Université Paris 7 - Licence d'informatique - L2

Eléments d'Algorithmique

Corrigé de l'Examen du 26 Mai 2010

Exercice 1 - Tas (3 points).

Donnez une implémentation détaillée de l'opération `delete(T : tas, i : entier)` dans un tas-min binaire implémenté dans le type `tas` présenté en cours, qui retourne un tas-min dans lequel l'élément situé à l'indice i dans le vecteur a été supprimé. Attention aux effets de bord au niveau des feuilles et au problème des noeuds avec un seul fils !

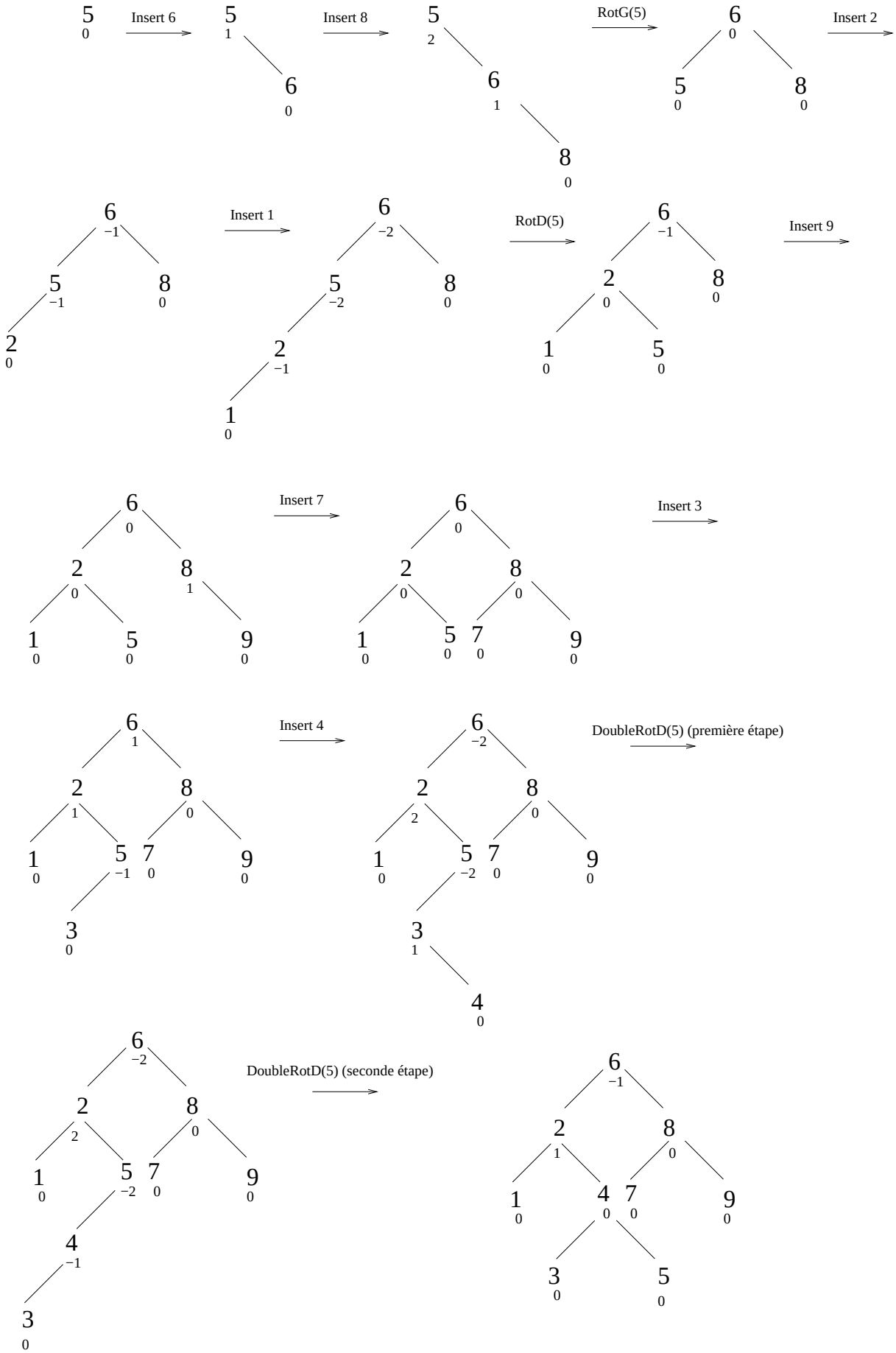
Solution :

```
tas delete(T : tas, i : entier)
  j, k : entier;
  T.element[i] := T.element[T.taille];   \\ on écrase i-ème élément avec le dernier
  T.taille-- ;                           \\on supprime l'ancien dernier élément
  j:=i;                \\il faut faire redescendre l'ancien dernier élément de
                       \\ la place i a sa bonne place. L'index j va le suivre...
  tantque (2j+1 <= T.taille-1) faire      \\tant que j n'est pas une feuille
    si (2j+2 <=  T.taille-1) alors        \\si j a deux fils
      si T.element[2j+1] < T.element[2j+2]\\ on détermine quel est le plus petit
        alors k:=2*j+1                    \\avec lequel il faudra échanger pour rentasser
      else k:=2*j+2
    else k:=2*j+1;                        \\si j a un seul fils, on échange avec lui
    Echanger(T.element, j, k);
    j:=k;                                 \\on met j à jour pour suivre l'élémént
  fin tant que;
  return(T);
fin delete
```

