

Interrogation d'ITC n°3

Semaine du 16/03/2026

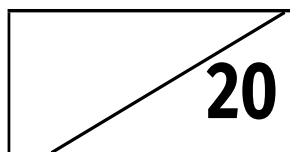
Durée : 35 minutes

Nom :

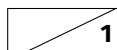
Prénom :

Consignes

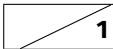
- Les codes doivent être présentés en mentionnant explicitement l'indentation, au moyen par exemple de barres verticales sur la gauche.
- Pour qu'un code soit compréhensible, il convient de choisir des noms de variables les plus explicites possibles.
- La performance du code proposé à chaque question entrera dans l'évaluation, de même que l'utilisation de boucles appropriées.
- Vous pouvez bien sûr utiliser une feuille de brouillon en parallèle.



Présentation (écriture, propreté, marqueurs d'indentation du code, etc....) :



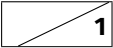
Exercice 1 On désire écrire une fonction $S(n)$ qui, pour $n \in \mathbb{N}$, renvoie la valeur de : $S_n = \sum_{k=0}^n \frac{1}{k!}$.

1. 1.1)  On rappelle la fonction ci-dessous, renvoyant la valeur de $n!$.

```
1 def f(n:int)->int:
2     if n == 0:
3         return 1
4     else:
5         return n*f(n-1)
```

On note $C(n)$ le nombre d'opérations algébriques lors de l'exécution de $f(n)$. Déterminer $\alpha \in \mathbb{N}$ de sorte que : $C(n) = O(n^\alpha)$.

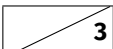


1.2)  Calculer alors la complexité $C_1(n)$ du code ci-dessous :

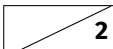
```
1 def S(n:int)->float:
2     S = 0
3     for i in range(n+1):
4         S += 1/f(i)
5     return S
```

On déterminera $\beta \in \mathbb{N}$ de sorte que : $C_1(n) = O_{n \rightarrow \infty}(n^\beta)$.



2.  Proposer une autre fonction S_2 , qui renvoie aussi la valeur S_n , de complexité linéaire cette fois-ci, c'est-à-dire en $O_{n \rightarrow \infty}(n)$. Justifier ensuite la complexité.



5.2)  On admet que la suite $(C(a, b))_a$ est croissante (relativement à a). Pour $m \in \mathbb{N}^*$, on rappelle que $2^n \leq m < 2^{n+1}$ avec $n = \lfloor \log_2(m) \rfloor$.

À l'aide de tout ce qui précède, établir que :

$$\forall b \in \mathbb{N}, \quad C(m, b) = O_{m \rightarrow \infty}(\ln(m)).$$



Correction de l'Interrogation d'ITC N°3

Solution 1

1. **1.1)** On note $C(n)$ le nombre d'opérations algébriques lors de l'exécution de $f(n)$. La fonction donne :

$$C(0) = 1 \quad \text{et} : \quad \forall n \geq 1, \quad C(n) = 2 + C(n-1).$$

En effet, on réalise soit un test (cas de base), soit 1 test et 1 multiplication si $n \geq 2$. Ainsi, $C(n) = C(0) + 2n = 1 + 2n = \boxed{O(n)}$.

- 1.2)** On a : $C_1(n) = 1 + \sum_{i=0}^n (3 + C(i)) = 1 + \sum_{i=0}^n (4 + 2i)$, car on a une affectation ligne 2, et 3 opérations ligne 4 (une affectation, une addition, un quotient ainsi qu'un appel à $f(i)$). Ainsi,

$$\begin{aligned} C_1(n) &= 1 + \sum_{i=0}^n (4 + 2i) = 1 + 4(n+1) + n(n+1) \\ &= n^2 + 5n + 5 = \boxed{O(n^2)}. \end{aligned}$$

2. Pour diminuer l'ordre de complexité, on peut recycler les anciennes valeurs de la factorielle. Le code ci-dessous répond à cette problématique.

```
1 def S2(n:int)->float:
2     S, f = 0, 1
3     for i in range(n+1):
4         S += 1/f
5         f *= i+1
6     return S
```

```
>>> S(3)
2.6666666666666665
>>> S2(3)
2.6666666666666665
```

On a : $C_2(n) = 2 + \sum_{i=0}^n 6$, car on a deux affectations ligne 2, et 6 opérations lignes 4 et 5. Ainsi, $C_2(n) = 2 + 6(n+1) = \boxed{O(n)}$.

