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,
Coefficient **1**, durée **1 h 30**
(Spécialités : IA, MFA, SIGL)

Exercice 1 : (8 points)

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

Réponse :

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

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

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

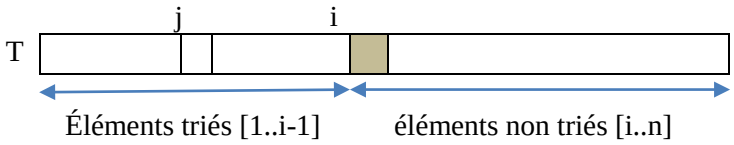
b. Les affirmations ci-dessous sont-elles vraies ou fausses ? Si vous pensez qu'une affirmation est fausse, indiquez pourquoi et corrigez l'affirmation. (Reproduire, sur la copie, le tableau ci-dessous et le compléter).

Affirmation	Vraie ou Fausse ?	Affirmation correcte
Si $f(n) \in O(g(n)),\ alors\ g(n) \in O(f(n))$.	fausse	si $f(n) \in O(g(n))$, $alors\ g(n) \in \Omega(f(n))$.
Soit $S(n) \in O(f(n)), T(n) \in O(g(n))$ Si $f(n) \in O(g(n))$, $alors\ S(n) + T(n) \in O(f(n))$.	fausse	Soit $S(n) \in O(f(n)), T(n) \in O(g(n))$ Si $f(n) \in O(g(n))$, $alors\ S(n) + T(n) \in O(g(n))$.
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	
$\sqrt{n} = O((logn)^2)$	fausse	$\sqrt{n} = \Omega((logn)^2)$

Exercice 2 : (12 points) Tri par insertion

Le tri par insertion d'un tableau T[1..n] de n éléments consiste à insérer chaque élément à sa place. Le premier élément constitue, à lui tout seul, une suite triée de longueur 1. On range ensuite le second élément pour constituer une suite triée de longueur 2, puis on range le troisième élément pour avoir une suite triée de longueur 3 et ainsi de suite...

Le principe du tri par insertion est donc d'insérer à la i-ème itération le i-ème élément à la bonne place. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion.



1. Ecrire l'algorithme « Tri_insertion1 » du tri par insertion tel que décrit précédemment. Donner sa complexité en nombre de comparaisons.

Réponse :

```
Procédure tri_insertion1(E/T : tableau[n] d'entiers, entier n)
var i,j, x :entier ;
Début
    pour i ← 2 à n  faire
            x ← T[i] ;
            j ← i ;
            tantque (j > 0 et T[j - 1] > x) faire T[j] ← T[j - 1] ; j ← j – 1 ; fait ;
            T[j] ← x ;
    fait;
Fin;
```

Ou bien

```
Procédure tri_insertion1(E/T : tableau[n] d'entiers, entier n)
var i,j, x :entier ;
Début
     pour i ← 2 à n faire
            j←1 ;
            tantque (j<i et T[i]>T[j]) faire  j←j+1 ; fait ;
            k←i ;
            x←T[i] ;
            tantque (k > j) faire T[k] ← T[k-1] ; k←k-1 ; fait ;
            T[j] ← x ;
     fait;
Fin;
```

Complexité de l'ordre de $O(n^2)$ car au pire cas pour i de 2 à n on fait i-1 comparaisons ce qui donne $\sum_{i=2}^{n}(i-1) = 1 + 2 + \cdots + n - 1 = \frac{n(n-1)}{2} \sim O(n^2)$

2. Puisque la suite d'éléments, dans laquelle on cherche le rang d'insertion, est triée (T[1..i-1]), on propose d'utiliser la recherche dichotomique.
    a. Donner l'algorithme « Tri_insertion2 ».

```
Fonction Dichotomie (E/ T :tableau ; i :entier) :entier ;
Var d, f, m :entier ;
début
        d←1 ; f←i-1 ;
        tantque (d≤f) faire
                m←(d+f)/2 ;
                si (T[i]>T[m]) alors d←m+1
                sinon si (T[i]< T[m]) f←m-1
                        sinon retourner m ;
                fsi
        fait ;
        retourner m ;
fin ;
```

```
Procédure tri_insertion2(E/T : tableau[n] d'entiers, entier n)
var i,j, x :entier ;
Début
    pour i ← 2 à n faire
            j←Dichotomie(T, i-1) ;
            k←i ;
            x←T[i] ;
            tantque (k > j) faire T[k] ← T[k-1] ; k←k-1 ; fait ;
            T[j] ← x ;
    fait;
Fin;
```

b. Donner sa complexité en nombre de comparaisons.

Réponse : A chaque étape du tri, le nombre de comparaisons est égal à $\lfloor \log_2(i-1) \rfloor + 2$, soit dans tous les cas au total $\sum_{i=2}^{n} \lfloor \log_2(i-1) \rfloor + 2n - 2$ comparaisons. La complexité est égal à $O(n \log_2 n)$. Toutefois, cette méthode perd beaucoup de son intérêt puisque l'insertion nécessite toujours un décalage linéaire. La complexité de ce tri reste donc $O(n^2)$.

c. La complexité du tri a-t-elle été améliorée ?

Cette variante est à peine plus efficace que le tri par insertion séquentielle. Donc pas d'amélioration.

3. Soit l'algorithme récursif suivant :

```
Procédure Tri_insertion3(E/ T : tableau[n] entier ; p : entier)
Var k : entier ;
Début
        Si (p>0) alors
                Tri_insertion3(T, p-1) ; k←p ;
                tantque (k>1 et T[k-1]>T[k]) faire permuter(T[k-1], T[k]) ;  k←k-1 ;  fait ;
        fsi ;
Fin ;
```

L'appel initial se fait par Tri_insertion3(T, n).

a. Dérouler l'algorithme avec n=8 et T=[6| 3| 7| 4| 2| 8| 1| 5]

Réponse :

6	3	7	4	2	8	1	5
3	6	7	4	2	8	1	5
3	6	7	4	2	8	1	5
3	6	4	7	2	8	1	5
3	4	6	7	2	8	1	5
3	4	6	2	7	8	1	5
3	4	2	6	7	8	1	5
3	2	4	6	7	8	1	5
2	3	4	6	7	8	1	5
2	3	4	6	7	1	8	5
2	3	4	6	1	7	8	5
2	3	4	1	6	7	8	5
2	3	1	4	6	7	8	5
2	1	3	4	6	7	8	5
1	2	3	4	6	7	8	5
1	2	3	4	6	7	5	8
1	2	3	4	6	5	7	8
1	2	3	4	5	6	7	8

b. Donner l'équation de récurrence qui décrit le temps d'exécution de l'algorithme. Résoudre l'équation.

Réponse :

L'équation de récurrence : $\begin{cases} T(0) = 0 \\ T(n) = T(n-1) + (n-1) \end{cases}$

$$T(n) = T(n-2) + (n-2) + (n-1)$$

$$T(n) = T(n-3) + (n-3) + (n-2) + (n-1)$$

$$\dots$$

$$T(n) = T(n-n) + (n-n) + (n-(n-1)) + \cdots + (n-2) + (n-1)$$

$$T(n) = 0 + 1 + 2 + \cdots + (n-2) + (n-1)$$

$$T(n) = \sum_{k=0}^{n-1} k = \frac{n(n-1)}{2} \sim O(n^2)$$

4. Que constatez-vous (comparer les trois algorithmes) ?

Réponse : Tous les tris ont même complexité mais on va préférer Tri_insertion2 car ***le nombre de comparaison*** est un peu meilleur que les 2 autres.