Exercice 2 - AVL (4 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 $5, 6, 8, 2, 1, 9, 7, 3, 4$ 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.


5 0
Insert 6
5 1
6 0
Insert 8
5 2
6 1
8 0
RotG(5)
6 0
5 0
8 0
Insert 2
6 −1
5 −1
8 0
2 0
Insert 1
6 −2
5 −2
8 0
2 −1
1 0
RotD(5)
6 −1
2 0
8 0
1 0
5 0
Insert 9
6 0
2 0
8 1
1 0
5 0
9 0
Insert 7
6 0
2 0
8 0
1 0
5 0
7 0
9 0
Insert 3
6 1
2 1
8 0
1 0
5 −1
7 0
9 0
3 0
Insert 4
6 −2
2 2
8 0
1 0
5 −2
7 0
9 0
3 1
4 0
DoubleRotD(5) (première étape)
6 −2
2 2
8 0
1 0
5 −2
7 0
9 0
4 −1
3 0
DoubleRotD(5) (seconde étape)
6 −1
2 1
8 0
1 0
4 0
7 0
9 0
3 0
5 0

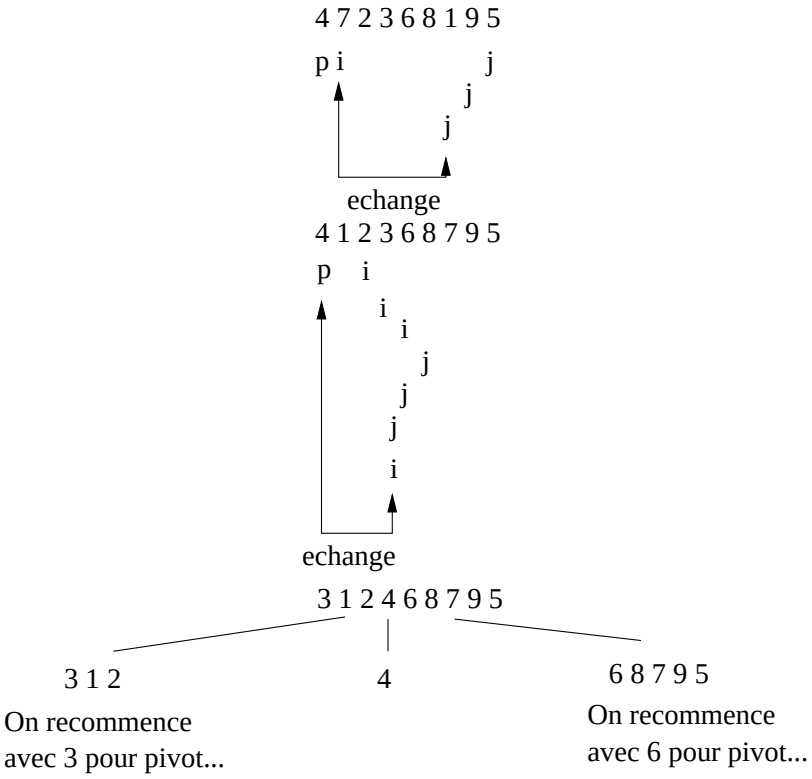
2. Dans cet AVL donner le résultat et les étapes intermédiaires de la suppression des valeurs : 5 et 9.

