme n ne peut être plus petit que n^2 , il ne peut donc pas s'agir du même algorithme.

*Question # 16

Déterminez si les énoncés suivants sont vrais ou faux. Justifiez en mots (ou en montrant les étapes de votre démarche si nécessaire).

A) $376n + 98 \in O(n)$

Solution :

Vrai. $O(n)$ est l'ensemble des fonctions dont l'ordre de croissance est plus petit ou égal à n .

Par la règle du maximum on laisse tomber la constante 98.

De plus, la constante 376 n'affecte pas l'ordre de grandeur de la fonction.

Donc, $376n + 98 \in O(n)$.

B) $2n^2 + n \in \Omega(n)$

Solution :

Vrai. $\Omega(n)$ est l'ensemble des fonctions dont l'ordre de croissance est plus grand ou égal à n .

En utilisant la limite du ratio $\frac{2n^2+n}{n}$ lorsque n tend vers l'infini pour comparer $2n^2 + n$ et n , nous obtenons ∞ .

De ce fait, $2n^2 + n \in \Omega(n)$.

C) $\ln(n^n) + \log_2 n \in \Theta(\ln n)$

Solution :

Faux. $\Theta(\ln n)$ est l'ensemble des fonctions dont l'ordre de croissance est $\ln n$.

Par la règle du maximum on laisse tomber la fonction $\log_2 n$.

Or, par l'arithmétique des logs nous avons que $\ln(n^n) = n \ln(n)$.

Donc, $\ln(n^n) + n \notin \Theta(\ln n)$.

*Question # 17

Soit l'algorithme suivant.

```

Algorithm Secret( $A[0 \dots n - 1]$ ) :
    //Entré : Un tableau  $A[0 \dots n - 1]$  de  $n$  nombres réels
     $minval \leftarrow A[0]$ 
     $maxval \leftarrow A[0]$ 
    FOR  $i \leftarrow 1$  TO  $n - 1$ 
        IF  $A[i] < minval$ 
             $minval \leftarrow A[i]$ 
        IF  $A[i] > maxval$ 
             $maxval \leftarrow A[i]$ 
    RETURN  $maxval - minval$ 

```

A) Que fait cet algorithme ?

Solution :

Il calcule l'écart maximal entre deux éléments du tableau. C'est donc l'écart entre le plus grand élément et le plus petit.

B) Quelle est son opération de base ?

Solution :

La comparaison d'un élément.

C) Combien de fois l'opération de base est-elle exécutée ?

Solution :

$$C(n) = \sum_{i=1}^{n-1} 2 = 2(n - 1)$$

D) À quelle classe d'efficacité appartient cet algorithme ?

Solution :

Linéaire : $\Theta(n)$

*Question # 18

Soit l'algorithme suivant.

```

Algorithm inefficace( $A[0 \dots n - 1]$ ) :
    //Entrée : Un tableau  $A[0 \dots n - 1]$  de  $n$  nombres entiers
    FOR  $i \leftarrow 0$  TO  $n - 1$ 
        Effacer( $A, 0$ )
    RETURN  $A$ 

```

Supposons que la fonction **Effacer** est l'équivalent de la méthode **erase** de la classe **vector** en C++. Pour répondre à cette question, il est conseillé de consulter la documentation de cette méthode.

A) Que fait cet algorithme ?

Solution :

Il efface tous les éléments du vecteur A , à partir du début. Chaque fois qu'un élément est effacé, tous les éléments qui le suivent sont recopiés.

B) Quel est le temps d'exécution d'un appel à **Effacer** ?

Solution :

Soit $C_{\text{effacer}}(m)$, le temps d'exécution de la fonction **Effacer**, où m est le nombre d'éléments dans le vecteur lors de l'appel à la fonction. Selon la documentation de la STL, cette fonction est linéaire en fonction du nombre d'éléments effacés plus le nombre d'éléments déplacés. L'algorithme supprime un seul élément à la fois et c'est toujours le premier élément du vecteur. Il y a donc une suppression et $m - 1$ éléments déplacés. On a donc $C_{\text{effacer}}(m) = 1 + m - 1 = m$.

