UNIVERSITE IBN TOFAIL
Faculté des sciences
Département d'Informatique
Kenitra

Année: 2018/2019
Filières : SMI
Semestre : 3

Algorithmique II
Examen de rattrapage
Durée : 1h 30mn

Exercice 1 : (Sur 6 points)

On considère un tableau d'entiers T[1..n].

Ecrire une procédure ***PlusLongSousTabCroissant (T : Entier[1..n])*** **qui détermine le premier sous-tableau de T de la plus grande taille constitué d'une suite d'entiers croissante. La procédure affichera les indices du début et de fin de ce sous-tableau de T. Autrement dit, si i et j sont respectivement les indices du début et de fin du sous-tableau trouvé alors : T[i]<=T[i+1]<=…<=T[j]. De plus le tableau T ne contient pas de suite croissante de taille strictement supérieur à (j - i+1).**

Corrigé

```
Procédure PlusLongSousTabCroissant(T : Entier[1..n])
Var     i, j, k, deb : Entier
Début
        deb←1          //Indice du début du sous-tableau croissant
        i←1
        k←1            //Taille du sous tableau croissant
        Tant que (i <=n-2) Faire
                j←i
                Tant que (j < n Et T[j]<=T[j+1]) Faire
                        j←j+1
                Fin Tant que
                Si (k<j-i+1) Alors
                        k←j-i+1
                        deb←i
                Fin Si
                i←i+1
        Fin Tant que
        Ecrire("Le début du sous-tableau est ", deb, " et sa fin est ", deb+k-1)
Fin
```

Exercice 2 : (Sur 7 points)

On considère A[1..m, 1..m] un tableau d'entiers de dimension (m,m). Soit la fonction Mystere(A : Entier[1..m, 1..m]) donnée par

Fonction Mystere(A : Entier[1..m, 1..m]) : Booleen

```
Var     i, j, k : Entier
Début
        Pour ( i ← 1 à m-1) Faire
                Pour  (j←i+1 à m) Faire
                        k ← 1
                        Tant que (k<=m Et T[i, k] = T[j, k]) Faire
                                Si (k = m) Alors
                                        Retourner VRAI
                                Sinon
                                        k ←k + 1
                                Fin Si
                        Fin Tant que
                Fin Pour
        Fin Pour
        Retourner FAUX
Fin
```

1. **Quel est le but de la fonction Mystere ?**
2. **Calculer C1(m) la complexité temporelle dans les pires des cas de la fonction Mystere.**
3. **Calculer C2(m) la complexité temporelle dans les meilleurs des cas de la fonction Mystere.**

Corrigé

1. ***But de la fonction Mystere***

   Il est clair que la fonction Mystere retourne "VRAI" s'il existe deux lignes du tableau T qui sont identiques, sinon elle retourne "FAUX".

2. ***Complexité temporelle C1(m) dans les pires des cas de la fonction Mystere***

   Dans les pires des cas on passe (m-1) fois dans la boucle externe 'Pour'. A chaque passage i dans cette boucle on passe (m-i) fois dans la boucle interne 'Pour'. Et pour chaque passage dans la boucle interne 'Pour' on passe (m-1) fois dans la boucle 'Tant que'. Donc,

   $C1(m) = \sum_{i=1}^{m-1}[\sum_{j=i+1}^{m}(taffect + (m-1) * [2tcomp + taffect + taddi] + tcomp)] + tretour$

   Donc $C1(m)=\Theta(m^3)$

3. ***Complexité temporelle C2(m) dans les meilleurs des cas de la fonction Mystere***

   Dans les meilleurs des cas on passe une seule fois dans la boucle externe 'Pour', une seule fois dans la boucle interne 'Pour' et m fois dans la boucle 'Tant que'.

   Donc, C2(m) = taffect+ m*(2tcomp + taffect + taddi) + tcomp+ tretour

   Donc $C2(m)=\Theta(m)$

Exercice 3 : (Sur 7 points)

Le trie stupide est l'algorithme de trie le plus simple à comprendre et à programmer. Il fonctionne de la manière suivante :

Soit Tab[1..n] un tableau d'entiers à trier par ordre croissant en utilisant cet algorithme.

On procède de la manière suivante :

a. **Parcourir le tableau Tab du début jusqu'à rencontrer deux éléments successives qui ne sont pas dans l'ordre.**

b. **Permuter les deux éléments trouvés.**
c. **Si les deux éléments trouvés ne sont pas à la fin du tableau, aller vers l'étape 1.**
d. **Sinon arrêter le processus, le tableau est trié**

1. **Ecrire en pseudo code l'algorithme ci-dessus.**
2. **L'ordre de grandeur de la complexité temporelle <u>dans les meilleurs des cas</u> de cet algorithme est $\theta(n)$. Comment sont les éléments du tableau Tab dans ce cas-là ?**
3. **L'ordre de grandeur de la complexité temporelle <u>dans les pires des cas</u> de cet algorithme est $\theta(n^3)$. Comment sont les éléments du tableau Tab dans ce cas-là ?**

Corrigé

1.

```
Procedure TrieStupide( Tab : Entier[1..n])
Var     i, x : Entier
Debut
        i←1
        Tant que (i <= n-1) Faire
                Si (Tab[i] <= Tab[i+1]) Alors
                        i←i+1
                Sinon
                        x←Tab[i]
                        Tab[i]←Tab[i+1]
                        Tab[i+1]←x
                        i←1
                Fin Si
        Fin Tant que
Fin
```

2. **Complexité dans le meilleur des cas : Si le tableau Tab est trié par ordre croissant alors on passe seulement (n-1) fois dans la boucle 'Tant que', sans faire de permutations, d'où la complexité temporelle dans ce cas-là est de l'ordre de $\theta(n)$.**
3. **Complexité dans le pire des cas : Si le tableau Tab est trié par ordre strictement décroissant alors on passe dans la boucle 'Tant que' de l'ordre de n^3 fois. D'où la complexité temporelle dans ce cas-là est de l'ordre de $\theta(n^3)$.**