Solution 2

1. Utilisons Python pour cela.

```
>>> inconnue(9,11)
Ai Bi Ci

9 11 0
4 22 11

2 44 11

1 88 11

0 176 99

99
```

2. On conjecture que $\text{inconnue}(a, b)$ renvoie : $\boxed{a \times b}$.
3. Montrons la terminaison. Notons A_i la valeur de la variable A à la fin de l'itération i . S'il y a une itération $i+1$, alors $A_i \neq 0$. Peu importe la parité de A_i , on a à l'itération $i+1$: $A_{i+1} = A_i / 2$.
- **[1^{er} cas]** Si A_i est pair, alors $A_i = 2k$ avec $k \in \mathbb{N}^*$ car $A_i \neq 0$. Donc $A_{i+1} = k < 2k = A_i$.
 - **[2^{ème} cas]** Si A_i est impair, alors $A_i = 2k + 1$ avec $k \in \mathbb{N}$. Donc $A_{i+1} = k < 2k + 1 = A_i$.
- Ainsi, $(A_i)_i$ est une suite strictement décroissante d'entiers.
- Si par l'absurde la boucle **while** ne s'arrête pas, alors on a $A_i > 0$ pour tout $i \in \mathbb{N}$. Or, par stricte décroissance, il existe en entier i_0 de sorte que $A_{i_0} \leq 0$ — **Contradiction**.
4. Pour $i \in \llbracket 0, N \rrbracket$, on pose $\mathcal{P}(i)$: « $A_i \times B_i + P_i = ab$ ». Montrons que $\mathcal{P}(i)$ est un invariant de boucle, déduisons la correction du programme.

Initialisation. Pour $i = 0$: $A_0 B_0 + P_0 = a \times b + 0 = a \times b$. La propriété est initialisée.

Hérédité. Soit $i \in \llbracket 0, N-1 \rrbracket$, tel que $A_i \times B_i + P_i = ab$. Alors :

- **[1^{er} cas]** Si A_i est pair, alors $P_{i+1} = P_i$, $A_{i+1} = \frac{A_i}{2}$ et $B_{i+1} = 2B_i$. Donc :

$$A_{i+1} B_{i+1} + P_{i+1} = \frac{A_i}{2} (2B_i) + P_i = A_i B_i + P_i = ab,$$

d'après l'invariant.

- **[2^{ème} cas]** Si A_i est impair, alors $P_{i+1} = P_i + B_i$ et $A_{i+1} = \frac{A_i - 1}{2}$ et $B_{i+1} = 2B_i$. Donc :

$$A_{i+1} B_{i+1} + P_{i+1} = \frac{A_i - 1}{2} (2B_i) + P_i + B_i = A_i B_i + P_i = ab,$$

d'après l'invariant.

L'invariant est donc prouvé.

Correction. Pour $i = N$, on a $A_N B_N + P_N = ab$. Or $A_N = 0$ en fin de **while**, donc $P_N = ab$ et c'est le résultat renvoyé.

5. **5.1)** Lorsque $a = 2^n$, a est alors pair et prend les valeurs $2^n, 2^{n-1}, \dots, 2^1, 2^0, 0$. Il y'a alors :

- 3 affectations avant le **while**,
- n itérations avec à chaque fois 2 tests (**while** et **if**), 1 division euclidienne dans un test, deux affectations (ligne 6,7), 1 quotient et 1 produit soit au total 7 opérations.
- 1 itération lorsque $a = 2^0 = 1$, où il y'a 9 opérations (on ajoute la ligne 5).
- Et enfin 1 test de sortie de boucle **while** lorsque $a = 1 // 2 = 0$.

Bilan : $C(2^n, b) = 3 + 7n + 9 + 1 = \boxed{7n + 13}$.

5.2) Soit $b \in \mathbb{N}$. Par croissance, on a : $C(2^n, b) \leq C(m, b) \leq C(2^{n+1}, b)$.

Or, $C(2^n, b) = 6n + 12 = 6\lfloor \log_2(m) \rfloor + 12$ et $C(2^{n+1}, b) = 6n + 18 = 6\lfloor \log_2(m) \rfloor + 18$. Donc par théorème d'encadrement et la définition de la partie entière, on a : $C(m, b) = \boxed{\underset{m \rightarrow \infty}{O}(\ln(m))}$.