C) Quel est le temps d'exécution de l'algorithme inefficace ?

Solution :

Soit $C_{\text{inefficace}}(n)$, le temps d'exécution de l'algorithme inefficace. On a

$$\begin{aligned}
 & C_{\text{inefficace}}(n) \\
 = & \quad \langle \text{À chaque itération, le vecteur contient un élément de moins} \rangle \\
 & \sum_{i=0}^{n-1} C_{\text{effacer}}(n - i) \\
 = & \quad \langle C_{\text{effacer}}(m) = m \rangle \\
 & \sum_{i=0}^{n-1} (n - i) \\
 = & \quad \langle \sum_{i=0}^{n-1} (n - i) = (n - 0) + (n - 1) + \dots + (n - n + 1) = \\
 & \quad 1 + 2 + \dots + n = \sum_{i=1}^n i \rangle \\
 & \sum_{i=1}^n i \\
 = & \quad \langle \text{Aide-mémoire} \rangle \\
 & \frac{n(n+1)}{2} = \frac{n^2+n}{2}
 \end{aligned}$$

D) À quelle classe d'efficacité appartient cet algorithme ?

Solution :

$C_{\text{inefficace}}(n) = \frac{n^2+n}{2} \in \Theta(n^2)$ par la règle du maximum. En effet, puisque $\frac{n^2}{2} \in \Theta(n^2)$ et $\frac{n}{2} \in \Theta(n)$, nous avons $\frac{n^2+n}{2} \in \Theta(\max(n^2, n)) = \Theta(n^2)$. L'algorithme appartient donc à la classe d'efficacité quadratique.

*Question # 19

Soit l'algorithme suivant.

```

Algorithm Enigma(A[0...n - 1, 0...n - 1]) :
  //Entré : Une matrice A[0...n - 1, 0..n - 1] de nombres réels
  FOR i ← 0 TO n - 2
    FOR j ← i + 1 TO n - 1
      IF A[i, j] ≠ A[j, i]
        RETURN false
  RETURN true

```

A) Que fait cet algorithme ?

Solution :

Il retourne *true* si la matrice d'entrée est symétrique, et *false* sinon.

B) Quelle est son opération de base ?

Solution :

La comparaison de deux éléments de la matrice.

C) Combien de fois l'opération de base est-elle exécutée en pire et en meilleur cas ?

Solution :

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 = \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) = \sum_{i=0}^{n-2} (n-i-1) = \sum_{i=1}^{n-1} i = \frac{(n-1)n}{2}$$

$$C_{best}(n) = 1$$

D) À quelle classe d'efficacité appartient cet algorithme ?

Solution :

Quadratique en pire cas : $C_{worst}(n) \in \Theta(n^2)$ (ou $C(n) \in O(n^2)$)

Constant en meilleur cas : $C_{best}(n) \in \Theta(1)$

*Question # 20

Analysez la complexité de l'algorithme suivant :

```

Algorithme Mystérieux(A[0...n - 1]) :
    /* Input : A[0...n - 1], un vecteur d'entiers positifs. */
    TriFusion(A)      /* S'exécute en  $\Theta(n \log n)$  dans tous les cas. */
    val  $\leftarrow \infty$ 
    i  $\leftarrow 0$ 
    WHILE i  $\leq$  n - 2
        IF A[i] = A[i + 1]
            RETURN 0
        IF |A[i] - A[i + 1]| < val
            val  $\leftarrow$  |A[i] - A[i + 1]|
            i  $\leftarrow$  i + 1
    RETURN val
    
```

Effectuez toutes les étapes pour l'analyse :

- Choix d'une opération de base.
- Calcul du nombre de fois où l'opération de base est exécutée.
- Poser la classe de complexité.

