Correction : Examen partiel

Exercice 1 Complexité

A) $T(n) = 5T(n/2) + O(n)$ donc $\Theta(n^{\log_2 5})$ car $\log_2 5 > 1$.

B) $T(n) = 2T(n-1) + O(1)$ donc $\Omega(2^n)$.

C) $T(n) = 9T(n/3) + O(n^2)$ donc $O(n^2 \log n)$ car $\log_3 9 = 2$

L'algo C est le plus rapide car $\log_2 5 > \log_2 4 = 2$.

Exercice 2 Soit f telle que $f(xy) = f(x) + f(y)$.

1. Avec $y = 1$, on a $f(x) + f(1) = f(x)$ donc $f(1) = f(x) - f(x) = 0$.
2. Avec $x = x/y$, on a $f(x/y) + f(y) = f(x)$ donc $f(x/y) = f(x) - f(y)$.
3. $\log_b(x)$ est le réel y tel que $b^y = x$.
4. $b^u = xy$, $b^v = x$, et $b^w = y$ implique $b^u = b^{v+w}$. D'où, comme $b > 1$, $u = v + w$.
5. $a^u = x$, $a^v = b$, $b^w = x$ implique $a^u = a^{vw}$ d'où $u = vw$.

Exercice 3 (Algorithme de Karatsuba).

Soient A,B deux entiers de n-digits. $A = a.10^{n/2} + b$ et $B = c.10^{n/2} + d$, où a,b,c,d sont $n/2$-digits. $AB = ac.10^n + (ad + bc).10^{n/2} + bd$. Avec $(ad + bc) = ac + bd + (b - a)(c - d)$ on peut calculer AB en n'effectuant que 3 appels récursifs de produit sur des entiers $n/2$-digits. Comme l'addition est $\Theta(n)$, on a $T(n) = 3T(n/2) + \Theta(n)$, donc la complexité est $\Theta(n^{log_2 3})$.

Exercice 4 1. Par convention, on note $g(n) = O(f(n))$ pour $g(n) \in O(f(n))$, avec les notations:

- $O(f(n)) = \{g(n) : 0 \leq g(n) \leq cf(n)$, pour une constante $c > 0$ et n suffisamment grand$\}$,
- $\Omega(f(n)) = \{g(n) : g(n) \geq cf(n)$, pour une constante $c > 0$ et n suffisamment grand$\}$
- $\Theta(f(n)) = \{g(n) : c_1 f(n) \leq g(n) \leq c_2 f(n)$, pour deux constantes c_1,$c_2 > 0$ et n suffisamment grand$\}$

2. Soit $g(n) = \sum_{i=1}^{i=k} a_i O(f(n)$. Alors $\exists c_1, \ldots, c_k$ tels que $g(n) \leq \sum_{i=1}^{i=k} a_i c_i f(n)$. D'où $g(n) \leq c \sum_{i=1}^{i=k} a_i f(n)$ avec $c = \max_i c_i$. Donc $g(n) = O(\sum_{i=1}^{i=k} a_i f(n))$.

Soit $g(n) = O(\sum_{i=1}^{i=k} a_i f(n))$. Alors $\exists c$ tel que $g(n) \leq c \sum_{i=1}^{i=k} a_i f(n)$. Donc $g(n) \leq \sum_{i=1}^{i=k} a_i c_i f(n)$ avec $c_i = c$ pour tout i. Donc $g(n) = \sum_{i=1}^{i=k} a_i O(f(n))$.

Il s'ensuit $O(\sum_{i=1}^{i=k} a_i f(n)) = \sum_{i=1}^{i=k} a_i O(f(n))$.

Exercice 5 1. Un code java implémentant cet algorithme est:

```
static int Alkwa (int a, int b){
  int c=0;
  while (a !=0){
    if (a%2==1) c+=b;
    a/=2;
    b*=2;
  }
  return c;
}
```

2. Preuve de validité. On peut décomposer les valeurs a et b entrées en base 2, donc on a $a = \sum_{i=0}^{i=n} a_i 2^i$ et $b = \sum_{i=0}^{i=n} b_i 2^i$ avec $a_i,b_i \in \{0,1\}$ pour tout $i = 1,\dots,n$. Donc

$$ab = \sum_{i=0}^{i=n} a_i b 2^i$$

Par récurrence, après un nombre k de passages dans la boucle while, on a les trois invariants suivants:

1. $\mathtt{a} = \sum_{i=k}^{i=n} a_i 2^{i-k}$
2. $\mathtt{b} = b2^k$
3. $\mathtt{c} = \sum_{i=0}^{i=k-1} a_i b 2^i$

Pour $k = 0$ c'est clairement vrai puisque $\mathtt{a} = a$, $\mathtt{b} = b$, et $\mathtt{c} = 0$. L'invariant 1 se conserve car faire $\mathtt{a} \leftarrow \lfloor \mathtt{a}/2 \rfloor$ revient à faire $a_k \leftarrow 0$ et $2^i \leftarrow 2^{i-1}$ dans l'égalité. L'invariant 2 se conserve clairement par $\mathtt{b} \leftarrow 2\mathtt{b}$. L'invariant 3 se conserve car $\mathtt{a}$ modulo 2 =1 ssi $a_k = 1$.

Donc l'algorithme retourne bien $\sum_{i=0}^{i=n} a_i b 2^i$.