

### Exercice 1 : Vrai ou Faux

- 1) Pour prouver la terminaison d'un programme itératif, on peut utiliser un invariant de boucle ?
- 2) Si un algorithme se termine alors le résultat est correct ?
- 3) Une assertion résout-elle l'erreur ?
- 4) Un algorithme de tri fusion qui effectue  $n$  recherches dichotomiques sur des listes de longueur  $n$  a une complexité en  $O(n \log_2(n))$
- 5) Plus la complexité est élevée, plus le résultat est précis

- 1) Non, l'invariant c'est par la correction, le variant c'est pour la terminaison
- 2) Non terminaison et correction sont deux choses différentes
- 3) Non, elle signale la présence d'un problème sans d'ailleurs préciser exactement où !
- 4) Oui
- 5) Non, temps de calcul et précision n'ont, à priori, rien à voir.

### Exercice 2 :

Dans la fonction suivante, les valeurs des variables  $a$  et  $b$  sont des entiers naturels :

```
def f(a,b):  
    s,k=a,b  
    while k>0:  
        s=s+1  
        k=k-1  
    return s
```

Quelle affirmation est fautive ?

- 1) La propriété  $s + k = a + b$  est un invariant de boucle
- 2) La valeur finale de  $k$  est 1
- 3) Le résultat renvoyé est égal à la somme  $a + b$

L'affirmation 2 est fautive.

### Exercice 3 : Correction d'un programme

On considère la fonction réalisant la division euclidienne  $a/b$  de deux entiers naturel  $a$  et  $b$  et renvoyant le quotient  $q$  et le reste  $r$ .

```
def division(a,b):
    """a est un entier naturel
    b est un entier naturel non nul
    renvoie le quotient q et le reste r de la division a/b
    a=q*b+r est un invariant de boucle et r<b """
    q=0
    r=a
    while r>b:
        q=q+1
        r=r-b
    return q,r
```

- 1) Prouver la terminaison de cet algorithme.
- 2) Chercher à montrer que ce code est correct. Conclure.
- 3) Proposer un exemple d'assertion soulignant le problème de correction de cette fonction.
- 4) Corriger ce programme et vérifier sa correction au regard des spécifications
  - 1) Ici  $r$  est le variant de boucle, initialement  $r = a$  et après  $k$  itération  $r = a - kb$ . Le nombre d'itérations est donc limité par  $r > b$  soit  $a - kb > b \rightarrow k < \frac{a-b}{b}$
  - 2) L'invariant de boucle est tout trouvé  $a = qb + r$ 
    - Initialisation :  $q = 0, r = a$  et on a bien  $a = qb + r = r$
    - Continuité : si  $a = qb + r$  alors à l'itération suivante  $q' = q + 1$  et  $r' = r - b$   
 $q'b + r' = qb + r = a$
    - Terminaison : Le programme doit continuer si  $r = b$  or il se termine si  $r < b$ . L'inégalité stricte ne permet pas d'avoir un résultat juste à une division sans reste c'est-à-dire pour  $a = nb$ . L'algorithme s'arrête alors que  $r = b$  (ce qui ne correspond pas à une division) et  $q = (n - 1)$
  - 3) `assert division(4,2)==(2,0)`
  - 4) En enlevant l'inégalité stricte, l'étape de terminaison aboutit bien à l'arrêt pour  $r < b$  et  $a = qb + r$

### Exercice 4 : Test

On considère la fonction réalisant la division euclidienne  $a/b$  de deux entiers naturels  $a$  et  $b$  et renvoyant le quotient  $q$  et le reste  $r$ .

```
def division(a,b):
    """a est un entier naturel
    b est un entier naturel non nul"""
    q=0
    r=a
    while r>=b:
        q=q+1
        r=r-b
    return q,r
```

Ecrire une fonction *test*, ne prenant aucun argument mais utilisant un *assert*, permettant de vérifier que la fonction *division* renvoie bien des valeurs de  $q, r$  telles que  $a = qb + r$  pour les valeurs des  $a, b$  suivante :

```
for a in range(0,11):
    for b in range(1,11):
```

Corrigé

```
def test():
    for a in range(0,11):
        for b in range(1,11):
            q,r=division(a,b)
            assert a==q*b+r
test()
```

### Exercice 5 :

On considère la fonction dichotomie définie ci-dessous. Tester, à l'aide de l'instruction assert, cette fonction à l'aide d'exemples simples afin de déterminer l'erreur qui s'est glissée et la corriger.

```
def dichotomie(L,elt):
    deb=0
    fin=len(L)-1
    while fin>deb:
        m=(fin+deb)//2
        if L[m]==elt:
            return True
        elif L[m]>elt:
            fin=m-1
        else:
            deb=m+1
    return False
```

Corrigé :

`L=[0,1,2,3,4,5,6,7,8,9,10]`

`assert dichotomie(L,L[5])==True#OK`

`assert dichotomie(L,L[0])==True# pb à cause de l'inégalité stricte`

`assert dichotomie(L,L[-1])==True#pb à cause de l'inégalité stricte`

### Exercice 6 :

Voici un algorithme pour déterminer le plus grand élément d'une liste non vide de nombres. Prouver sa correction

```
def maximum(L):
    i=0
    maxi=L[i]
    for i in range(1,len(L)):
        if L[i]>maxi:
            maxi=L[i]
    return maxi
```

Dans un 1<sup>e</sup> temps, on peut affirmer qu'avec la boucle bornée, ce programme se termine.

