Algorithmique TD3 (Corrigé): Tris simples

30 octobre 2008

1. Tri par insertion :
   (a) Réecrire le tri par insertion vu en cours pour trier dans l'ordre décroissant

```
procedure triInsertion(ES tableau tab(N):entier, taille:
    entier)
variables
  i:entier
  j:entier
  cle:entier
  pour j←2 a taille pas 1
    cle ← tab(j)
    i ← j-1
    tant que i>0 et tab(i) < cle faire
      tab(i+1) ← tab(i)
      i ← i-1
    ftq
    tab(i+1) ← cle
  fpour
finprocedure
```

   (b) Qelles sont les meilleurs et pires configurations en terme de coût
   Contrairement au tri sélection par ordre croissant, dont la pire configuration est un tableau trié par ordre décroissant, la pire configuration du tri sélection par ordre décroissant est un tableau trié par ordre croissant.

   Comme pour le tri sélection en ordre croissant, le meilleur cas intervient lorsque le tableau est déjà bien trié (ici ordre décroissant).

   (c) Quel sont ces coûts
   Comme pour le tri sélection par ordre croissant le pire des coûts est en $\Theta(n^2)$ et le meilleur en $\Theta(n)$

   (d) Est-il possible d'améliorer le coût de cet algorithme en appliquant une recherche dichotomique (coût logarithmique, cf TD2) pour l'insertion d'une valeur dans le sous tableau déjà trié.
   Dans le pire des cas cela ne change rien pour le coût car, même si la recherche de l'emplacement ou il faut insérer est plus rapide ($log(n)$ au lieu de n), l'insertion dans un tableau trié nécessite un décalage

qui lui est forcément linéaire. Avec une structure permettant une insertion en temps constant (exemple listes chaînées), on obtiendrait un gain de performances.

2. Tri par sélection :
   On considère le tri suivant de n nombres rangés dans un tableau A : on commence par trouver le plus petit élément de A et on le permute avec $A(1)$. On trouve ensuite le deuxième plus petit élément de A et on le permute avec $A(2)$. On continue de cette manière pour les $n-1$ premiers éléments de A.

   (a) Ecrire l'algorithme de ce tri connu sous le nom de tri par sélection

```
procedure triSelection(ES tableau tab(N):entier, taille:
    entier)
variables
  i:entier
  j:entier
  temp:entier
  pour j←1 a taille-1 pas 1
    imin ← j
    pour i←j+1 a taille pas 1
      si tab(i)<tab(imin) alors
        imin ← i
      fsi
    fpour
    si imin ≠ j alors
      temp ← tab(j)
      tab(j) ← tab(imin
      tab(imin) ← temp
    fsi
  fpour
finprocedure
```

   (b) Quel est l'invariant de boucle de cet algorithme :
   Pour la boucle interne un bon invariant est :
   `{tab(imin) est le minimum de [tab(j),...,tab(i-1)]}`
   - Initialisation : pour $i = j+1$, $imin = j = i-1$
   - Conservation : calcul du nouveau min
   - Terminaison : $i > taille$ donc
     `{tab(imin) est le minimum de [tab(j),...,tab(taille)]}`

   Pour la boucle externe un bon invariant est :
   `{tab(1)<=...<=tab(j-1) et tab(j-1)<=tab(k) pour tout k>=j}`
   - Initialisation : pour $j = 1$, le sous tableau est vide.
   - Conservation : `{tab(1)<=...<=tab(j-1) et tab(j-1)<=tab(k) pour tout k>=j}`.
     D'après l'invariant de la boucle interne,
     `{tab(imin) est le minimum de [tab(j),...,tab(i-1)]}`
     Donc `{tab(1)<=...<=tab(j) et tab(j)<=tab(k) pour tout k>j}`.

   - Terminaison : `{tab(1)<=...<=tab(taille-1) et tab(taille-1)<=tab(taille)}`.
     Donc le tableau est complètement trié.

(c) Pourquoi suffit-il d'exécuter l'algorithme pour les $n-1$ premiers éléments
Le tableau non traité ne contient plus qu'un élément qui d'après l'invariant est supérieur à tous les éléments déjà traités. Cela ne sert donc à rien de le traiter.

(d) Donner les temps d'exécutions du cas moyen et du cas le plus défavorable, exprimés avec la notation Θ
Meilleur des cas : $\Theta(n^2)$
Pire des cas : $\Theta(n^2)$
Donc, cas moyen : $\Theta(n^2)$
Dans tous les cas la recherche du minimum nécessite de parcourir tout le tableau non trié.

3. Tri à bulles :
   Le tri à bulles est un algorithme de tri très populaire. Il consiste à effectuer des permutations répétées d'éléments contigus qui ne sont pas dans le bon ordre, jusqu'à ce qu'il n'y en ai plus.

   (a) Ecrire l'algorithme du tri à bulle en se basant sur la trace d'exécution suivante :

5	**2**	4	6	1	3
2	**5**	**4**	6	1	3
2	4	5	**6**	**1**	3
2	4	5	1	**6**	**3**
2	4	5	1	3	6
2	4	**5**	**1**	3	6
2	4	1	**5**	**3**	6
2	4	1	3	5	6
2	**4**	**1**	3	5	6
2	1	**4**	**3**	5	6
2	1	3	4	5	6
2	**1**	3	4	5	6
1	2	3	4	5	6

```
procedure triBulles(ES tableau tab(N):entier, taille:
    entier)
variables
  i:entier
  j:entier
  temp:entier
  pour j←1 a taille pas 1
    pour i←1 a taille-j pas 1
      si tab(i)>tab(i+1) alors
        temp ← tab(i)
        tab(i) ← tab(i+1)
        tab(i+1) ← temp
      fsi
    fpour
  fpour
finprocedure
```

   (b) Modifier l'algorithme précédent pour qu'il s'arrête dès qu'un passage n'a détecté aucune permutation.

```
procedure triBullesOpt(ES tableau tab(N):entier, taille:
    entier)
variables
  i:entier
  j:entier
  temp:entier
  permut:booleen

  j←1
```

```
  permut ← vrai
  tant que permut et j ≤ taille faire
    permut ← faux
    pour i←1 a taille−j pas 1
      si tab(i)>tab(i+1) alors
        permut ← vrai
        temp ← tab(i)
        tab(i) ← tab(i+1)
        tab(i+1) ← temp
      fsi
    fpour
    j← j+1
  ftq
finprocedure
```