Utilisez la notation Θ . Si le temps d'exécution peut varier entre deux instances de même taille alors il faut procéder à l'analyse en meilleur cas et en pire cas. **Solution :**

L'algorithme *Mystérieux* :

- La complexité l'algorithme *TriFusion* est de $\Theta(n \log n)$ dans tous les cas.
- L'opération de base est IF |A[i] - A[i + 1]| < val.
- Pire cas :

En pire cas, il n'y a aucuns doublons et on doit donc parcourir le tableau au complet. La sommation à poser pour calculer le nombre $C_{worst}(n)$ de fois où celle-ci est exécutée est

$$C_{worst}(n) = \sum_{i=0}^{n-2} 1 \tag{7}$$

Après simplification de la sommation nous avons une complexité de $\Theta(n \log n + n)$ (où $n \log n$ provient de l'appel à *TriFusion*). Par l'utilisation de la règle du maximum, nous obtenons que *Mystérieux* est en $\Theta(n \log n)$ en pire cas.

— Meilleur cas :

Dans le meilleur cas, l'opération de base n n'est exécutée qu'une seule fois lorsque les deux plus petits éléments du tableau (qui se retrouvent au début du tableau une fois trié) sont égaux. Nous avons donc une complexité de $\Theta(n \log n + 1)$ (où $n \log n$ provient de l'appel à *TriFusion*). Par la règle du maximum, *Mystérieux* est en $\Theta(n \log n)$ en meilleur cas.

***Question # 21**

En utilisant l'approximation par intégrale, déterminer l'ordre exacte de croissance pour les fonctions suivantes :

A) $\sum_{i=0}^{n-1} (i^2 + 1)^2$

Solution :

Soit $S = \sum_{i=0}^{n-1} (i^2 + 1)^2$. Nous avons :

$$\begin{aligned} S &\leq \int_0^n (x^4 + 2x^2 + 1) dx \\ &\leq \left[\frac{x^5}{5} + \frac{2x^3}{3} + x \right]_0^n \\ &\leq \frac{n^5}{5} + \frac{2n^3}{3} + n \\ &\in O(n^5) \text{ D'après la règle du maximum} \end{aligned}$$

Le retrait du terme pour $i = 0$ ci-dessous permet à la fonction qui est intégrée d'être croissante entre 0 et $n - 1$. Elle ne le serait pas entre -1 et $n - 1$.

$$\begin{aligned} S &\geq \sum_{i=1}^{n-1} (i^2 + 1)^2 \geq \int_0^{n-1} (x^4 + 2x^2 + 1) dx \\ &\geq \left[\frac{x^5}{5} + \frac{2x^3}{3} + x \right]_0^{n-1} \\ &\geq \frac{(n-1)^5}{5} + \frac{2(n-1)^3}{3} + (n-1) \end{aligned}$$

Lorsque $n \geq 2$, nous avons $\frac{2(n-1)^3}{3} \geq 0$ et $(n-1) \geq 0$. De plus, nous avons $n-1 \geq n - \frac{n}{2}$.

$$\begin{aligned} S &\geq \frac{(n - \frac{n}{2})^5}{5} + 0 + 0 \\ &\geq \frac{1}{2^5 \cdot 5} n^5 \\ &\in \Omega(n^5) \end{aligned}$$

$$\text{Donc } S = \sum_{i=0}^{n-1} (i^2 + 1)^2 \in \Theta(n^5)$$

B) $\sum_{i=2}^{n-1} \log_2 i^2$

Solution :

$$\begin{aligned} \sum_{i=2}^{n-1} \log_2 i^2 &= 2 \sum_{i=2}^{n-1} \log_2 i \\ \Rightarrow \quad &\langle \text{Approximation par intégrale} \rangle \\ 2 \int_1^{n-1} \log_2 x dx &\leq 2 \sum_{i=2}^{n-1} \log_2 i \leq 2 \int_2^n \log_2 x dx \\ \Rightarrow \quad & \\ 2 \left(\frac{x \ln(x)}{\ln(2)} - \frac{x}{\ln(2)} \right) \Big|_1^{n-1} &\leq 2 \sum_{i=2}^{n-1} \log_2 i \leq 2 \left(\frac{x \ln(x)}{\ln(2)} - \frac{x}{\ln(2)} \right) \Big|_2^n \\ \Rightarrow \quad & \end{aligned}$$

