info719 : Rappels et compléments de programmation
TD 2 : complexité

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22, poste : 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/
wiki : http://www.lama.univ-savoie.fr/wiki

Exercice 1 : classes de complexité

Question 1. À l'aide de petites fonctions ecrite en Python, remplissez le tableau suivant. La première colonne indique la complexité en microseconde d'un algorithme pour une entrée de taille n. Pour chaque colonne, estimez le temps d'exécution de cet algorithme pour une entrée de la taille donnée.

	5	10	20	40	100	500
n	5×10^{-6} s	10^{-5} s	2×10^{-5} s	4×10^{-5} s	10^{-4} s	5×10^{-4} s
$n\log(n)$						
n^2						
n^3						
n^5						
2^n						
3^n						
n^n						
2^{2^n}						

Question 2. Quelles sont les classes de complexité "utilisables" ?

Si on estime que la loi de Moore est valide (la puissance de calcul double tous les deux ans), quelles sont les classes de complexité qui peuvent basculer du "infaisable" dans le "raisonnable" ?

Exercice 2 : les nombres entiers

Question 1. En cryptographie, on manipule régulièrement des nombres entiers de quelques centaines de chiffres. Par défaut, les entiers d'un ordinateur moderne sont limités (ils sont stockés sur quelques octets).

Python utilise automatiquement une notion de grand entier quand c'est nécessaire. D'autre langages peuvent nécessiter l'utilisation d'une bibliothèque appropriée.

À votre avis, comment peut-on représenter un entier de plusieurs centaines de chiffres si on ne dispose que du type *entier sur 4 octets* ?

Question 2. Décrivez succinctement les algorithmes d'addition et de multiplication. (Il n'est pas nécessaire de les programmer.) Quelles sont leurs complexités ?

Réfléchissez à la division et au modulo...

Question 3. Une autre opération fondamentale en cryptographie est la puissance. Étant donné un grand entier x et un autre grand entier n, on veut calculer le résultat de

$$x^n \pmod{m}$$

où m est un grand entier.

Essayez de programmer cette opération, et estimez sa complexité *en fonction du nombre d'opérations arithmétiques sur les grands entiers.*

Qu'en pensez-vous ? Cette opération est-elle utilisable sur des grands entiers ? Pouvez-vous l'améliorer ?

Exercice 3 : les polynômes

Question 1. combien de multiplications sont nécessaires pour évaluer un polynôme de degré n sur un entier ? (On suppose que tous les coefficients sont non-nul.)

Question 2. quel est le nombre (exact) de multiplications et additions utilisées si on calcule les puissances "normalement", mais en réutilisant les calculs. (On suppose à nouveau que tous les coefficients sont non-nuls.)

Question 3. même question si on utilise la règle de Horner :

$$c_0 + c_1 x + c_2 x^2 + \ldots + c_n x^n \quad = \quad \Big(\ldots \Big(\big((c_n x + c_{n-1}) x + c_{n-2} \big) x + c_{n-3} \Big) \ldots \Big) x + c_0$$

Question 4. en utilisant la fonction `random.randrange` de la bibliothèque `random`, générez des polynômes aléatoires et testez l'efficacité des différentes version de votre fonction d'évaluation.

Pour chronométrer facilement une fonction, vous pouvez utiliser la fonction `time.time()` de la bibliothèque `time` :

```
import time
t = time.time()
# truc à chronométrer
...
t = time.time() - t
print 'temps : %f ms' % (t*1000)
```

Question 5. Pensez-vous que l'on peut améliorer la fonction d'évaluation pour les polynômes *creux*. (Les polynômes où la plupart des coefficients sont nuls.)