Algorithmique

TD7 – Semaine du 24/11/2003

Éric Colin de Verdière, Yan Jurski, Sylvain Peyronnet

Exercice 1. Algorithme de tri Shell.

Soit un tableau de taille 2^k. On se propose de trier ce tableau en triant *récursivement* les éléments d'indices pairs, puis les éléments d'indices impairs, puis en triant ce que l'on a obtenu par insertion. En d'autres termes, on trie d'abord les indices qui ont le même reste modulo 2^{k-1}, ensuite ceux avec le même reste modulo 2^{k-2}, etc.

1. Appliquer cet algorithme à l'exemple [2 8 7 4 6 3 5 1].
2. Soit $u_1 = 0$, $u_{2p} = \frac{p(p+1)}{2} + 2u_p$. Montrer que u_{2^k} est le nombre d'échanges dans le cas le pire dans la méthode de tri ci-dessus.
3. On se propose de calculer u_{2^k}. On pose pour $k \geq 0$, $v_k = \frac{u_{2^k}}{2^k}$. Calculer v_k pour tout k et conclure.

Exercice 2. Recherche du minimum et du maximum d'un tableau.

1. Appliquer l'approche diviser pour régner au problème de trouver simultanément le minimum et le maximum d'un tableau. Calculer la complexité de l'algorithme en nombre de comparaisons d'éléments du tableau. (On supposera pour ce calcul de complexité que la taille n du tableau est une puissance de 2.) Est-elle est meilleure ou pire que l'approche de la feuille 5 ?
2. On veut montrer qu'il faut faire au moins $3n/2 - 2$ comparaisons pour résoudre ce problème. Pour cela, soit, à une certaine étape de l'algorithme, r le nombre d'éléments dont on peut prouver (par les comparaisons que l'on a faites jusque-là) qu'ils ne sont ni un maximum ni un minimum. Soit m, resp. M, le nombre d'éléments dont on peut prouver qu'ils ne sont pas des maxima (ce sont des minima potentiels), resp. des minima. Soit q le nombre d'éléments dont on ne peut rien dire.

   Soit $c(i)$ la valeur de $3q+2(M+m)$ après la $i^{\text{ème}}$ comparaison entre éléments du tableau. Montrer que $c(0)$ vaut $3n$ et qu'après la dernière comparaison k, $c(k)$ vaut 4. Montrer, en étudiant les différents cas possibles, que dans le pire des cas $c(i) - c(i+1)$ vaut au plus 2. Conclure.

Exercice 3. Recherche d'un élément majoritaire dans un tableau.

Soit T un tableau de taille n. Un élément x de ce tableau est dit *majoritaire* si x apparaît strictement plus que $n/2$ fois dans T.

1. Donner un algorithme « diviser pour régner » qui calcule l'élément majoritaire s'il existe en supposant **qu'on ne peut que** tester l'égalité entre deux éléments. Analyser la complexité de l'algorithme.
2. Montrer que l'algorithme suivant renvoie un élément majoritaire ou -1 s'il n'y en a pas (en Java, le mieux serait de modéliser ce cas par la levée d'une exception). On pourra préalablement tester l'algorithme sur le tableau d'entiers `[3 3 3 4 2 2 2 2 1 1 1 1]`. Quelle est sa complexité ?

```
T : tableau[1..n] d'entiers
i : entier
candidat : entier
```

```
// première boucle : trouver un élément tel qu'aucun autre élément ne
// peut être majoritaire
candidat <- T[1]
compt <- 1
pour i <- 2 à n faire
   si candidat = T[i] alors compt <- compt + 1
   sinon si (compt > 0) alors compt <- compt - 1
                        sinon candidat <- T[i]
                              compt <- 1
         fin si
   fin si
fin pour

// seconde boucle : tester si candidat est majoritaire
compt <- 0
pour i<-1 à n faire
  si T[i]=candidat alors compt <- compt + 1
  fin si
fin pour
si compt>(n div 2) alors retourner candidat
                   sinon retourner -1
fin si
```