La propriété invariante est ici :  $maxi = \max(L[0:i + 1])$

- Initialisation : avant l'entrée en boucle  $L[0:i + 1] = L[0]$  et donc  $maxi = L[0]$  et est bien le maximum de cette liste à un élément
- Continuité : Supposons vrai la propriété à l'itération  $k$  ( $maxi = \max(L[0:k + 1])$ ) alors à l'itération suivante 2 cas possibles :
  - $L[k + 1] < maxi$  et donc on a bien  $maxi = \max(L[0:k + 2])$
  - $L[k + 1] > maxi$  et donc  $maxi \leftarrow L[k + 1]$  et on a bien ensuite  $maxi = \max(L[0:k + 2])$
- Terminaison : le programme se termine au dernier élément de la liste qui est alors tester comme les précédents

La fonction renvoie donc bien le maximum.

### Exercice 7 : Nombre d'opérations

Déterminer pour les cas suivants, le nombre d'opérations et donc le niveau de complexité  $T(n)$  associé. Dans les codes qui suivent, les pointillés sous-entend un nombre fixe de  $q$  d'opérations.

Cas 1 :

```
for i in range(n):
    .....
    for j in range(k):
        .....
```

Cas 2 :

```
for i in range(n):
    .....
    for j in range(n):
        .....
```

Cas 3 :

```
for i in range(n):
    .....
    for j in range(i):
        .....
```

### Exercice 7 : Nombre d'opérations

Cas 1 :

Il y a :

- $n$  passages dans la boucle externe
- A chaque passage dans cette boucle, il y a  $q$  opérations puis  $k$  passages dans la boucle interne. Donc  $q + kq$  opérations élémentaires
- Le nombre total d'opérations est  $n(q + kq) \propto nk$
- Le niveau de complexité est linéaire en  $n$

Cas 2 :

Il y a :

- $n$  passages dans la boucle externe
- A chaque passage dans cette boucle, il y a  $q$  opérations puis  $n$  passages dans la boucle interne. Donc  $q + nq$  opérations élémentaires
- Le nombre total d'opérations est  $n(q + nq) \propto n^2$
- Le niveau de complexité est quadratique

Cas 3 :

Il y a :

- $n$  passages dans la boucle externe
- A chaque passage dans cette boucle, il y a  $q$  opérations puis  $i$  passages dans la boucle interne :  $(q + iq)$ 
  - $q$  opérations élémentaires pour  $i = 0$
  - $q + q$  opérations élémentaires pour  $i = 1$
  - $q + 2q$  opérations élémentaires pour  $i = 2, \dots$
- Le nombre total d'opérations est  $nq + \frac{(n-1)n}{2}q \propto n^2$
- Le niveau de complexité est quadratique

### Exercice 8 : Etude de complexité

- 1) Estimer le nombre d'opérations effectuées par le programme suivant en fonction de  $n$  avec  $n$  un entier strictement positif.
- 2) Quelle est la complexité de l'algorithme en fonction de  $n$  ?

```
c=0
while n>0:
    c=c+1
    n=n//10
```

### Exercice 8 : Etude de complexité

Ce programme permet de connaître la plus petite puissance  $10^c$  qui majore  $n$ .

On veut  $k$  itérations telle  $\frac{n}{10^{k+1}} \approx 0,1$

On a donc typiquement  $k$  itérations avec  $k = \log_{10}(n)$

Avant la boucle :

- Une affectation

Pour les  $k$  itérations, on a :

- Une comparaison
- Deux affectations
- Une addition
- Une division

En sortie de boucle :

- Une comparaison

On a donc  $T(n) = 1 + 5k + 1$

Donc la complexité est logarithmique  $T(n) = O(\log_{10}(n))$

## Exercice 9 : comparaison de deux algorithmes

Voici deux programmes pour calculer  $2^n$  avec un entier  $n$  naturel

```
def puissance1(n):
    p=1
    for i in range(1,n+1):
        p=p*2
    return p
def puissance2(n):
    b=2
    p=1
    m=n
    while m>0:
        if m%2==1:
            p=b*p
        b=b*b
        m=m//2
    return p
```

- 1) Dans le 1<sup>e</sup> programme, quel est le nombre d'affectations et de multiplications effectuées dans le corps de la boucle en fonction de  $n$  ? Evaluer le niveau de complexité.
- 2) Prouver la terminaison du 2<sup>e</sup> programme en estimant le nombre d'itérations.
- 3) Prouver la correction du 2<sup>e</sup> programme en considérant l'invariant de boucle  $pb^m = 2^n$
- 4) Justifier alors que le niveau de complexité de cet algorithme est en  $O(\log_2(n))$

