Université Bordeaux 1 — **Algèbre et calcul formel – Master**
Mathématiques — Année 2012–2013

FEUILLE D'EXERCICES n° 2

Exercice 1 – Soit `Fer` la procédure définie récursivement par le code Sage suivant.

```
def Fer(n):
  if n==0:
    return 2
  else:
    return Fer(n-1)*Fer(n-1)
```

1) Que calcule `Fer` ?

2) Déterminer la complexité algébrique de `Fer`.

3) Comment améliorer de façon significative cette complexité, en changeant un seul élément de la procédure ? Calculer la complexité obtenue.

Exercice 2 – [Fibonacci]

Soit `Fib` la procédure définie récursivement par le code Sage suivant.

```
def Fib(n):
  if n<=1:
    return n
  else:
    return Fib(n-1)+Fib(n-2)
```

1) Que calcule `Fib` ?

2) Montrer que pour tout n, on a

$$\texttt{Fib}(n) = \frac{1}{\sqrt{5}}\big(\Phi^n - \overline{\Phi}^n\big),$$

où $\Phi = (1+\sqrt{5})/2$ est le nombre d'or et où $\overline{\Phi} = (1-\sqrt{5})/2$ est son conjugué.

3) En déduire que la complexité algébrique de `Fib` est exponentielle (établir que si c_n est le nombre d'additions effectuées pour calculer $\texttt{Fib}(n)$, on a $c_n = \texttt{Fib}(n+1) - 1$).

4) Proposer pour le calcul de $\texttt{Fib}(n)$ un algorithme itératif de complexité en $O(n)$.

Exercice 3 – [Horner]

Soit `Calc` la procédure définie par le code Sage suivant

```
def Calc(n,T,x):
  u = [1]
  s = 0
  for i in range(1,n+1):
    u.append(x*u[i-1])
```

```
  for i in range(0,n+1):
    s = s+u[i]*T[i]
  return s
```

où n est un entier naturel, T une liste de réels dont les indices vont de 0 à n, et où x est un réel.

1) Que calcule `Calc` ?

2) Montrer que la procédure suivante calcule la même chose.

```
def Horn(n,T,x):
  s=T[n]
  for i in range(n-1,-1,-1):
    s=T[i]+x*s
  return s
```

3) Comparer les complexités algébriques de `Calc` et `Horn`.

Exercice 4 – [EXPONENTIATION RAPIDE, SQUARE AND MULTIPLY]

On se donne ici un entier n dont on considère le développement binaire

$$n = \sum_{i=0}^{m} a_i 2^i \quad (\text{avec } a_i \in \{0,1\} \text{ pour tout } i \text{ et } a_m = 1).$$

Notons que c'est sous cette forme qu'est défini n sur machine. L'objectif est de calculer de façon économique x^n où x est un réel donné.

1) Montrer que la seule connaissance des x^{2^i} $(i \leqslant m)$ permet de calculer x^n.

2) En tenant compte de la précédente observation et en observant le lien qu'il y a entre les divisions successives de n par 2 et les a_i, rédiger un algorithme récursif permettant de calculer x^n.

3) En donner une version itérative.

4) Comparer la complexité algébrique de ces algorithmes avec celles des versions naïves $x^n = x * x^{n-1}$ (cas récursif) ou $x^n = \prod_{i=1}^{n} x$ (cas itératif).

Note. Cette méthode a été utilisée par Euler (1758) - de façon modulaire - pour calculer $7^{160} \mod 641$. On peut aussi l'utiliser pour établir la non-primalité du cinquième nombre de Fermat $F_5 = \texttt{Fer}(5) + 1$ en montrant que $5 \cdot 2^7 + 1 = 5^4 + 2^4$ divise F_5 (Euler 1732). En effet $F_5 = ((2^8)^2)^2 + 1$ et deux élévations au carré suffisent.