Les noeuds 5 et 9 sont des feuilles, de plus, leur suppression ne déséquilibre pas l'arbre. Leur suppression est donc immédiate (il suffit de les effacer sans effectuer aucun ré-équilibrage).

Exercice 3 - Tri Rapide (5 points)

Simuler l'algorithme de tri rapide vu en cours pour trier le tableau : $4, 7, 2, 3, 6, 8, 1, 9, 5$. On détaillera chaque application de la procédure `Partition` en sachant qu'elle choisit comme pivot l'élément de trouvant au début de la partie du tableau à partitionner. On pourra représenter avec un graphe l'imbrication des appels récursifs effectués, ainsi que les tableaux qu'ils retournent.
On appliquera la procédure Partition ci-dessous.

```
Partition (T:tableau[0..n-1] d'entiers; d,f: entiers) : int
  i, j : entier;
  pivot := T[d];
  i:=d+1;
  j=f;
  tant que (j>i) faire
    tant que (T[i]  < pivot) faire i++;
    tant que (T[j] > pivot) faire j--;
    si (j>i) alors echanger(T,i,j)
  fin tant que
  si (T[i] > pivot) alors i--;
  echanger(T,i,j);
  return(i);
fin Partition
```



Bien entendu, d'autres variantes de Partition peuvent être considérées, cependant la simulation faite doit être basée sur la procédure proposée. Certains étudiants ont décrit une variante de Partition mais ils en ont ensuite appliqué une autre !!

Problème 4 - Algorithmes et complexité (8 points)

Soit T un tableau de n boules, dont certaines sont bleues et d'autres rouges (le nombre de boules rouges et de boules bleues n'est pas connu et n'est pas déterminant pour la suite du problème).

a) Ecrire une fonction itérative (en pseudo-code) qui répartit le tableau de manière à ce que toutes les boules bleues soient placées devant les boules rouges. La fonction proposée doit travailler sur place (c'est à dire, sans utiliser de tableau auxiliaire), doit effectuer uniquement des déplacements de boules et doit avoir une complexité en temps en $O(n)$ et en espace $O(1)$. On n'est pas autorisé à effectuer un comptage des boules bleues ou des boules rouges.
On rappelle que pour la complexité en espace, seulement les variables de travail sont prises en compte, alors que les données (ici, le tableau) ne le sont pas.

On applique le même principe que la procédure Partition du Quicksort : un curseur i parcourt le tableau du début vers la fin, un curseur j parcourt le tableau de la fin au début, si T[i]=rouge et T[j]=bleu et $j > i$ alors les boules T[i] et T[j] doivent être échangées.

```
TRI_2_COULEURS (T:tableau[0..n-1] d'entiers)
  i, j : entier;
  i:=0;
  j=n-1;
  tant que (j>i) faire
    tant que (T[i]  = bleu) faire i++;
    tant que (T[j]  = rouge) faire j--;
    si (j>i) alors echanger(T,i,j)
  fin tant que
fin Partition
```

Soit maintenant T un tableau de n boules, dont certaines sont bleues, d'autres blanches et d'autres rouges (le nombre de boules de chaque couleur n'est pas connu et n'est pas déterminant pour la suite du problème).

b) Ecrire une fonction itérative (en pseudo-code) qui répartit le tableau de manière à ce que toutes les boules bleues soient placées devant, suivies des boules blanches, et enfin les boules rouges, pour former le drapeau français. Comme la précédente, cette fonction doit travailler sur place, doit effectuer uniquement des déplacements de boules et doit avoir une complexité en temps en $O(n)$ et en espace $O(1)$. On n'est pas autorisé à effectuer un comptage des boules de chaque couleur, ni à changer la couleur d'une boule.