$$\begin{aligned}
& 2\left(\frac{\ln(n-1)n - \ln(n-1) - n + 2}{\ln(2)}\right) \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq 2\left(\frac{n \ln(n) - n + 2 - 2 \ln(2)}{\ln(2)}\right) \\
\Rightarrow & 2\left(\frac{\ln(\frac{n}{2})n - \frac{1}{4}n \ln(n) - \frac{1}{4}n \ln(n)}{\ln(2)}\right) \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq \frac{2}{\ln(2)}n \ln(n) \text{ pour } n \geq 4 \\
\Rightarrow & \langle \text{Puisque } \ln(\frac{n}{2}) = \ln(n) - \ln(2) \rangle \\
& 2\left(\frac{(\ln(n) - \ln(2))n - \frac{1}{4}n \ln(n) - \frac{1}{4}n \ln(n)}{\ln(2)}\right) \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq \frac{2}{\ln(2)}n \ln(n) \\
\Rightarrow & \frac{1}{\ln(2)}n \ln(n) - 2n \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq \frac{2}{\ln(2)}n \ln(n) \\
\Rightarrow & \langle \text{En multipliant } 2n \text{ par } \frac{n}{4 \ln(2)} \text{ pour obtenir deux termes en } n \ln n \rangle \\
& \frac{1}{\ln(2)}n \ln(n) - \frac{1}{2 \ln(2)}n \ln(n) \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq \frac{2}{\ln(2)}n \ln(n) \text{ pour } n \geq 4 \ln(2) \\
\Rightarrow & \frac{1}{2 \ln(2)}n \ln(n) \leq 2 \sum_{i=2}^{n-1} \log_2 i \leq \frac{2}{\ln(2)}n \ln(n) \\
\Rightarrow & \sum_{i=2}^{n-1} \log_2 i^2 \in \Theta(n \ln(n))
\end{aligned}$$

C) $\sum_{i=0}^{n-1} \sum_{j=0}^{i-1} (i+j)$

Solution :

Posons :

$$S = \sum_{i=0}^{n-1} \sum_{j=0}^{i-1} (i+j)$$

$$\begin{aligned}
\int_{-1}^{n-1} \int_{-1}^{x-1} (x+y) dy dx &\leq S \leq \int_0^n \int_0^x (x+y) dy dx \\
\int_{-1}^{n-1} \left[xy + \frac{1}{2}y^2 \right]_{-1}^{x-1} dx &\leq S \leq \int_0^n \left[xy + \frac{1}{2}y^2 \right]_0^x dx \\
\int_{-1}^{n-1} \left(x(x-1) + \frac{1}{2}(x-1)^2 + x - \frac{1}{2} \right) dx &\leq S \leq \int_0^n (x^2 + \frac{1}{2}x^2 - 0 - 0) dx \\
\int_{-1}^{n-1} \left(\frac{3}{2}x^2 - x \right) dx &\leq S \leq \int_0^n \frac{3}{2}x^2 dx \\
\left[\frac{1}{2}x^3 - \frac{1}{2}x^2 \right]_{-1}^{n-1} &\leq S \leq \left[\frac{1}{2}x^3 \right]_0^n \\
\frac{1}{2} [(n-1)^3 - (n-1)^2 + 1 + 1] &\leq S \leq \frac{1}{2}n^3 \\
\frac{1}{2}n^3 - 2n^2 + \frac{5}{2}n &\leq S \leq \frac{1}{2}n^3 \\
\frac{1}{2}n^3 - 2n^2 \cdot \frac{1}{8}n + 0 &\leq S \leq \frac{1}{2}n^3 \\
\frac{1}{4}n^3 &\leq S \leq \frac{1}{2}n^3 \\
S &\in \Theta(n^3)
\end{aligned}$$

Pour $n \geq 8$

***Question # 22**

Analysez la complexité de l'algorithme suivant en fonction de n :

Algorithme Complexe(n)

pour $i = 2..n$
 $c = 0$
while $c < n$
 $c = c + i$

Effectuez toutes les étapes pour l'analyse :

- Choix d'une opération de base.
- Calcul du nombre de fois où l'opération de base est exécutée.
- Poser la classe de complexité.

