

## TP 15- Proposition de solutions

**Solution 1** 1. Cette fonction retourne True si  $n$  est premier et False sinon.

2. Elle recherche un diviseur éventuel parmi tous les entiers compris entre 2 et  $\sqrt{n}$ .

3. La variable  $d$  est incrémentée de 1 à chaque tour de boucle, ainsi elle constitue un variant. Les valeurs prises sont des entiers naturels et majorée (par  $\sqrt{n} + 1$ ) donc en nombre fini : la boucle while termine en temps fini.

4. Repérons chaque boucle par la valeur de la variable  $d$ . Au début de l'étape repérée par  $d$ , un invariant est : " $d \leq \sqrt{n}$  et  $n$  ne possède aucun facteur entre 2 et  $d$ ".

Lorsque la boucle se termine, il y a deux cas possibles (la négation d'un "et" par de Morgan donne un "ou") :

- soit  $d^2 > n$ , alors l'invariant donne que  $n$  ne possède aucun facteur (premier) inférieur à  $\sqrt{n}$  donc  $n$  est premier et la fonction retourne True.

- soit  $d$  divise  $n$ , alors  $n$  est composé et donc la fonction retourne False ;

### Solution 2 Taille binaire

1. Voici un script qui retourne la plus petite puissance de deux supérieure à  $n$  :

```
def p2(n):
    x=1
    while x<n:
        x=x*2
    return(x)
```

2. Un invariant de boucle est  $x = 2^{k-1} < n$ . La complexité est  $\mathcal{O}(\ln(n))$ .

3. La taille binaire donnée par la plus petite puissance de deux qui est strictement supérieure à  $n$  :

```
def tb(n):
    x,p=1,0
    while x<=n:
        x,p=x*2,p+1
    return(p)
```

**Solution 3** 1. Compléter l'algorithme suivant qui calcule le polynôme de Taylor de la fonction sinus

$$\sum_{k=0}^n (-1)^k \frac{x^{2k+1}}{(2k+1)!} \text{ avec } x \in \mathbb{R} \text{ et } n \in \mathbb{N}$$

---

```
s ← x
m ← -x**3
f ← 6
pour k variant de 1 à n faire
    s ← s + m/f
    m ← -x*x*m
    f ← f*(2*k+2)*(2*k+3)
finpour
afficher s
```

La récurrence est évidente de par l'écriture même de l'invariant.

2. Un invariant de boucle est, pour  $k \in [1, n]$  :

$$\begin{cases} s = \sum_{j=0}^{k-1} (-1)^j \frac{x^{2j+1}}{(2j+1)!} \\ m = (-1)^k x^{2k+1} \\ f = (2k+1)! \end{cases}$$

3. • Il y a exactement  $n$  itérations. La suite  $(n-k)$  est strictement décroissante dans  $\mathbb{N}$ . L'algorithme se termine.

- La variable  $s$  contient au début du tour  $k=n$  :

$$\sum_{j=0}^{n-1} (-1)^j \frac{x^{2j+1}}{(2j+1)!}$$

A la fin de ce tour, elle contient :  $\sum_{j=0}^n (-1)^j \frac{x^{2j+1}}{(2j+1)!}$

La correction est vérifiée.

### Solution 4 Exponentiation rapide

Prog

1. Suivi des variables lors de l'exécution de algo(-2,13) :

r	m	p	p%2 (test lig.4)
1	-2	13	∅
-2	$(-2)^2 = 4$	6	1
-2	$2^4 = 16$	3	0
$(-2)^5 = -32$	$2^8 = 256$	1	1
$(-2)^{13} = -8192$	$2^{16}$	0	1

2. On a :  $13 = 8 + 4 + 1 = 2^3 + 2^2 + 2^0 = 1101$

On note que les chiffres de l'écriture binaire de  $n$  correspondent aux valeurs de  $p\%2$  (du bit de poids le plus faible au bit de poids le plus fort).

3. Notons  $r_k$ ,  $m_k$ , et  $p_k$  les valeurs respectives des variables  $r$ ,  $m$  et  $p$  au début de la boucle de rang  $k$ .

Montrons par récurrence que pour tout  $k \in \mathbb{N}$ ,  $r_k m_k^{p_k} = x^n$ .

- Initialisation : elle est réalisée à la ligne 1 par :  $r_1 = 1$ ,  $m_1 = x$  et  $p_1 = n$ . Ainsi,

$$r_1 m_1^{p_1} = 1 \times x^n = x^n$$

La relation est vrai au rang 1.

• Hérédité : considérons un rang  $k \geq 1$  et supposons que  $r_k m_k^{p_k} = x^n$ .

