TD d'Algo n° 5

EPITA ING1 2013

1 Frankenstein joue avec QUICKSORT

On considère une variante de QUICKSORT où le pivot choisi par la procédure de partition est la médiane du sous-tableau. Baptisons cette variante QUICKSORTM, le M signifiant au choix Médiane ou Monstrueux.

Entrée : un tableau $A[l..r]$ d'entiers
Sortie : le tableau A trié par ordre croissant

```
QUICKSORTM(A, l, r)
1  if l < r then
2      p ← PARTITIONM(A, l, r)
3      QUICKSORTM(A, l, p)
4      QUICKSORTM(A, p + 1, r)
```

Entrée : un tableau $A[l..r]$ d'entiers
Sortie : un indice p, et le tableau A réordonné tel que $A[l..p] \leq A[p+1..r]$

```
PARTITIONM(A, l, r)
1  x ← A[MEDIAN(A, l, r)]
2  i ← l − 1 ; j ← r + 1
3  repeat forever
4      do i ← i + 1 until A[i] ≥ x
5      do j ← j − 1 until A[j] ≤ x
6      if i < j then
7          A[i] ↔ A[j]
8      else
9          return j − (j = r)
```

On suppose que MEDIAN(A, l, r) retourne l'*indice* de la médiane du tableau $A[l..r]$, en ayant le droit de modifier l'ordre des valeurs de $A[l..r]$. Pour un l et r donnés, on note $n = r + 1 - l$ la taille du tableau considéré.

1. Justifiez que si MEDIAN est basé sur des comparaisons, il en utilise au moins $n - 1$, quelle que soit son implémentation.
2. Expliquez (en deux lignes), comment implémenter l'algorithme MEDIAN en $\Theta(n \log n)$ comparaisons.
3. Justifiez que l'algorithme MEDIAN que voici retourne effectivement l'indice de la médiane de $A[l..r]$:

```
MEDIAN(A, l, r)
1  m ← ⌊(l + r)/2⌋
2  for i ← l to m do
3          min ← i
4          for j ← i + 1 to r do
5                  if A[j] < A[min] then min ← j
6          A[i] ↔ A[min]
7  return m
```

4. Donnez, en fonction de n, la complexité de l'algorithme ci-dessus.
5. Déduisez-en, en fonction de n, la complexité de PARTITIONM utilisant l'algorithme MEDIAN ci-dessus.
6. Donnez une équation récursive (de la forme $T(n) = aT(n/b) + f(n)$) de la complexité de l'algorithme QUICKSORTM utilisant les sous-procédures décrites ici.
7. À l'aide du théorème général, déduisez-en la complexité de QUICKSORTM. (Mérite-t-il son nom ?)
8. On modifie l'algorithme principal comme suit, sans changer la définition de MEDIAN :

```
QUICKSORTM2(A, l, r)
1  if l < r then
2      p ← MEDIAN(A, l, r)
3      QUICKSORTM2(A, p + 1, r)
```

   Expliquez pourquoi cela est toujours correct, c'est-à-dire pourquoi le tableau retourné par QUICKSORTM2 sera trié alors que nous n'avons pas fait de recursion sur la première moitié, et que nous n'appelons même plus PARTITIONM.
9. Calculez la complexité de QUICKSORTM2.

10. La recursion de QUICKSORTM2 étant terminale, montrez comment réécrire QUICKSORTM2 en appelant toujours MEDIAN, mais sans faire d'appel récursif à QUICKSORTM2. Vous pouvez-même y *inliner* MEDIAN, et renommer QUICKSORTM2 avec son vrai nom.

2 SELECTION inspirée de QUICKSORT

On pourrait imaginer cet exercice comme l'inverse du précédant : au lieu d'utiliser une médiane dans le QUICKSORT, nous alons nous inspirer du QUICKSORT pour calculer une médiane. Ou plus généralement, nous voulons *sélectionner* la valeur de rang i dans un tableau (c'est-à-dire la i^e plus petite valeur). Le minimum a pour rang 1, le maximum a pour rang n, et la médiane a pour rang $n/2$.

1. Proposez un algorithme pour trouver la valeur d'un rang i et donnez sa complexité.
2. Déroulez l'algorithme suivant sur quelques exemples puis expliquez pourquoi il retourne effectivement la valeur de rang i.

Entrée : un tableau $A[l..r]$ d'entiers, un rang i
Sortie : la valeur de rang i

```
SELECTION(A, l, r, i)
1  if l = r then return A[l]
2  m ← RANDOMIZEDPARTITION(A, l, r)
3  k ← m − l + 1
4  if i ≤ k
5       then return SELECTION(A, l, m, i)
6       else return SELECTION(A, m + 1, r, i − k)
```

Entrée : un tableau $A[l..r]$ d'entiers
Sortie : un indice p, et le tableau A réordonné tel que $A[l..p] \leq A[p+1..r]$

```
RANDOMIZEDPARTITION(A, l, r)
1  x ← A[RANDOM(l, r)]
2  i ← l − 1 ; j ← r + 1
3  repeat forever
4       do i ← i + 1 until A[i] ≥ x
5       do j ← j − 1 until A[j] ≤ x
6       if i < j then
7            A[i] ↔ A[j]
8       else
9            return j − (j = r)
```

3. Calculez la complexité de SELECTION en pire, et meilleur cas. Vous supposerez que RANDOM est en $\Theta(1)$.
4. Cet algorithme a une complexité moyenne de $\Theta(n)$ (ce sera démontré en cours).