ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE
Section d'Informatique et de Systèmes de Communication

Corrigé de la série 8 22 November 2010

1. Sorting

a) Considérons l'algorithme suivant:

Input: Une suite de 3 éléments $a[0], a[1], a[2]$.
Output: La permutation (p_0, p_1, p_2) de $(0, 1, 2)$ telle que $a[p_0] \leq a[p_1] \leq a[p_2]$.

```
1:  if a[0].key < a[1].key then
2:    if a[1].key < a[2].key then
3:      return (0, 1, 2)
4:    else
5:      if a[2].key < a[0].key then
6:        return (2, 0, 1)
7:      else
8:        return (0, 2, 1)
9:  else
10:   if a[0].key < a[2].key then
11:     return (1, 0, 2)
12:   else
13:     if a[2].key < a[1].key then
14:       return (2, 1, 0)
15:     else
16:       return (1, 2, 0)
```

On rappelle qu'il y a $3! = 6$ permutations possibles de 3 éléments, nous avons donc 6 possibilités pour la permutation initiale (chacune ayant la même probabilité $\frac{1}{6}$):

$$\begin{array}{lllll} a[0] & < & a[1] & < & a[2] \\ a[0] & < & a[2] & < & a[1] \\ a[1] & < & a[0] & < & a[2] \\ a[1] & < & a[2] & < & a[0] \\ a[2] & < & a[0] & < & a[1] \\ a[2] & < & a[1] & < & a[0] \end{array}$$

En analysant l'algorithme on voit que si $a[0] < a[1] < a[2]$ ou si $a[1] < a[0] < a[2]$ il faudra 2 comparaisons, et dans les 4 autres cas il en faudra 3. Il faudra donc en moyenne

$$\frac{2+2+3+3+3+3}{6} = \frac{16}{6}$$

comparaisons.

b) On a:

• Au début du premier passage, $a[0].key$ sera la clé la plus grande, cet élément va donc remonter jusqu'à la position $N-1$, (ce qui nécessitera $N-1$ échanges).

• Au début du deuxième passage, $a[0].key$ sera la deuxième plus grande clé, et cet élément va donc remonter jusqu'à la position $N-2$ (ce qui nécessitera $N-2$ échanges).

• De même, au début du $i^{\text{ème}}$ passage, $a[0].key$ sera la $i^{\text{ème}}$ plus grande clé, et cet élément va donc remonter jusqu'à la position $N-i$ (ce qui nécessitera $N-i$ échanges).

Au total il faudra donc

$$\sum_{i=1}^{N-1}(N-i) = \sum_{i=1}^{N-1} i = \frac{N(N-1)}{2}$$

échanges.

c) Si un élément x se trouve à la position j au début d'un passage de BubbleSort (donc $a[j] = x$), après le passage il se trouvera à une position k avec $k \geq j-1$. En effet, lors d'un passage un élément peut "monter" jusqu'à la fin de la suite (i.e. la position $N-1$), mais ne ne peut "descendre" qu'au plus d'une position.

L'élément à la position p_i doit par définition se trouver à la position i à la fin de l'algorithme. Si $p_i > i$, cet élément doit donc "descendre" de $(p_i - i)$ position. Puisqu'il ne peut "descendre" que d'au plus une position par passage, il faudra au moins $(p_i - i)$ passages pour qu'il arrive à sa place.

Ainsi il faut à BubbleSort au moins

$$\max_{0 \leq i \leq N-1} (p_i - i)$$

passages pour trier la suite.

2. Nombre moyen d'échanges avec Selection Sort

a) Voici le tableau d'échanges s'il n'y a que trois éléments:

permuation	nombre d'échanges
012	0
021	1
102	1
120	2
201	2
210	1

En effet, 012 est déjà ordonné donc aucun échange n'est nécessaire. Pour 021 il suffit d'échanger 1 et 2, pour 102 on échange 0 et 1. Pour 120 on échange d'abord 0 et 1, puis 1 et 2, donc deux échanges sont nécessaires. Et ainsi de suite.

Si toutes ces permutations ont la même probabilité, le nombre moyen d'échanges sera

$$\frac{0+1+1+2+2+1}{6} = \frac{7}{6}.$$

b) (i) Il y a $k!$ permutations de $0, \ldots, k-1$ au total (il y a k éléments). Nous voulons compter dans combien d'entre elles 0 est à la bonne place. Pour construire une permutation dans laquelle 0 est à la bonne position, on le place d'abord en premier, puis on peut placer les $k-1$ éléments qui restent de n'importe quelle façon (donc selon n'importe laquelle des $(k-1)!$ permutations possibles). Ainsi la probabilité que 0 est à la bonne place est

$$\frac{(k-1)!}{k!} = \frac{(k-1)\cdots 2\cdot 1}{k\cdot(k-1)\cdots 2\cdot 1} = \frac{1}{k}.$$

(ii) Comme expliqué dans la question, SELECTIONSORT fait $N-1$ itérations pour i allant de 0 à $N-2$. Pendant l'itération i on fait un échange si et seulement si le $i^{\text{ème}}$ élément n'est pas à la bonne place.

