SCMO2IN01 : Complexité des algorithmes — Université d'Orléans
2008-2009 — Master MIAGE 1

Juin 2009

1) QCM (8 points)

a) La recherche par dichotomie dans un tableau trié de taille n se fait en ... étapes.

1. $\log(n)$ 2. n 3. n^2 4. 2^n

b) L'addition de deux nombres, x et y, de même taille t, se fait en ... étapes (indiquer *toutes* les bonnes réponses).

1. $\log(t)$ 2. t 3. x 4. $\log(x)$

c) Trier un tableau de n valeurs en utilisant un tas se fait en ... étapes.

1. $\log(n)$ 2. n 3. $n\log(n)$ 4. n^2

d) Classer par ordre croissant de croissance les fonctions suivantes :

1. $\log^2(n)$, 2. n, 3. $n\log(n)$, et 4. n^2.

2) Évaluation d'une fonction polynôme en un point (6 points)

On s'intéresse à l'évaluation d'une fonction polynôme P de degré n en un point x_0. On suppose P représenté par le tableau $a[0..n]$ de ses coefficients ; il s'agit donc de calculer la somme

$$a_0 + a_1x_0 + \ldots + a_nx_0^n \ .$$

a) Écrire un algorithme très simple, et calculer sa complexité, en prenant comme opération élémentaire les opérations arithmétiques (addition et multiplication).

b) Proposer un algorithme pour calculer la puissance n^{e} d'un nombre x (c.-à-d. x^n) en ne faisant que de l'ordre de $\log(n)$ multiplications.

N.B. Si l'on utilise cette mise à la puissance dans la question précédente, on obtient un algorithme en $n\log(n)$.

c) Écrire le polynôme sous la forme

$$\lambda_0 + x_0(\lambda_1 + x_0(\lambda_2 + x_0(\lambda_3 + \ldots x_0(\lambda_n)\ldots))) \ .$$

En utilisant cette factorisation, montrer que l'évaluation en un point d'une fonction polynomiale peut être réalisée en n additions et n multiplications. Écrire l'algorithme correspondant (c'est l'algorithme de Horner).

T.S.V.P.

3) Structure d'ensemble (6 points)

On veut réaliser un objet implantant une structure d'ensemble pour n'importe quel type d'éléments (on ne peut donc supposer l'existence d'un comparateur entre éléments). Les éléments sont stockés dans un tableau que l'on suppose toujours de taille suffisante (sa taille ne varie donc pas). Chaque instance (de cette classe ensemble) possède en plus un attribut entier, *nombre_éléments*, enregistrant le nombre d'éléments. Les ensembles sont créés vides — et donc avec *nombre_éléments* à 0.

a) Les éléments sont uniquement manipulés à l'aide de références (l'algorithme ne manipule donc que des références). Le seul test dont on dispose est l'égalité au sens des références (noté, p.e. `==`).

Quelles sont les incidences en termes de manipulations, de risques d'effets de bord et de complexités de ceci ?

b) Pour chacune des méthodes suivantes, proposer un algorithme et donner la complexité en fonction de n le nombre d'éléments présents dans le tableau. Si la complexité n'est pas toujours la même — à n fixé — la donner dans le pire et le meilleur cas. (Les réponses doivent bien entendu être argumentées.)

i) *donner_cardinalité () : entier* qui retourne le nombre d'éléments présents dans l'ensemble.

ii) *est_présent (élément e) : booléen* qui dit si l'élément est présent ou non dans l'ensemble.

iii) *ajouter (élément e)* qui ajoute un élément à l'ensemble (rien n'est fait s'il s'y trouve déjà).

iv) *enlever (élément e)* qui enlève l'élément `e` de l'ensemble — s'il est présent.

c) On considère qu'un élément a une probabilité p d'être présent, et que s'il l'est, il a la même probabilité d'être dans chacune case du tableau (*i.e.* probabilité uniforme).

Quelle est alors la complexité en moyenne de la méthode *est_présent* ?