Correction TD2

Algorithm I

Matthieu Jimenez
Hiver 2015

Correction TD2

Algorithm I

Exercice I

Calculer:

15n+12=O(n)

17n2+3n+4= O(n2)

25n3+20log(n) = O(n3)

Correction:

a) D'après la définition on doit prouver: 15n+12<=cn pour c>0 et n suffisamment grand.
Divisant par n on obtient:
15+12/n<=c.
On peut donc choisir n>=1 (n0=1) et c=27.

En effet le maximum de 12/n sera atteint pour n=1, donc le maximum qu'atteindra 15+12/n pour n>=1 sera 27donc en prenant c=27, l'equation sera toujours respecté.

b) D'après la définition on doit prouver: 17n2+3n+4<=cn2 pour c>0 et n suffisamment grand. Divisant par n on obtient:
17+3/n+4/n2<=c.
On peut donc choisir n>=1 (n0=1) et c=24.

De même les maximums de 3/n et 4/n2 seront tout 2 atteint pour n=1.

c) D'après la définition on doit prouver: 25n3+20log(n) <=cn3 pour c>0 et n suffisamment grand. Divisant par n on obtient:
25+20log(n)/n3 <=c

On peut donc choisir n>=1 (n0=1) et c=45.

Attention: le maximum de la fonction 20log(n)/n3 n'est pas atteint pour n= 1 pour n=1,40 soit n=2 (n étant un nombre entier), à ce moment 20log(n)/n3 = 0.75. On aurait donc pu choisir c= 25.75 ou 26 mais prendre c=45 permet de ne pas prendre de risque.

Exercice II

Enoncé:

Calculer:

$2n3+10n+17=\Theta(n3)$

$n/3+15=\Theta(n)$

Correction:

Pour démontrer $\Theta(f(n))$, il est nécessaire de démontrer $\Omega(f(n)$ et $O(f(n))$,

a) $2n3+10n+17=\Theta(n3)$

  a) $2n3+10n+17=\Omega(n3)$

   D'après la définition on doit prouver: $2n3+10n+17>=cn3$ pour $c>0$ et n suffisamment grand. Divisant par n on obtient:
   $2+10/n2+17/n3>=c$.
   On peut donc choisir $n>=1$ ($n0=1$) et $c=2$

En effet, on cherche ici le minimum que pourra atteindre cette fonction, or plus n tendra vers l'infini plus $10/n2$ & $17/n3$ tendront vers 0, cette fonction tendra donc vers 2, on peut donc le choisir comme c.

  b) $2n3+10n+17=O(n3)$

   D'après la définition on doit prouver: $2n3+10n+17<=cn3$ pour $c>0$ et n suffisamment grand. Divisant par n on obtient:
$2+10/n2+17/n3>=c$.
On peut donc choisir $n>=1$ ($n0=1$) et $c=29$

b) $n/3+15=\Theta(n)$

  a) $n/3+15=\Omega(n)$

   D'après la définition on doit prouver: $n/3+15>=cn$ pour $c>0$ et n suffisamment grand. Divisant par n on obtient:
   $2+10/n2+17/n3>=c$.
   On peut donc choisir $n>=1$ ($n0=1$) et $c=1/3$

  b) $n/3+15=O(n)$

   D'après la définition on doit prouver: $n/3+15<=cn$ pour $c>0$ et n suffisamment grand. Divisant par n on obtient:
$1/3+15/n>=c$.
On peut donc choisir $n>=1$ ($n0=1$) et $c<15,34$ ou $c=16$

Exercice III

Enoncé:

Une méthode simple pour découvrir le mot de passe d'un compte est d'utiliser le Brute Force, c'est à dire de tester toutes les possibilités de mots de passe existants En supposant que le mot de passe que vous tentez de découvrir par Brute Force est composé de n caractères alphanumériques (A-Z,a-z,0-9), quel sera la complexité dans le pire des cas du code que vous utiliserez?

Correction:

Dans le pire des cas, toutes les possibilité devront être testé soit 64^n soit une complexité Cn avec C=64. Ce type de complexité est celle que l'on souhaite le plus éviter. Le Brute Force n'est donc pas la solution la plus rapide.

Exercice IV

Enoncé:

On veut écrire un algorithme qui décide si un mot est un palindrome ou non. Un mot est un palindrome si sa première lettre est identique à la dernière, sa deuxième à l'avant dernière etc…

1. Proposez un pseudo-code

- input: Chaine de caractères
- output: boolean

2. Quelle est la complexité de votre algorithme?

Correction:

```
boolean isPalindrome(String A ){
    int gauche=1;
    int droite = A.size();
    while (gauche<droite){
        if (A[gauche] != A[droite])
            return false;
        gauche ++;
        droite –;
    }
    return true;
}
```

D'autres solutions peuvent être envisagées, comme l'utilisation d'une boucle for sur la moitié de la chaine de caractères.

Remarque: Faîtes très attention à vos index, si on commence à 0, on s'arrête à n-1 et si on commence à 1 on s'arrête à n.

Attention également à ne pas utiliser de double boucle!

La complexité de cette algorithme sera O(n/2) car on boucle sur la moitié du mot (pire cas), toutefois comme aucun coefficient ne doit apparaitre sur une complexité, **la complexité est O(n)**

Exercice V

Enoncé:

Quel sera la complexité de cet algorithme?

Donner directement sous la forme Big O

```
int sum = 0;
for (int i = 1; i <= N; i++)
      for (int j = 1; j <= i*i; j++)
            for (int k = 1; k <= j; k++)
                  sum++;
```

Correction:

Ici, on a une première boucle qui va être exécuter N fois O(N), à l'intérieur pour chaque itération de la première boucle, cette seconde boucle va être exécuté jusqu'à N*N fois O(N*N). Enfin la troisième boucle va être exécuter jusqu'a N*N fois lors de chaque itération de la seconde boucle O(N*N).

On se retrouve alors avec une complexité de O(N)*O(N*N)*O(N*N) = O(N^5)