Informatique en CPGE (2018-2019)
TP 8 : algorithmes, validité et complexité

La fonction `time` du module `time` retourne le temps en seconde, du type **float**, écoulé depuis le premier jour de l'année 1970.

Les instruction suivantes permettent d'afficher le temps d'exécution d'un programme :

```
from time import time
start=time()
...
programme
...
print(time()-start)
```

Exercice 1

Dans le programme qui suit, on calcule 2^i puis 1.01^i pour 20000 valeurs de i entier ; deviner quel sera le calcul le plus rapide ?

```
from time import time

st=time()
for i in range(20000):
    c=2**i
print(time()-st)

st=time()
for i in range(20000):
    c=1.01**i
print(time()-st)
```

Ecrire le programme et le tester. Quelle peut être l'explication du comportement observé ?

Modifier le programme pour calculer cent millions de fois 2^{950} puis cent millions de fois $1,9^{950}$ et comparer les temps de calcul. On se contentera de dix millions si l'ordinateur est lent. Commentaires ?

Exercice 2

Pour calculer x^n avec n entier naturel, on peut utiliser un algorithme "naïf" dont le niveau de complexité est en $\mathcal{O}(n)$:

```
def puissance(x,n):
    p=1
    for k in range(1,n+1):
        p=p*x
    return p
```

Quel est le nombre exact de multiplications effectuées en fonction de n ?

Etudier le programme suivant et expliquer ce qu'il fait.

```
def rapide(x,n):
    p=1
    while n>0:
        n,r=n//2,n%2
        p=p*x**r
        x=x*x
    return p
```

On choisit $x = 3$ et $n = 13$. Donner l'écriture binaire de 13. Compléter le tableau ci-dessous avec les valeurs respectives de r, p, x et n après chaque passage dans la boucle while.

	initialisation	passage 1	passage 2	passage 3	passage 4
r					
p	1				
x	3				
n	13				
$p * x^n$	1594323				

Prouver que l'algorithme se termine et qu'il est valide en utilisant l'invariant de boucle : px^n.
Déterminer le nombre exact d'opérations mathématiques ("//", "%", "*") effectuées.
Montrer que le niveau de complexité de cet algorithme est en $\mathcal{O}(\ln n)$.
Compléter avec un compteur le programme qui affichera en sortie le nombre d'opérations effectuées. Vérifier que pour $n = 263$ et $x = 2$, le compteur affiche 45 opérations.

Exercice 3

On compare deux algorithmes permettant d'évaluer la valeur de $P(x)$ pour une valeur de x donnée avec $P(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$. On note $a = [a_0, a_1, \ldots, a_n]$.

1. Le programme correspondant au premier algorithme ("naïf") est le suivant :

```
def poly(x,a):
    p=0
    for i in range(len(a)):
        f=1
        for j in range (1,i+1):
            f*=x
        p+=a[i]*f
    return p
```

Déterminer la complexité de cet algorithme.
Tester ce programme avec : $a = [1.3, -3, 1, 4, 8, -4, 2, 1, 5, -6, 3]$, $(n = 10)$.

2. Le programme correspondant au deuxième algorithme, appelé *algorithme de Hörner*, est basé sur une écriture différente de $P(x)$: $P(x) = a_0 + x(a_1 + x(a_2 + \ldots + x(a_{n-1} + xa_n)\ldots))$.
Ecrire le programme et déterminer la complexité de cet algorithme.

3. Quel est l'algorithme le plus efficace ?

Mesurer pour chaque programme le temps de calcul des 100000 valeurs de $P(x)$ pour x entier variant de 0 à 99999, avec : $n = 10$ et $a = [1.3, -3, 1, 4, 8, -4, 2, 1, 5, -6, 3]$.

Mesurer ensuite le temps de calcul pour chaque programme des 100000 valeurs de $P(x)$ pour x entier variant de 0 à 99999, avec : $n = 20$
et $a = [1.3, -3, 1, 4, 8, -4, 2, 1, 5, -6, 3, 8, -4, 8, 2, -5, 7, 3, -4, 2, 5]$.

Par quels facteurs respectifs sont multipliés les temps de calcul de chaque programme ?
Commentaires ?