Utilisez la notation Θ . Si le temps d'exécution peut varier entre deux instances de même taille alors il faut procéder à l'analyse en meilleur cas et en pire cas. **Solution :**

L'opération de base est $c = c + i$. Il faut incrémenter $\lceil \frac{n}{i} \rceil$ fois c de i avant de sortir du while. Ainsi, dans tous les cas cette opération de base est exécutée

$$C(n) = \sum_{i=2}^n \left\lceil \frac{n}{i} \right\rceil \quad (8)$$

fois. Pour résoudre la sommation (8), il faut utiliser l'approximation de la somme par des intégrales définies (voir l'aide-mémoire). Notons tout d'abord que $C(n)$ est non croissante. Ainsi, nous avons la

borne inférieure suivante :

$$\sum_{i=2}^n \left\lceil \frac{n}{i} \right\rceil \geq \sum_{i=2}^n \frac{n}{i} \quad (9)$$

$$\geq \int_2^{n+1} \frac{n}{x} dx \quad (10)$$

$$= \left[n \ln x \right]_2^{n+1} \quad (11)$$

$$= n \ln(n+1) - n \ln 2 \quad (12)$$

$$\geq n \ln n - n \ln 2 \quad (13)$$

$$\geq n \ln n - \frac{1}{2} n \ln n \quad (\forall n \geq 4) \quad (14)$$

$$= \frac{1}{2} n \ln n \quad (15)$$

$$\in \Omega(n \ln n) \quad (16)$$

Posons maintenant la borne supérieure :

$$\sum_{i=2}^n \left\lceil \frac{n}{i} \right\rceil \leq \sum_{i=2}^n \left(\frac{n}{i} + 1 \right) \quad (17)$$

$$\leq \int_1^n \left(\frac{n}{x} + 1 \right) dx \quad (18)$$

$$= \left[n \ln x + x \right]_1^n \quad (19)$$

$$= n \ln n + n - 1 \quad (20)$$

$$\leq n \ln n + n \quad (21)$$

$$\in O(n \ln n) \quad // \text{Loi du maximum} \quad (22)$$

D'où $C(n) \in \Theta(n \ln n)$.

***Question # 23**

Résolvez les relations de récurrences suivantes :

A) $x(n) = x(n-1) + 5, x(1) = 0$

Solution :

$$\begin{aligned}x(n) &= x(n-1) + 5 \\&= [x(n-2) + 5] + 5 \\&= [x(n-3) + 5] + 5 \times 2 \\&= \dots \\&= x(n-i) + 5 \times i \\&= \dots \\&= x(1) + 5 \times (n-1) \\&= 5(n-1)\end{aligned}$$

B) $x(n) = 3x(n-1), x(1) = 4$

Solution :

$$\begin{aligned}x(n) &= 3x(n-1) \\&= 3[3x(n-2)] \\&= 3^2[3x(n-3)] \\&= \dots \\&= 3^i x(n-i) \\&= \dots \\&= 3^{n-1} x(1) \\&= 4 \times 3^{n-1}\end{aligned}$$

C) $x(n) = x(n-1) + n, x(0) = 0$

Solution :

$$\begin{aligned}x(n) &= x(n-1) + n \\&= [x(n-2) + (n-1)] + n \\&= [x(n-3) + (n-2)] + (n-1) + n \\&= \dots \\&= x(n-i) + (n-i+1) + (n-i+2) + \dots + n \\&= \dots \\&= x(0) + 1 + 2 + \dots + n \\&= \sum_{i=1}^n i \\&= \frac{n(n+1)}{2}\end{aligned}$$

