TD 4 : LazySelect et QuickSort

Exercice 1. *L'algorithme* LazySelect

On revisite le problème de la sélection : étant donné un tableau T de n entiers (supposés tous distincts pour simplifier) et un entier k, on cherche le $k^{\text{ème}}$ plus petit élément de T, noté $T_{(k)}$. Pour un tableau T et un entier $x \in T$, on note $r_T(x)$ son *rang* dans T, *i.e.* $x = T_{(r_T(x))}$ par définition. On étudie dans cet exercice l'algorithme LazySelect pour ce problème.

LazySelect(T, k)

1. Tirer uniformément, indépendamment et *avec remise* [1] un multi-ensemble S de $n^{3/4}$ éléments de T
2. Trier S, et poser $x = k/n^{1/4}$
3. Trouver l'élément a de rang $\ell = \max(1, \lfloor x - \sqrt{n} \rfloor)$ de S et l'élément b de rang $h = \min(n^{3/4}, \lceil x + \sqrt{n} \rceil)$
4. Calculer les rangs $r_T(a)$ et $r_T(b)$ de a et b dans T
5. Calculer l'ensemble R défini par
   - $\{y \in T : y \leq b\}$ si $k < n^{1/4}$
   - $\{y \in T : y \geq a\}$ si $k > n - n^{3/4}$
   - $\{y \in T : a \leq y \leq b\}$ sinon
6. Si $T_{(k)} \in R$ et $|R| \leq 4n^{3/4} + 2$: en triant R, identifier et renvoyer $T_{(k)}$
7. Sinon : recommencer à l'étape 1.

---

1. En fonction du cas de l'étape 5., comment identifier $T_{(k)}$ dans R ?

2. Montrer que le test $T_{(k)} \in R$ peut s'effectuer facilement, connaissant $r_T(a)$ et $r_T(b)$.

3. Montrer que si l'algorithme termine au premier essai, il effectue $2n + o(n)$ comparaisons.

L'objectif est maintenant de montrer que la probabilité que l'algorithme ne termine pas au premier essai est $\leq 1/n^{1/4}$. Les possibilités d'échec sont de deux types : soit R ne contient pas $T_{(k)}$, soit R contient trop d'éléments.

4. Soit X_i la variable aléatoire qui vaut 1 si le $i^{\text{ème}}$ élément tiré aléatoirement à l'étape 1. est $\leq T_{(k)}$, et 0 sinon. On note $X = \sum_{i=1}^{n^{3/4}} X_i$. *On suppose que* $n^{1/4} \leq k \leq n - n^{3/4}$.

i. Calculer $\Pr[X_i = 1]$ et $\mathbb{E}[X]$, et montrer que $\sigma^2(X) \leq n^{3/4}/4$.
Indications. Les X_i sont des variables indépendantes, et montrer que $\frac{k}{n}(1 - \frac{k}{n}) \leq 1/4$ pour $0 \leq k \leq n$.

ii. En déduire que $\Pr\left[|X - \mathbb{E}[X]| \geq \sqrt{n}\right] \leq 1/(4n^{1/4})$.

iii. En déduire que $\Pr\left[a > T_{(k)}\right] \leq 1/(4n^{1/4})$ et $\Pr\left[b < T_{(k)}\right] \leq 1/(4n^{1/4})$.

5. On veut maintenant borner la probabilité que R soit *trop grand*. Soit $k_\ell = \max(1, k - 2n^{3/4})$ et $k_h = \min(n, k + 2n^{3/4})$.

i. Montrer que $\Pr\left[|R| > 4n^{3/4} + 2\right] \leq \Pr\left[a < T_{(k_\ell)} \vee b > T_{(k_h)}\right]$.

ii. Montrer que $\Pr\left[a < T_{(k_\ell)}\right] \leq 1/(4n^{1/4})$ et $\Pr\left[b > T_{(k_h)}\right] \leq 1/(4n^{1/4})$.

6. En déduire que la probabilité que l'algorithme termine au premier essai est $\geq 1 - 1/n^{1/4}$.

7. Borner l'espérance du nombre total de comparaisons effectuées par l'algorithme.

Le meilleur algorithme déterministe connu pour le problème de la sélection effectue $3n$ comparaisons dans le pire cas. D'autre part, il a été montré que tout algorithme calculant la médiane de n éléments effectue au moins $2n$ comparaisons. L'algorithme LazySelect *surpasse donc les algorithmes déterministes connus, et permet quasiment d'atteindre la borne inférieure.*

1. Ce qui signifie qu'un élément x de T peut être tiré plusieurs fois.

Exercice 2. *Implantation en place de* QUICKSORT

On propose les implantations suivantes en C de l'algorithme de PARTITION commun à QUICKSORT et QUICKSELECT. On suppose que `swap(T, i, j)` échange les éléments d'indices `i` et `j` du tableau `T`. L'objectif des deux algorithmes est le même : étant donné un tableau T et des indices b, h et p vérifiant $0 \leq b \leq p \leq h < |T|$, le sous-tableau $T[b:h]$ (éléments d'indices b à h, inclus) est partitionné en utilisant l'élément $T[p]$ comme pivot. La partition est effectuée *en place*, c'est-à-dire que le sous-tableau $T[b:h]$ contient après exécution les mêmes éléments qu'avant, mais les éléments inférieurs au pivot sont *au début* du sous-tableau, et les autres à la fin. L'entier renvoyé permet de connaître la limite entre les deux sous-parties de $T[b:h]$. *On suppose dans l'exercice que T ne contient que des éléments distincts deux à deux.*

```
int partition_lomuto(int* T, int b, int h, int p) {
    int pivot = T[p];
    swap(T, h, p);
    int i = b-1;
    for(int j = b; j < h; j++)
        if (T[j] <= pivot) swap(T, ++i, j);
    swap(T, ++i, h);
    return i;
}

int partition_hoare(int* T, int b, int h, int p) {
    int pivot = T[p];
    swap(T, b, p);
    int i = b - 1, j = h + 1;
    while(1) {
        do i++; while(T[i] < pivot);
        do j--; while(T[j] > pivot);
        if (i >= j) return j;
        swap(T, i, j);
    }
}
```

1. Montrer la correction de `partition_lomuto` et `partition_hoare`.
2. Analyser la complexité dans le pire cas des algorithmes, en nombre d'échanges (appels à `swap`).
3. On suppose qu'on exécute `partition_lomuto` sur un tableau T, avec $b = 0$, $h = n-1$ (où $n = |T|$) et un indice de pivot p choisi de manière aléatoire uniforme entre 0 et $n-1$. Calculer l'espérance du nombre d'échanges effectués.
4. *(Bonus, à la maison !)* Donner des implantations complètes en place de QUICKSORT utilisant chacun des deux algorithmes et comparer leurs performances en pratique.