M1.L4 : Série d'exercices sur les algorithmes / récursivité

1 Que font ces algorithmes?

Voici quatre algorithmes :

algo1
entrée : entier naturel n sortie :??

Si $n = 0$
   sortir : 0
$k \leftarrow 0$
Pour j allant de 1 à $2n - 1$
   Si j est impair
      $k \leftarrow k + j$
sortir : k

algo2
entrée : entier naturel n sortie :??
sortir : n^2

algo3
entrée : entier naturel n sortie :??
Si $n = 0$ sortir : 0 sortir : $2n - 1$+algo3($n - 1$)

algo4
entrée : entier naturel n sortie :??

Si $n = 0$
   sortir : 0
Si $n = 1$
   sortir : 1
sortir : n+ algo4($n - 2$)

Pour chacun des quatre algorithmes ci-dessus, répondre aux questions suivantes :

a) En général, que fait l'algorithme?
(si vous répondez juste à cette question, toutes les suivantes sont faciles)

b) Que se passe-t-il si on exécute l'algorithme avec la valeur d'entrée 6?

c) Quelle est l'ordre de complexité de l'algorithme? (utiliser la notation de Landau $O(\cdot)$)

d) L'algorithme est-il récursif ou non?

Voici quatre autres algorithmes :

algo5	algo6
entrée : entier naturel n sortie :??	entrée : entier naturel n sortie :??
Si $n = 0$ sortir : 0 sortir : n+ algo5(n)	Si $n = 0$ sortir : 0 sortir : n+ algo6(n +1)
algo7	algo8
entrée : entier naturel n sortie :??	entrée : entier naturel n sortie :??
Si $n = 0$ sortir : 0 Si $n = 1$ sortir : 1 sortir : 2·algo7(n−1) - algo7(n−2)	Si $n = 0$ sortir : 0 Si $n = 1$ sortir : 1 sortir : algo8(2n)+ algo8(2n − 1)

f) Un seul de ces algorithmes fonctionne correctement : lequel?

g) Et celui qui fonctionne a un gros défaut : lequel?

h) Question subsidiaire : que calcule-t-il?

2 Au temps des Egyptiens, troisième partie

Proposez une version récursive de l'algorithme vu à l'exercice 5 de la série M1.L1 et à l'exercice 2 de la série M1.L2 sous la forme d'une boucle conditionnelle :

devinette1
entrée : a, b deux entiers naturels non nuls sortie :??
c ← 0 ***Répéter*** ***Si*** ((b mod 2) = 1) *// si b est impair* c ← c + a b ← b/2 *// division entière* a ← 2.a ***Tant que*** b > 0 ***Sortir*** c

Au temps des Grecs

Proposez une version récursive de l'algorithme d'Euclide, qui calcule le plus grand diviseur commun de deux entiers naturels a et b. Voici une variante différente de celle vue à la série M1.L2 :

Algorithme d'Euclide
entrée : a, b deux entiers naturels non nuls sortie : pgcd (a,b)

$r \leftarrow 0$
Tant que $b > 0$
    $r \leftarrow a \bmod b$
    $a \leftarrow b$
    $b \leftarrow r$
sortir : a

3 Tri-Fusion

a) Dessiner les étapes de l'exécution du tri-fusion vu en cours sur la liste {20, 3, 4, 12, 2, 16, 3}.
b) Cet algorithme s'exécute-t-il avec le même nombre d'étapes pour la liste {1,2,3,4,5,6,7} ?

4 Taille de liste : le retour

Cet exercice reprend la donnée de l'exercice 3 de M1.L3. Proposez une version récursive de l'algorithme **TailleDichotomique(L, a, b)** qui cherche la taille inconnue t d'une liste L sachant qu'elle appartient à l'intervalle [a, b[.

TailleDichotomique
entrée : L, a, b sortie : le nombre d'éléments de L

Tant que $a < b - 1$
    $c \longleftarrow a + \left\lfloor \frac{b-a}{2} \right\rfloor$
    Si a_element(L , c)
        $a \leftarrow$ c
    Sinon
        b ← c
sortir a

<u>Rappel</u> : on dispose de algorithme **a_element(L , i)** qui nous dit (vrai ou faux) s'il existe un ***i*^e^** élément dans la liste : il répond donc **vrai** si i est inférieur ou égal la taille de la liste et **faux** sinon.[1]

[1] . Notez qu'il n'est pas évident de toujours avoir un tel algorithme (sans avoir taille, bien sûr!). En réalité cela dépend de la représentation effective de L. Mais nous faisons ici l'hypothèse qu'un tel algorithme existe pour L.