Pour la première itération ce sera la cas avec probabilité $1-\frac{1}{N}$ (d'après la question (i), où ici on a N éléments donc $k=N$).

Le nombre moyen d'échanges durant la première itération est donc

$$1\cdot\left(1-\frac{1}{N}\right)+0\cdot\frac{1}{N} = 1-\frac{1}{N} = \frac{N-1}{N}.$$

De même, au début de la $i^{\text{ème}}$ itération on sait que les éléments $0, \ldots, i-1$ sont tous à la bonne position. Il reste donc $N-i$ éléments disposés de façon aléatoire dans la liste. Il faudra faire un échange si et seulement si l'élément i n'est pas à la bonne position, ce qui arrive avec probabilité $1-\frac{1}{N-i}$ (question (i) avec $N-i$ éléments). Comme ci-dessus, le nombre moyen d'échanges pendant la $i^{\text{ème}}$ itération est donc

$$\frac{N-i-1}{N-i}.$$

Finalement, le nombre moyen d'échanges au total est égal à la somme des nombres moyens d'échanges à chaque itération, donc

$$\frac{N-1}{N}+\frac{N-2}{N-1}+\ldots+\frac{2}{3}+\frac{1}{2}.$$

3. Dégénérescence de QuickSort

a) Si la suite est déjà triée, le pivot sera à chaque fois le plus grand élément. La suite sera donc coupée en une suite vide (à droite) et une suite consistant en tous les éléments sauf le pivot (à gauche). Il faudra donc faire $N-1$ étapes qui nécessiteront $1, \ldots, N-1$ comparaisons, donc au total $\Omega(N^2)$.

b) Il s'agit à nouveau de construire des suites de sorte que le pivot soit fréquemment un élément très grand (ou très petit) de la suite. Cette stratégie empêche qu'on puisse arranger la suite de sorte que le pivot soit le plus grand élément de la suite, mais c'est toujours possible que ça soit le deuxième plus grand élément.

Supposons que les clés sont les nombres $0, 1, \ldots, N-1$, et que N est pair. Alors l'arrangement

$$0, *, \cdots, *, (N-1), *, \cdots, *, a, (N-2),$$

(où le "$(N-1)$" se trouve à la position $N/2$, et les "$*$" dénotent n'importe quelle clé), fait que l'algorithme choisit $(N-2)$ comme pivot, puisque c'est la médiane de $(0, N-1, N-2)$. Après le premier pas de partitionnement de QuickSort on aura

$$\underline{0, *, \cdots, *, a, *, \cdots, *}, (N-2), (N-1),$$

où les $*$ n'ont pas changé de position. (Remarquons que QuickSort effectue $N-2$ comparaisons pour arriver à cette étape.)

On veut que le comportement pathologique ci-dessus se répète: pour y arriver, on peut disposer les éléments dans la sous-suite soulignée ci-dessus de manière analogue, i.e., positionner $(N-3)$ au milieu et $(N-4)$ tout à droite. Donc l'arrangement

$$0, *, \cdots, *, N-3, N-1, *, \cdots, *, N-4, *, N-2$$

fait que les deux premières itérations exhiberont un comportement pathologique.

En répétant la construction, on peut s'arranger pour que les premières $\lfloor N/4 \rfloor$ itérations seront de ce genre:

$$0, *, \cdots, N-\frac{2N}{4}-1, \cdots, N-3, N-1, \cdots, *, N-\frac{2N}{4}, *, N-\frac{2N}{4}+2, *, \cdots, N-4, *, N-2.$$

Pour effectuer chacune de ces itérations, $\geq N/2$ comparaisons sont nécessaires, donc au total $\geq N/4 \cdot N/2 = \Omega(N^2)$ comparaisons.

Remarque: Comme il n'était pas précisé que les éléments de la suite doivent être *distincts*, on peut trouver une solution bien plus simple: Si tous les éléments sont égaux ($a[0] = \ldots = a[N-1]$) il faudra $\Omega(N^2)$ comparaisons.

En effet, à chaque passage le pointeur p ne s'arrêtera que lorsqu'il arrivera au début de la suite (la position ℓ dans l'algorithme du cours), alors que q ne s'arrêtera qu'à la position $r-1$. Ainsi la sous-suite de droite sera toujours vide. Il faudra donc faire un passage de QuickSort sur une suite de taille N, puis $N-1$, puis $N-2$, etc... Comme chaque passage nécessite $\Omega(N)$ comparaisons, il faudra $\Omega(N^2)$ comparaisons au total.