Université Paris 7 - Licence d'informatique - L2

Eléments d'Algorithmique

Examen du 26 Mai 2009 - durée : 2h30

Il va sans dire que la rigueur des raisonnements, la clarté des explications, mais aussi la qualité de la présentation influera sensiblement sur la note.

Le barème n'est donné qu'à titre indicatif et pourra donc être modifié.

Exercice 1 - Connaissance des algorithmes de tri - Invariants (6 points)

On rappelle ici les trois algorithmes de tri simples vus en cours. Dans les commentaires on distingue les points de contrôles $S1, S2, S3, S4, B1, B2, B3, B4, I1, I2, I3, I4$

```
public static void triselection(int[] tab,int l,int r){
        for(int i=l;i<=r;i++){
            int min=i;
            // S1
            for(int j=i+1;j<=r;j++){
                // S2
                if(tab[j]<tab[min])min=j;
                // S3
            }
            int tmp=tab[i];
            tab[i]=tab[min];
            tab[min]=tmp;
            // S4
        }
    }
public static void tribulle(int[]t, int l,int r){
        for(int i=l;i<=r;i++)
         // B1
            for(int j=r;j>i;j--){
             //B2
                if(t[j]<t[j-1])echange(t,j-1,j);
                //B3
            }
            //B4
    }
public static void triinsertion(int t[],int l,int r){
        int i;
        for(i=l;i<=r;i++)
         // I1
            for(int j=i;j>l;j--){
```

```
            // I2
               if(t[j]<t[j-1])echange(t,j-1,j);
               // I3
            }
            I4
    }
```

On cherche à savoir, pour chacun des invariants suivants, s'il est ou non vérifié en un point de contrôle donné. Vous répondrez à cette question sous forme d'un tableau, où les lignes sont les invariants et les colonnes les points de contrôles. Vous mettrez sur chaque case si oui ou non l'invariant est vérifié.
Cet exercice sera noté comme un QCM, c'est à dire qu'une mauvaise réponse sera comptée négativement, une bonne réponse positivement, et l'absence de réponse n'aportera pas de points, ni n'en enlèvera.

- Inv 1 - t est trié sur l'intervalle $[l, i[$
- Inv 2 - t est trié sur l'intervalle $[l, i]$
- Inv 3 - t est trié sur l'intervalle $[l, i[$ et ne changera plus de valeur (sur cet intervalle)
- Inv 4 - t est trié sur l'intervalle $[l, i]$ et ne changera plus de valeur (sur cet intervalle)
- Inv 5 - $t[j]$ est le plus petit de l'intervalle $[j, r]$
- Inv 6 - $t[j]$ est le plus petit de l'intervalle $[i, j]$
- Inv 7 - $t[j]$ est le plus petit de l'intervalle $[l, j]$
- Inv 8 - $t[j]$ est le plus petit de l'intervalle $[j, i]$
- Inv 9 - t est trié sur $[j, i]$
- Inv 10 - t est trié sur $[i, j]$
- Inv 11 - t est trié sur $[j-1, i]$
- Inv 12 - $t[j]$ est le plus grand de chaque élement de $[i, j]$

Exercice 2 - AVL (X points)

On rappelle que dans un AVL chaque nœud contient un champs *bal* (pour balance) dont la valeur marque la différence de hauteur entre les sous-arbres du noeud.

1. Dessinez la suite des arbres AVL construits par insertions successives des entiers $5, 6, 7, 2, 1, 3, 4$ dans un AVL initialement vide, en expliquant les étapes des insertions. On souhaite voir précisément le résultat de l'insertion dans l'arbre avant les rotations, les valeurs des balances, la suite des rotations qui mène à l'AVL.
2. Dessinez l'arbre AVL complet dont les nœuds sont $1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15$. Dans cet AVL donner le résultat et les étapes intermédiaires de la suppression des valeurs : $5, 6, 7, 1$

Exercice 3 - Algorithme et complexité (6 points)

L'objectif de cet exercice est de donner un algorithme efficace pour trouver le i-ème plus petit élément d'un tableau T.

1. Donner un algorithme simple en $O(n \log n)$ qui, sur l'entrée (T, i), renvoie la valeur du i-ème plus petit élément de T. Justifier sa complexité.
2. En s'inspirant de l'algorithme du tri rapide, et en particulier de la procédure qui partitionne le tableau en deux parties, proposer un algorithme récursif pour résoudre le même problème. Analyser sa complexité dans le pire cas.
3. Afin d'avoir une idée de sa complexité en moyenne, on suppose qu'à chaque appel récursif, le tableau est divisé en deux parties de même taille (qui est la situation la plus favorable). Sous cette hypothèse, évaluer la complexité de l'algorithme.