Université Bordeaux I — **Algorithmique et Structures de Données**
Master 1 BioInformatique 2005-2006 — **Devoir surveillé**

Mardi 8 Novembre 2005 — Durée 1h30

Exercice 1

```
mystere (n) {
    s <- 0;
    tant que n > 9 faire {
        s <- s + n % 10;
        n <- n / 10;
    }
    retourner s + n;
}
```

1. Faire tourner l'algorithme `mystere` avec `n = 1984`

s	n	$n\%10$
0	1984	4
4	198	8
12	19	9
21	1	

$s + n = 22$, la fonction retourne 22.

2. Que fait la fonction `mystere` ? Elle retourne la somme des chiffres de `n`.
3. Quelle est sa complexité ?

On passe $\lfloor log_{10}(n) \rfloor$ fois dans la boucle.
La complexité est en $O(log(n))$.

Exercice 2

On rappelle l'algorithme de recherche dichotomique d'un élément `x` dans un tableau trié `t` :

```
def dichoiter(x, t) {
    g <- 0;
    d <- len(t) - 1;
    tant que g <= d faire {
        m <- (g + d) / 2;
        si t[m] = x alors
            retourner m;
        si x < t[m] alors
            d <- m - 1;
        sinon
            g <- m + 1;
    }
    retourner -1;
}
```

Proposez une version récursive de cet algorithme. On pourra envisager une fonction récursive `dichorec(x, t, g, d)` qui recherche l'élément `x` dans les cases `g` à `d` du tableau `t`.

```
def dichorec(x, t, g, d):
    if g > d:
        return -1
    m = (g + d) / 2
    if x == t[m]:
        return m
    if x < t[m]:
        return dichorec(x, t, g, m - 1)
    return dichorec(x, t, m + 1, d)

def recherche(x,t):
    return dichorec(x,t,0,len(t)-1)
```

Exercice 3

On considère la famille de suites récurrentes (U_n^v) paramétrées par leur valeur initiale v et définies par :

$$u_0 = v$$
$$u_{n+1} = \begin{cases} u_n/2 & \text{si } u_n \text{ est pair} \\ \dfrac{3u_n + 1}{2} & \text{si } u_n \text{ est impair} \end{cases}$$

1. Pour $v = 3$, écrire les valeurs des 6 premiers termes de la suite : $u_0, u_1, u_2, u_3, u_4, u_5$.

$$u_0 = 3, u_1 = 5, u_2 = 8, u_3 = 4, u_4 = 2, u_5 = 1$$

2. Écrire la fonction `suivant(u)` qui retourne le terme suivant un terme `u` dans une telle suite.

```
def suivant(u):
    if u % 2 == 0:
        return u/2
    return (3 * u + 1)/2
```

3. Écrire la fonction `u_n(n,v)` qui retourne le terme u_n de la suite (U_n^v).

```
def u_n(n,v):
    if n == 0:
        return v
    return suivant(suite(n-1,v))
```

4. À quoi peut servir la fonction suivante (dans laquelle `v` définit la suite (U_n^v)) :

```
atteint_un (v) {
    u <- v;
    i <- 1;
    tant que vrai faire {
        u <- suivant(u);
        si u = 1 alors
            retourner i;
        i <- i + 1;
    }
}
```

Cette fonction retourne la première valeur de n pour laquelle $u_n = 1$ si une telle valeur [a]existe. Elle boucle sinon.

[a]Expérimentalement, il existe toujours une telle valeur de n. Mais ce fait n'est toujours pas prouvé à ce jour. Cette suite est connue depuis l'antiquité sous le nom de "Suite de Syracuse".