

Durée : 1 heures. L'utilisation de la calculatrice n'est pas autorisée.

Le code doit être écrit en langage Python. En particulier, on prendra soin de bien respecter l'indentation.

Exercice 1

On considère la fonction suivante où les variables d'entrée sont deux entiers naturels a et n et qui doit retourner a^n :

```
def puissance(a,n):  
    P = 1  
    i = 0  
    while i <= ? :  
        P = P * a  
        i = i + 1  
    return P
```

1. On note $P_0 = 1$, $i_0 = 0$, puis P_k et i_k les valeurs prises par les variables P et i à la sortie de la k -ème itération de la boucle `while`.
Montrer que l'assertion $(P_k = a^k \text{ et } i_k = k)$ est un invariant de boucle.
2. Que doit-on écrire à la place du point d'interrogation pour obtenir le résultat voulu ?

Exercice 2

1. a. Écrire une fonction nommée `factorielle` qui prend en argument un entier naturel n et renvoie $n!$.
On utilisera une boucle `while`. On n'acceptera pas bien sûr de réponse utilisant la propre fonction `factorielle` du module `math`.
b. Justifier la terminaison de l'algorithme.
c. Justifier la correction de l'algorithme (c'est à dire que votre algorithme retourne bien $n!$). Pour cela on pourra utiliser un invariant de boucle utilisant les valeurs prises par les variables à la sortie des boucles.
2. Écrire une fonction booléenne nommée `estdivisible` qui prend en argument un entier naturel n et renvoie `True` si $n!$ est divisible par $n + 1$ et `False` sinon.
3. On considère la fonction suivante nommée `mystere` où n est un entier naturel :

```
def mystere(n):  
    s = 0  
    for k in range(1,n+1):  
        s = s + factorielle(k)  
    return s
```

- a. Quelle valeur renvoie `mystere(4)`.
- b. Déterminer le nombre de multiplications qu'effectue `mystere(n)`.
- c. Proposer une amélioration du script de la fonction `mystere` afin d'obtenir une complexité linéaire.

Exercice 3

On considère la fonction suivante nommée `enigme` où n est un entier :

```
def enigme(n):
    p = 1
    A = 2
    k = n
    while k > 0:
        if k % 2 == 1:
            p = p * A
            A = A * A
            k = k // 2
    return p
```

1. On prend $n = 41$. Compléter le tableau suivant où i désigne l'indice de boucle, p_i , A_i et k_i les valeurs prises par p , A et k après la i -ième boucle.
Par convention l'indice 0 correspond aux valeurs avant la première boucle.

i	0	1	2	...
p_i				
A_i				
k_i				

2. Justifier que $p_i A_i^{k_i} = 2^n$ est un invariant de boucle.
3. Que fait cet algorithme ? Justifier.
4. Soit n un entier naturel. On se propose de déterminer la complexité de l'algorithme `enigme(n)` en fonction de n .

On note p le nombre d'itérations lors de l'exécution de `enigme(n)`. Proposer un encadrement de p en fonction de n .

Pour cela on pourra s'aider de la décomposition de n en base 2 :

$$n = a_0 + a_1 2 + a_2 2^2 + a_3 2^3 + \dots + a_{r-1} 2^{r-1} + a_r 2^r, \text{ où } a_i \in \{0, 1\} \text{ et } a_r \neq 0.$$

Que peut-on dire de la complexité de cet algorithme ?

Exercice 4

Soit un entier naturel n non nul et une liste t de longueur n dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans t (c'est à dire figurant dans des cases consécutives). Par exemple le nombre maximal de zéros contigus de la liste $t1$ suivante vaut 4 :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$t1[i]$	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0	1	0

1. Écrire une fonction `nombreZeros(t, i)` prenant en paramètre une liste t de longueur n et un indice i compris entre 0 et $n - 1$ et renvoyant :

$$\begin{cases} 0 & \text{si } t[i] = 1 \\ \text{le nombre de zéros consécutifs dans } t \text{ à partir de } t[i] \text{ inclus, si } t[i] = 0 \end{cases}$$

Par exemple, les appels `nombreZeros(t1, 4)`, `nombreZeros(t1, 1)` et `nombreZeros(t1, 13)` renvoient respectivement les valeurs 3, 0 et 2.

2. Proposer une fonction `nombreZerosMax(t)` de paramètre t renvoyant le nombre maximal de zéros contigus d'une liste t non vide. On utilisera la fonction `nombreZeros`.
3. Que pouvez vous dire concernant la complexité de la fonction `nombreZerosMax` construite à la question précédente ?
4. Trouver un moyen simple, toujours en utilisant la fonction `nombreZeros`, d'obtenir un algorithme plus performant.