D) $x(n) = x(n/2) + n, x(1) = 1$ (on suppose $n = 2^k$)

Solution :

$$\begin{aligned}
 x(2^k) &= x(2^{k-1}) + 2^k \\
 &= [x(2^{k-2}) + 2^{k-1}] + 2^k \\
 &= [x(2^{k-3}) + 2^{k-2}] + 2^{k-1} + 2^k \\
 &= \dots \\
 &= x(2^{k-i}) + 2^{k-i+1} + 2^{k-i+2} + \dots + 2^k \\
 &= \dots \\
 &= x(2^{k-k}) + 2^1 + 2^2 + \dots + 2^k \\
 &= 1 + 2^1 + 2^2 + \dots + 2^k \\
 &= \sum_{i=0}^k 2^i \\
 &= 2^{k+1} - 1 \\
 &= 2 \cdot 2^k - 1 \\
 &= 2n - 1
 \end{aligned}$$

E) $x(n) = x(n/3) + 1, x(1) = 1$ (on suppose $n = 3^k$)

Solution :

$$\begin{aligned}
 x(3^k) &= x(3^{k-1}) + 1 \\
 &= [x(3^{k-2}) + 1] + 1 \\
 &= [x(3^{k-3}) + 1] + 1 + 1 \\
 &= \dots \\
 &= x(3^{k-i}) + i \\
 &= \dots \\
 &= x(3^{k-k}) + k \\
 &= x(1) + k \\
 &= 1 + \log_3 n
 \end{aligned}$$

***Question # 24**

Soit l'algorithme suivant.

```
Algorithm Min1( $A[0 \dots n - 1]$ ) :  
  //Entrée : Un tableau  $A[0 \dots n - 1]$  de nombres réels  
  IF  $n = 1$   
    RETURN  $A[0]$   
  ELSE  
     $temp \leftarrow \text{Min1}(A[0 \dots n - 2])$   
    IF  $temp \leq A[n - 1]$   
      RETURN  $temp$   
    ELSE  
      RETURN  $A[n - 1]$ 
```

A) Que fait cet algorithme ?

Solution :

Il trouve la valeur du plus petit élément du tableau.

B) Écrivez la relation de récurrence qui exprime le nombre de fois où l'opération de base est exécutée et résolvez-la.

Solution :

La récurrence pour le nombre de comparaison entre deux éléments est $C(n) = C(n - 1) + 1$, $C(1) = 0$. En résolvant cette récurrence, on obtient $C(n) = n - 1$.

***Question # 25**

Soit l'algorithme suivant qui résout le même problème que l'algorithme de la question 24.

```
Algorithm Min2( $A[l \dots r]$ ) :  
  IF  $l = r$   
    RETURN  $A[l]$   
  ELSE  
     $temp1 \leftarrow \text{Min2}(A[l \dots \lfloor (l + r)/2 \rfloor])$   
     $temp2 \leftarrow \text{Min2}(A[\lfloor (l + r)/2 \rfloor + 1 \dots r])$   
    IF  $temp1 \leq temp2$   
      RETURN  $temp1$   
    ELSE  
      RETURN  $temp2$ 
```

A) Écrivez la relation de récurrence qui exprime le nombre de fois où l'opération de base est exécutée et résolvez-la.

Solution :

L'algorithme appelle récursivement *Min2* sur un vecteur de taille $n_1 = \lfloor (l + r)/2 \rfloor - l + 1$ et un

vecteur de taille $n_2 = r - (\lfloor (l + r)/2 \rfloor + 1) + 1$. Pour exprimer ces expressions en fonction de n , nous supposons que nous avons un vecteur $A[1..n]$ en entrée. Nous commençons par ajuster l'expression de n_1 :

$$\begin{aligned} n_1 &= \lfloor (l + r)/2 \rfloor - l + 1 \\ &= \lfloor (1 + n)/2 \rfloor - 1 + 1 \\ &= \lfloor (1 + n)/2 \rfloor \\ &= \lceil n/2 \rceil \end{aligned}$$

