Poly technique SOUSSE EUR-ACE®	**Algorithmique Avancée et Langage C** **Classe :** 3ème Année Ingénieur Génie Informatique **Enseignant :** F. HAMDAOUI & S. Ben Ahmed

TD N°2

Exercice 1

Vous disposez d'un tableau T de taille n. Etudiez la complexité en temps des fonctions récursives suivantes sur les tableaux :

1. Tri à bulles
2. Tri par sélection
3. Tri rapide

Idée :

On partage la liste à trier en deux sous-listes telles que tous les éléments de la première soient plus petits que les éléments de la seconde. Par récurrence, on trie les deux sous-listes.
Comment partager la liste en deux?
On choisit une des clés de la liste, et on l'utilise comme pivot.
La liste d'origine est donc coupée en trois : une première liste (à gauche) composée des éléments ≤ au pivot, le pivot (à sa place définitive) et une liste (à droite) composée des éléments > au pivot.
Notre choix : le pivot est le premier élément de la liste.
Supposons qu'on a une procédure PARTITION(A,i,j,k) qui effectue la partition de la sous-liste de A entre les indices i et j, en fonction du pivot A[i], et rend comme résultat l'indice k, où ce pivot a été placé.
Avec cette procédure, on peut écrire une procédure TRI-RAPIDE.

3.1. Au pire des cas
3.2. Au meilleur des cas
3.3. Au moyenne des cas

Exercice 2

Soient (a1, . . . ,ap) et (b1, . . . ,bp) deux suites d'entiers dont les éléments sont triés par ordre croissant. Soit T = (a_1,b_1,a_2,b_2, . . . ,a_p,b_p).
Démontrer, par récurrence sur p, que le tri par insertion de T nécessite, dans le pire des cas, seulement $p(p+1)/2$ échanges.

Exercice 3

Considérons la suite récurrente

$$u_0 = 2 \text{ et } u_{n+1} = \frac{1}{2}\left(u_n + \frac{1}{u_n}\right)$$

Calculons la complexité (en terme d'opérations arithmétiques) de deux implémentations récursives différentes de cette suite :

```
1 def u1(n):
2     if n == 0:
3         return 2
4     return 0.5 * (u1(n - 1) + 1 / u1(n - 1))
```

```
1 def u2(n):
2     if n == 0:
3         return 2
4     x=u2(n-1)
5     return 0.5*(x+1/x)
```

Exercice 4

Soient les propriétés suivantes :

a) Pour tous entiers a, b,
$$\text{pgcd}(a,b) = \text{pgcd}(b,a)$$
b) Pour tout entier a,
$$\text{pgcd}(a,0) = a$$
c) Pour tous entiers a, b :
$$\text{pgcd}(a,b) = \text{pgcd}(b, a-b).$$
d) Pour tous entiers a, b, si r est le reste de la division euclidienne de a par b, alors :
$$\text{pgcd}(a,b) = \text{pgcd}(b,r).$$

1. Ecrire une fonction récursive pgcd(a,b) qui retourne le PGCD des entiers a et b.
2. Etudier sa complexité

Exercice 5

Considérons l'algorithme suivant :

```
1 def myst(n):
2     if n < 2:
3         return 1
4     return myst(n-1) + myst(n-2)
```

1. Que calcule l'algorithme suivant ?
2. Calculer sa complexité en fonction de n en prenant en compte les additions. Que dire de cette algorithme ?
3. Donner un algorithme permettant d'obtenir le même résultat mais avec une meilleure complexité.

Exercice 6

Soit la fonction récursive qui calcule la somme des n éléments d'un tableau A.

fonction somme_rec (A : tableau[1..n] ; n : entier) : entier

si n ≤ 0 **alors**

retourner (0)

sinon

retourner (somme_rec(A, n-1) + A[n])

fin si

Calculer la complexité temporelle et spatiale de cette fonction.

Exercice 7

Soit la fonction

Pour s'entraîner sur le théorème général

Trouver le comportement asymptotique des fonctions suivantes :

1. c(n) = 9c(n/3) + n
2. c(n) = c(2n/3) + 1
3. c(n) = 3c(n/4) + n $\log_2$ n

Exercice 8 Tours de Hanoi

1. Ecrire un algorithme récursif pour implémenter le jeu des tours de Hanoi qui a pour but de déplacer la tour complète de la première tige vers une des deux autres tiges.

Contraintes :

- On ne peut déplacer qu'un seul disque à la fois.
- Un disque ne peut jamais être déposé sur un disque de diamètre inférieur.

Indication:

Soit n le nombre de disques à déplacer. Si n=1 la solution est triviale.
Si on sait transférer n-1 disques alors on sait en transférer n.
Il suffit de transférer les n-1 disques supérieurs de la tours T1 vers la tours T3, de déplacer le disque le plus grand de T1 vers T2, puis de transférer les n-1 disques de T3 vers T2.

2. Etudier la complexité de cet algorithme