Université de Paris

L2 informatique - Année 2021–2022

TD d'Éléments d'Algorithmique n° 4

*** Les exercices marqués d'une étoile sont à faire à la maison.**

Exercice 1. *Mise en jambes.*
On considère la suite définie par

$$\begin{aligned} a_0 &= 0; \\ a_n &= 3 + a_{n-1} \qquad\qquad (n > 0). \end{aligned}$$

1. Donnez une forme close pour a_n. $a(n) = 3n$
2. Écrivez une fonction récursive qui calcule a_n en appliquant directement la définition ci-dessus. Cette fonction est-elle récursive terminale ?
3. Écrivez une fonction récursive terminale qui calcule la même chose que la fonction précédente.
4. Écrivez une fonction itérative (avec des boucles) qui calcule encore la même chose.

Exercice 2. *Suite de Fibonacci.*
Leonardo di Pisa, dit Fibonacci, élève des lapins. Les lapins de Fibonacci ont un cycle de vie un peu particulier : sa première année, un lapin est immature, et ne se reproduit donc pas. À partir de sa deuxième année, un lapin produit chaque année un autre lapin par parthénogénèse. Les lapins sont immortels. Il y a donc un lapin l'année zéro, un lapin la première année, deux lapins la deuxième année, trois lapins la troisième année, etc.
La *suite de Fibonacci* (f_n) est définie par la récurrence suivante :

$$\begin{aligned} f_0 &= 1; \\ f_1 &= 1; \\ f_n &= f_{n-2} + f_{n-1} \qquad\qquad (n \geq 2). \end{aligned}$$

1. Écrivez l'algorithme récursif qui calcule f_n suggéré par le système d'équations ci-dessus.
2. Écrivez l'arbre des appels effectués pour calculer f_4, et comptez le nombre d'additions.
3. À quelle formule de récurrence obéit le nombre d'additions ? Donnez les premiers termes de la suite.
4. Écrivez un algorithme itératif en espace constant (sans utiliser de tableaux) qui calcule f_n. Combien d'additions fait-il pour calculer f_4 ? Pour calculer f_n ?
5. Lorsqu'on écrit une fonction sur la forme récursive terminale, on stocke en général plus d'informations dans les paramètres.
   Ecrivez un algorithme récursif terminal qui calcule f_n.

6* Implémentez vos algorithme des questions 1, 4 et 5. Comparez le temps d'exécution qu'il prend pour calculer $f_{30}, f_{35}, f_{40}, f_{45}\ldots$ sur ces trois implémentations. (Pour des valeurs plus grandes que 45, vous devrez utiliser des long, et pas des int. Ce n'est pas nécessaire d'utiliser un outil de programmation. Vous mettrez en commentaire toute observation significative que vous avez pu faire sur ces comparaisons.)

Exercice 3. *Récursivité et état.*
On suppose donnée une fonction `char` qui convertit un entier compris entre 0 et 9 en un caractère compris entre `'0'` et `'9'`, et une fonction `afficher` qui affiche un caractère. On note `div` le quotient de la division entière, et `mod` le reste de la division entière. On considère ensuite la fonction H définie comme suit :

```
1 H(n):
2 if n > 0 {
3     H(n div 10)
4     afficher(char(n mod 10))
5 }
```

1. Décrivez l'arbre des appels effectués lors de l'évaluation de $H(123)$.
2. Que fait cette fonction ? Est-elle récursive terminale ?
3. Écrivez une version itérative (non récursive) de cette fonction. De combien d'espace supplémentaire cette version a-t-elle besoin ? Comparez cela à la version récursive. Quel est l'interêt de la récursion ?

Exo 1 : 1) $a(n) = 3 \cdot n$

2) public static int A(n) {
if (n==0) return 0;
else return 3+A(n-1);
}

3) fonction A(n, a) :
if n = 0 retourne a
sinon retourne A(n-1, a+3)

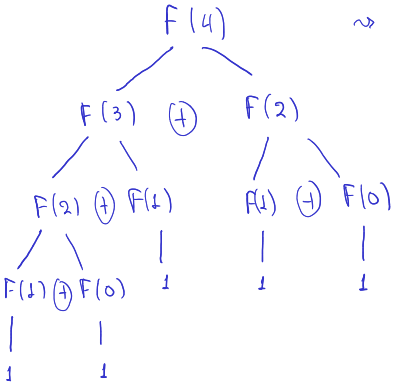
fonction AIER(n) :
retourne A(n,0)

4) fonction AIER(n) :
a = 0
pour (i = 1...n)
a = a+3
retourne a

Exo 2 :

1) fonction F(n) :
si n=0 || n=1 retourne 1
sinon retourne F(n-1)+F(n-2)

2)



⇝ F(4) = 5 / A(4) = 4 additions

3) A(n) :

A(0) = 0

A(1) = 0

A(n) = 1 + A(n-1) + A(n-2)

F(n) :

F(0) = 1

F(1) = 1

F(n) = F(n-1) + F(n-2)

A(n) = A(n-1) + 1 + A(n-2)

A(0)=0, A(1)=0, A(2)=1, A(3)=2, A(4)=1+2+1=4

A(5)=1+4+2=7, A(6)=1+7+4=11 ....

4) fonction FITER(n) :

f0 = 1

f1 = 1

pour (i = 2 ... n)

x = f0 + f1

f0 = f1

f1 = x

retourne f1

(f0_new = f1_old ; f1_new = f1_old + f0_old)

espace additionel utilisé (3) constant.

fonction FITER2(n) :

tableau T de taille n

T[0] = 1 ; T[1] = 1

pour (i = 2 ... n)

T[i] = T[i-1] + T[i-2]

retourne T[n]

espace additionel utilisé

O(n) linéaire.

5) fonction FINTER(n, a, b) : (a = f0, b = f1)

si n = 0 retourne a

si n = 1 retourne b

sinon retourne FINTER(n-1, b, a+b)

fonction FTER(n) :

retourne FINTER(n, 1, 1)

6) Utiliser la fonction static long System.currentTimeMillis() qui renvoie l'heure curnante en millisecondes pour calculer le temps écoulé.