Puisque nous avons un vecteur de $n = n_1 + n_2$ éléments, nous ajustons la seconde expression :

$$n_2 = n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$$

Pour chaque appel récursif, nous effectuons une fois l'opération de base $temp1 \leq temp2$. Nous pouvons maintenant poser la récurrence en fonction du nombre d'éléments n :

$$C(n) = C(\lfloor n/2 \rfloor) + C(\lceil n/2 \rceil) + 1, \quad C(1) = 0$$

Pour résoudre la récurrence, nous commençons par supposer que $n = 2^k$.

$$\begin{aligned} C(2^k) &= C(\lfloor 2^k/2 \rfloor) + C(\lceil 2^k/2 \rceil) + 1 \\ &= C(2^{k-1}) + C(2^{k-1}) + 1 \\ &= 2C(2^{k-1}) + 1 \\ &= 2(2C(2^{k-2}) + 1) + 1 \\ &= 2^2C(2^{k-2}) + 2 + 1 \\ &= 2^2(2C(2^{k-3}) + 1) + 2 + 1 \\ &= 2^3C(2^{k-3}) + 2^2 + 2 + 1 \\ &= \dots \\ &= 2^i C(2^{k-i}) + \sum_{j=0}^{i-1} 2^j \end{aligned}$$

Pour $i = k$

$$\begin{aligned} &= 2^k C(2^{k-k}) + \sum_{j=0}^{k-1} 2^j \\ &= 2^k C(1) + 2^k - 1 \\ &= 2^k - 1 \\ &= n - 1 \in \Theta(n) \end{aligned}$$

Puisque $C(2^k) = n - 1$ est éventuellement non-décroissante, que $f(n) = n$ est harmonieuse, et que $C(n) \in \Theta(n)$ pour $n = 2^k$ avec $k \in \mathbb{N}$, alors $C(n) \in \Theta(n)$ pour tout $n \in \mathbb{N}$ par la règle de l'harmonie.

- B) Lequel des algorithmes *Min1* ou *Min2* est le plus rapide ? Pouvez-vous contruire un nouvel algorithme qui serait plus efficace que *Min1* et *Min2* tout en résolvant le même problème ?

Solution :

Les deux algorithmes *Min1* et *Min2* ont la même efficacité. De plus, il est clair que tout algorithme voulant trouver l'élément minimal d'un tableau quelconque doit faire au moins n comparaisons, donc $\Omega(n)$ comparaisons. Cependant, un algorithme séquentiel n'aurait pas le "overhead" des appels récursifs.

***Question # 26**

Soit A la matrice $n \times n$ suivante.

$$\begin{bmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \dots & a_{nn} \end{bmatrix}$$

On dénote par $\det A$ le déterminant de la matrice A . Pour $n = 1$, $\det A = a_{11}$ et pour $n > 1$, $\det A = \sum_{j=1}^n s_j a_{1j} \det A_j$ où s_j est $+1$ lorsque j est impair et -1 lorsque j est pair, a_{1j} est l'élément de la matrice en ligne 1 et colonne j , et A_j est la matrice $(n-1) \times (n-1)$ obtenue de la matrice A en enlevant la ligne 1 et la colonne j de cette dernière.

- A) Écrivez la relation de récurrence décrivant le nombre de multiplications faite par l'algorithme implementant cette définition récursive pour le calcul du déterminant.

Solution :

Soit $M(n)$, le nombre de multiplications effectuées par l'algorithme en se basant sur la formule $\det A = \sum_{j=1}^n s_j a_{1j} \det A_j$. Si on ne tient pas compte des multiplications par s_j (c'est juste un changement de signe), alors :

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

- B) Sans résoudre la récurrence, que pouvez-vous dire au sujet de l'ordre de croissance de sa valeur par rapport à $n!$?

Solution :

Puisque $M(n) = nM(n-1) + n$, la fonction $M(n)$ grossit au moins aussi rapidement que la fonction factorielle qui est définie par $n! = n \times (n-1)!$, $1! = 1$.