## Exercice 9 : comparaison de deux algorithmes

- 1) Dans le corps de la boucle, à chaque itération, on a une affectation pour  $p$  et une multiplication, donc avec  $n$  itérations, on a :

$$T(n) = 2n = O(n)$$

- 2) L'idée de ce programme consiste à limiter le nombre d'opérations élémentaires par une méthode dichotomique.

$$2^n = 2^{2q+r} = 2^{2q}2^r = 2^{q^2}2^r = 2^{n'}2^r$$

On voit tout de suite le gain avec :

$$2^{17} = (2)^{16+1} = 2^{16} * 2 = 2^{8^2} * 2$$

L'idée est donc d'éviter de calculer 2 fois  $2^8$  et ainsi de suite :

$2^{17} = (2^8)^2 * 2 = 2^8 2^8 * 2$	On calcule $2^8$ une seule fois
$2^8 = 2^4 2^4$	On calcule $2^4$ une seule fois
$2^4 = 2^2 2^2$	On calcule $2^2$ une seule fois
$2^2 = 2^1 2^1$	On calcule $2^1$ une seule fois
$2^1 = 2 * 2^0$	

L'algorithme prend ce calcul à l'envers :

$m = 17$	$p = 1 * 2^1$	$b = 2^1 * 2^1$	$m = 8$
$m = 8$	$p = 2^1$	$b = 2^2 2^2$	$m = 4$
$m = 4$	$p = 2^1$	$b = 2^4 2^4$	$m = 2$
$m = 2$	$p = 2^1$	$b = 2^8 2^8$	$m = 1$
$m = 1$	$p = 2^1 2^{16}$	$b = 2^{16} 2^{16}$	$m = 0$

On a une démarche dichotomique avec  $k$  itérations telles que  $\frac{n}{2^k} = 1$  et donc le programme se termine après  $\log_2(n)$  itérations « complètes ».

- 3) Pour :
  - L'initialisation :  $pb^m = 1 * 2^n = 2^n$
  - Continuité : si  $pb^m = 2^n$  à l'itération suivante  $p' = pb^{r'}$ ,  $b' = b^2$  et  $m = 2m' + r'$  donc  $p'b'^{m'} = pb^{r'}(b^2)^{\frac{m-r'}{2}} = pb^{r'}b^{m-r'} = pb^m = 2^n$
  - Continuité : la boucle s'arrête pour  $m = 0$  soit  $p = 2^n$  ce qui est conforme aux spécifications.
- 4) A chaque itération :
  - 4 affectations
  - 5 opérations

$$T(n) = O(\log_2(n))$$

### Exercice 10 :

L'objectif est de comparer deux algorithmes permettant d'évaluer la valeur d'un polynôme  $P(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$  pour une valeur de  $x$  données. La liste des coefficients de ce polynôme est noté  $a = [a_0, a_1, \dots, a_{n-1}, a_n]$

On donne le 1<sup>e</sup> algorithme ci-dessous :

```
def polynome(x,a):
    resultat=0
    for i in range(len(a)):
        resultat=resultat+a[i]*x**i
    return resultat
```

- 1) Estimer la complexité de ce programme en fonction de  $n$  en analysant le nombre de multiplications dans la boucle (on admettra que le calcul  $x^i$  nécessite  $i$  multiplications)
- 2) Le programme correspondant au 2<sup>e</sup> algorithme, appelé algorithme de Hörner, est basé sur une écriture différente du polynôme :

$$P(x) = a_0 + x(a_1 + x(a_2 + \dots) + x(a_{n-1} + xa_n) \dots)$$

Ecrire le programme itératif associé et déterminer sa complexité.

- 1) La complexité est en  $T(n) = (1 + 0) + (1 + 1) + (1 + 2) \dots + (1 + n) = n + \frac{n(n+1)}{2} = O(n^2)$
- 2) Ici la complexité est  $T(n) = n - 1 = O(n)$

```
def polynome2(x,a):
    resultat=a[-1]
    for i in range(len(a)-2,-1,-1):
        resultat=resultat*x+a[i]
    return resultat
print(polynome2(3,a))

def polynome3(x,a):
    resultat=a[-1]
    for i in range(len(a)-1):
        resultat=resultat*x+a[len(a)-2-i]
    return resultat
print(polynome3(3,a))

def polynome4(x,a):
    resultat=a[-1]
    for i in range(1,len(a)):
        resultat=resultat*x+a[len(a)-1-i]
    return resultat
print(polynome4(3,a))
```

## Exercice 11

Obtenir la représentation de différentes complexités  $T(n)$  en fonction de la longueur  $n \in [1,10]$  pour :

- La complexité linéaire  $T(n) = O(n)$
- La complexité quadratique  $T(n) = O(n^2)$
- Complexité cubique  $T(n) = O(n^3)$
- Complexité logarithmique  $T(n) = O(\log(n))$
- Complexité linéarithmique :  $T(n) = O(n \log(n))$
- Complexité exponentielle  $T(n) = O(2^n)$

