جامعة هواري بومدين للعلوم و التكنولوجيا
Université des Sciences et de la Technologie Houari Boumediene
Faculté d'Electronique et d'Informatique
Département d'Informatique

Concours d'accès au Doctorat 3 ième Cycle Informatique 2019– 2020

Le 26/10/2019

Matière 1 : Algorithmique avancée et complexité, Systèmes d'exploitation,
Coefficient **1**, durée **1h30**
(Spécialité : RSI)

Partie 1 : Algorithmique avancée et complexité

Exercice 1 : (8 points)

a. Donnez les définitions des notations Landau Ω , Θ et O.

Réponse :

$O(g(n)) = \{ f : IN \rightarrow IN \mid \exists\ c>0 \text{ et } n_0 \geq 0 \text{ tels que } \mathbf{0 \leq f(n) \leq c.g(n)} \quad \forall\ n \geq n_0 \}$
$\Omega(g(n)) = \{ f : IN \rightarrow IN \mid \exists\ c>0 \text{ et } n_0 \geq 0 \text{ tels que } \mathbf{0 \leq c.g(n) \leq f(n)} \quad \forall\ n \geq n_0 \}$
$\Theta(g(n)) = \{ f : IN \rightarrow IN \mid \exists\ c_1>0, c_2>0 \text{ et } n_0 \geq 0 \text{ tels que } \mathbf{0 \leq c_1.g(n) \leq f(n) \leq c_2.g(n)} \quad \forall\ n \geq n_0 \}$

b. Les affirmations ci-dessous sont-elles vraies ou fausses ? Si vous pensez qu'une affirmation est fausse, indiquez pourquoi et corrigez l'affirmation. (Reproduire, sur la copie, le tableau ci-dessous et le compléter).

Affirmation	Vraie ou Fausse ?	Affirmation correcte
Si $f(n) \in O(g(n)),\ alors\ g(n) \in O(f(n))$.	fausse	si $f(n) \in O(g(n))$, $alors\ g(n) \in \Omega(f(n))$.
Si $f(n) \in O(g(n)), alors\ f(n) + g(n) \in O(g(n))$	vrai	
La meilleure borne asymptotique de $O(f(n)) + O(g(n))$ est $O(f(n) + g(n))$	fausse	$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
Si $g = O(f(n))$ et $h = O(g(n))$ alors $h = O(f(n))$	Vrai	
$5n + 8n^2 + 100n^3 \neq O(n^4)$	fausse	$5n + 8n^2 + 100n^3 = O(n^4)$
$100n + \log n = \Theta(n + (logn)^2)$	Vrai	
$n2^n = \Theta(3^n)$	Vrai	

Exercice 2 : (12 points)
Une attaque par déni de service (DOS) est une attaque visant à saturer une application de sorte qu'elle ne puisse plus répondre à de nouvelles requêtes. Typiquement, un pirate essaiera de saturer l'application par une multiplication de requêtes lourdes en temps de calcul. Un exemple classique est celui de "Apache" qui, en 1997, intégrait la fonction suivante :

```
Procédure Proc1(E-S/ T : Tableau[n] de caractère ; E/ n : entier) ;
Var x, y : entier ;
Début
   x←2 ;
   tant que (x<=n) Faire
         Si (T[x-1] = '/') et (T[x] == '/') Alors
               Pour y ←x+1 à n Faire    T[y-1] ← T[y] ;      fait;
               n←n-1 ;
         Sinon x ←x+1 ;
         finsi ;
   fait;
Fin;
```

a. Que fait cette fonction ?

Réponse :
Cette fonction parcourt une chaine de caractères et s'il y a présence de deux slash successifs alors elle supprime le deuxième slash en l'écrasant avec le reste de la chaine.

b. Quelle est sa complexité dans le pire des cas ?

Réponse :
Soit n la taille de la chaine donnée, au pire des cas, la chaine ne sera constituée que par des slashs. Soit *T(n)* la complexité de cette algorithme on vérifie n fois l'expression :

Si (T[x-1] = '/') & (T[x] == '/') Alors …

et la boucle « pour » qui suit l'expression ci-dessus, le nombre d'itérations = n-x. La complexité est donc la suivante :

$$\sum_{i=2}^{n}\sum_{j=i+1}^{n} 1 = \sum_{i=2}^{n}(n-i) = \sum_{i=2}^{n} n - \sum_{i=2}^{n} i = n(n-1) - \frac{n(n+1)}{2} - 1 \sim O(n^2)$$

c. Comment pouvait-on donc faire pour saturer un serveur web Apache ?

Réponse :
On peut faire saturer un serveur web Apache en lui donnant une très grande chaine de caractères ne contenant que des slashs.

d. Le patch proposé pour corriger cette faille de sécurité implémentait no2slash de la façon ci-après.

```
Procédure Proc1(E-S/ T : Tableau[n] de caractère ; n : entier) ;
Var x, y : entier ;
Début
   x ←1 ;    y ←1;
   Tant que (x <=n) Faire
       T[y]←T[x] ;
       Si (T[y] = '/') Alors
              Tant que (x<=n) et (T[x]= '/') Faire
                     x ← x+1;
              Fait ;
       Sinon x ←x+1 ;
       Finsi ;
       y←y+1 ;
   Fait;
   T[y] ←  '\0'  /* caractère de fin de chaine*/
Fin;
```

Quelle est la complexité de cette seconde fonction ?

Réponse :

Le pire cas est quand la chaine ne contient que des '/' donc à chaque itération on supprime un '/'. On a une boucle qui va s'exécuter qu'une seule fois donc la complexité sera :

$$T(n)=\sum_1^n 1 \sim O(n)$$

La faille est-elle corrigée ?

Réponse :

Ce deuxième algorithme est linéaire $O(n)$ donc il est meilleur au premier $O(n^2)$. La faille est corrigée pour un n petit, mais le problème reste posé quand n → ∞.