Suggestion. Utiliser trois indices `blu`, `bla`, `rou`. S'assurer que votre algorithme préserve l'invariant suivant :
- `blu` $\leq$ `bla` $\leq$ `rou`.
- Pour tout i avec $0 \leq i < blu$, on a T[i] = bleu.
- Pour tout i avec $blu \leq i < bla$, on a T[i] = blanc.
- Pour tout i avec $rou \leq i < n$, on a T[i] = rouge.
- Pour tout i avec $bla \leq i < rou$, on a T[i] = indéterminé.

Ceci est illustré par la figure suivante :

On peut, bien sûr, envisager d'autres solutions, mais il est impératif qu'elles respectent les conditions de la question b).
Voici une solution qui suit les idées de la suggestion, elle généralise l'idée de la réponse à la question a) (pour deux couleurs)

```
TRI_DRAPEAU_FRANCAIS (T:tableau[0..n-1] d'entiers)
  blu := 0; bla := 0; rge := N;
  tant que  (bla <= rge) faire
    case (a[bla])
      Bleu: echanger(T, blu, bla); blu++; bla++;
      Blanc: bla++;
      Rouge: echanger(T,bla, rge); rge--;
  fin tant que
fin algo
```

Une autre solution possible (mais dans ce cas l'invariant est différent) consiste à appliquer deux fois l'algo de la question 2. Une première fois en considérant Bleu et Blanc comme la même couleur, ce qui permet de placer toutes les boules rouges à la fin. La deuxième fois on se limitera à la partie initiale du tableau, celle qui contient toutes les boules bleues et blanches et aucune boule rouge, et on applique une seconde fois l'algo de la question a) pour séparer les bleues des blanches.

c) Si l'invariant précédent s'applique à votre algorithme, faites-en la preuve, sinon proposez-en un qui démontre la validité de votre algorithme.
Le corrigé de cette question n'est pas fourni.

d) Si l'incrémentation ou la decrémentation d'un index coute p et l'échange de deux boules coute $3q$, quelle est la fonction de complexité dans le meilleur cas en termes de ces opérations ? Et dans le pire des cas ?
Meilleur cas : toutes boules blanches, seulement n incrémentation : complexité pn.
Pire des cas : toutes boules bleus, $2n$ incrémentations, n échanges : complexité $(2p + 3q)n$

e) Proposez un autre algorithme en $O(n)$ et en espace $O(1)$, si on s'autorise aussi le comptage des boules et le changement de couleur d'une boule.

```
TRI_DRAPEAU_AVEC_COMPTAGE (T:tableau[0..n-1] d'entiers)
  cptblu := 0; cptbla := 0;
  Pour i de 0 à n-1 faire
     si (T[i]=Bleu) alors cptblu++;
     sinon si (T[i]=Blanc) alors cptbla++;
  fin Pour
  Pour i de 0 à cptblu-1 faire T[i]:=Bleu;
  Pour i de cptblu à cptblu + cptbla -1 faire T[i]:=Blanc;
  Pour i de cptblu + cptbla à n-1 faire T[i]:=Rouge;
fin algo
```

f) Si l'incrementation d'un compteur coute p et le changement de couleur d'une boule coute q quelle est la fonction de complexité de l'algorithme proposé à la question e) dans le meilleur cas en termes de ces opérations ? Et dans le pire des cas ?

emph Meilleur cas : toutes boules rouges, aucune incrémentation, n coloriages de boules : complexité qn.
Pire des cas : aucune boule rouge, n incrémentations, n coloriages de boules : complexité $(p+q)n$

g) Quelle est la condition sur p et q pour déterminer lequel des deux algorithmes on a intérêt à choisir dans le meilleur cas ? Et dans le pire des cas ?
Dans le meilleur cas : Complexité premier algo : pn, Complexité deuxième algo : qn. Si $p < q$ alors on choisit le premier, sinon on choisit le deuxième.

Dans le meilleur cas : Complexité premier algo : $(2p+3q)n$, Complexité deuxième algo : $(p+q)n$. Puisque p et q sont positifs, on a toujours $(2p+3q)n \geq (p+q)n$. Donc on doit toujours choisir le deuxième.

medskip