Il y a une alternative concernant la parité de  $p$  (ligne 4) ; procédons par disjonction de cas :

• soit  $p_k$  est impair ; il existe  $j \in \mathbb{N}$  tel que  $p_k = 2j + 1$  :

$$\begin{cases} r_{k+1} &= r_k m_k \\ m_{k+1} &= m_k^2 \\ p_{k+1} &= j \end{cases}$$

donc  $r_{k+1} m_{k+1}^{p_{k+1}} = r_k m_k (m_k^2)^j = r_k m_k^{2j+1} = r_k m_k^{p_k}$

• soit  $p_k$  est pair ; il existe  $j \in \mathbb{N}$  tel que  $p_k = 2j$  :

$$\begin{cases} r_{k+1} &= r_k \\ m_{k+1} &= m_k^2 \\ p_{k+1} &= j \end{cases}$$

donc  $r_{k+1} m_{k+1}^{p_{k+1}} = r_k (m_k^2)^j = r_k m_k^{2j} = r_k m_k^{p_k}$

Dans les deux cas, l'hypothèse de récurrence permet d'établir :

$$r_{k+1} m_{k+1}^{p_{k+1}} = r_k m_k^{p_k} = x^n$$

Donc la relation est vérifiée au rang  $k + 1$

• Conclusion :  $rm^p = x^n$  est un invariant de boucle pour algo

4. La fonction algo se termine avec  $p$  de valeur nulle, et retourne  $r$  :

$$x^n = r m^p = r m^0 = r$$

Ainsi, l'expression retournée par algo est  $x^n$ .

**Remarque** : Cet algorithme s'appelle l'exponentiation rapide.

5. La variable  $p$  est un variant de la boucle : la suite de ses valeurs est strictement décroissante d'entiers naturels et donc prend un nombre fini d'états. Ainsi, algo termine en temps fini.

6. Évaluation de la complexité de algo :

• Les opérations à chaque tour sont (au maximum) : 2 multiplications, 2 divisions euclidiennes, 1 comparaison et 3 affectations.

Considérant ces opérations usuelles comme référence, la complexité de algo va dépendre du nombre de boucles effectuées.

• La question 2) nous suggère de considérer la longueur de l'écriture binaire de  $n$ .

Autrement dit, considérant la variable  $p$ , initialisée à  $n$ , le nombre de boucle correspond au nombre de fois que l'on peut diviser  $n$  par 2 :

$$\min \{k; 2^k \geq n\}$$

On a :

$$2^k \geq n \Leftrightarrow k \ln(2) \geq \ln(n) \Leftrightarrow k \geq \frac{\ln(n)}{\ln(2)}$$

Ainsi, la complexité temporelle de algo est  $\mathcal{O}(\ln(n))$ .

7. Version récursive de algo :

### Version récursive

```
def algo_R(x,n):
    if n==0:
        return(1)
    elif n%2==1:
        return(x*algo_R(x*x,n//2))
    else:
        return(algo_R(x*x,n//2))
```

appel	x	n%2	sortie
(-2,13)	-2	1	$(-2)^{8+4+1} = (-2)^{13}$
(4,6)	$(-2)^2 = 4$	0	$(-2)^{8+4}$
(16,3)	$2^4 = 16$	1	$(-2)^{8+4}$
(256,1)	$(-2)^8 = 256$	1	$(-2)^8$
(X,0)	X	X	1

**Rappel** : le tableau se remplit en descendant sur les appels (colonnes de gauche), puis en remontant sur la sortie (colonne de droite).

8. L'écriture binaire de 100 est :  $100 = (1100100)_2$

$$A^{100} = \underbrace{(A^2)^2}_{=B} \times \underbrace{((B^2)^2)}_{=C} \times C^2$$

Ce qui donne 8 produits.

Vérification :  $A^{2^2} \times A^{2^5} \times A^{2^6} = A^{4+32+64} = A^{100}$ .

9. La méthode de Hörner consiste à factoriser depuis l'intérieur pour optimiser le calcul des puissance :

$$P = \sum_{k=0}^p b_k X^k = (((\dots (b_p X + b_{p-1})X + \dots + b_2)X + b_1)X + b_0$$

Comme  $100 = (1100100)_2$ , il vient :

$$A^{100} = (((((A^2 \times A)^2)^2)^2 \times A)^2)^2$$

Ce qui donne aussi 8 produits.

Vérification :  $A^{((2+1) \times 2 \times 2 \times 2 + 1) \times 2 \times 2} = A^{((3 \times 8 + 1) \times 4)} = A^{100}$ .