TD d'Algo n° 2

EPITA ING1 2013

Les exercices 1 et 2 préparent le TP de cet après-midi.

1 Recherche dichotomique

Du grec *dikha* (en deux) et *tomia* (coupure, tranche).

Entrée : un tableau $A[l..r]$ trié, une valeur v a y chercher.
Sortie : un indice $i \in [\![l, r+1]\!]$ tel que **soit** $A[i] = v$, **soit** $A[i-1] < v$ (si $i > l$) et $v < A[i]$ (si $i \leq r$).

```
BINARYSEARCH(A, l, r, v)
 1  if l ≤ r then
 2      m ← ⌊(l + r)/2⌋
 3      if v = A[m] then
 4          return m
 5      else
 6          if v < A[m] then
 7              return BINARYSEARCH(A, l, m − 1, v)
 8          else
 9              return BINARYSEARCH(A, m + 1, r, v)
10  else
11      return l
```

Attention, cet algorithme indique la position ou v *devrait* se trouver dans A.

1. Justifiez que $\lfloor (l+r)/2 \rfloor = l + \lfloor (r-l)/2 \rfloor$ lorsque l et r sont des entiers naturels.
2. Si l'égalité précédente est correcte mathématiquement, montrez qu'elle n'est plus valable lorsqu'on la transpose dans un langage de bas niveau comme C. Autrement dit, proposez deux `int`, `a` et `b`, tels que `assert((a+b)/2 == (a + (b-a)/2))` signale un problème. Laquelle des deux écritures faut-il utiliser ?
3. **Majorez** $T(n)$, la complexité de la recherche dichotomique en fonction de la taille du tableau $n = r + 1 - l$, à l'aide de $T(n/2)$.
4. En substituant $T(n)$ dans cette formule récursive jusqu'à $T(1)$, donnez un majorant "non-récursif" de $T(n)$. Vous pouvez supposer que n est une puissance de 2.
5. Les appels recursifs à `BinarySearch` sont-ils des appels récursifs terminaux ?
6. Déduisez-en une implémentation non-récursive de `BinarySearch`.
7. Quelle est la complexité de la version non-récursive de `BinarySearch` ?

2 Tri par insertion avec dichotomie

On reprend le tri par insertion vu en cours :

Entrée : un tableau $A[0..n-1]$ contenant n *objets*
Sortie : le tableau A trié par ordre croissant

```
INSERTIONSORT(A, n)
1  for i ← 1 to n − 1 do
2          key ← A[i]
3          j ← i − 1
4          while j ≥ 0 and A[j] > key do
5                  A[j + 1] ← A[j]
6                  j ← j − 1
7          A[j + 1] ← key
```

1. Rappelez le nombre de comparaisons *entre objets* effectuées dans le pire cas et dans le meilleur cas.
2. Calculez le nombre d'affectations d'objets effectués dans ces deux cas.
3. Proposez une variante du tri par insertion qui utilise une recherche dichotomique pour trouver la position d'insertion de *key*, puis décale toutes les valeurs une fois cette position trouvée.
4. Étudiez le nombre de comparaisons et d'affectations d'objets de votre nouvel algorithme, dans le pire cas.

3 La notation Θ

On rappelle la définition :

$$\Theta(g(n)) = \{f(n) \mid \exists c_1 \in \mathbb{R}^{+\star}, \exists c_2 \in \mathbb{R}^{+\star}, \exists n_0 \in \mathbb{N}, \forall n \geq n_0,\ c_1 g(n) \leq f(n) \leq c_2 g(n)\}$$

Par abus de notation, on note souvent $f(n) = \Theta(g(n))$ à la place de $f(n) \in \Theta(g(n))$. De même $n^3 + \Theta(n^2)$ doit s'interpréter comme une formule de la forme « $n^3 + f(n)$ » avec $f(n) \in \Theta(n^2)$.

1. Étant données trois fonctions positives f, g et h, prouvez rigoureusement les propriétés suivantes :
   (a) $\Theta(g_1(n)) + \Theta(g_2(n)) \subseteq \Theta(g_1(n) + g_2(n))$
   (b) $\Theta(g_1(n)) + \Theta(g_2(n)) \subseteq \Theta(\max(g_1(n), g_2(n)))$
   (c) $\Theta(g_1(n)).\Theta(g_2(n)) \subseteq \Theta(g_1(n).g_2(n))$
2. Prouvez que les propriétés suivantes sont fausses :
   (a) $g(n) = \Theta(g(n/2))$
   (b) $g_1(n) = \Theta(g_2(n)) \iff \log(g_1(n)) = \Theta(\log(g_2(n)))$

4 Recherche Trichotomique

Proposez une implémentation de *recherche trichotomique* et étudiez sa complexité.