Examen de Complexité

Durée : 1h30

*Documents autorisés. Connexion à Internet interdite (smartphones et ordinateurs interdits)
 Justifiez précisément chacune de vos réponses !*

Nous avons étudié pendant les CM, deux algorithmes de tri : le tri par bulles (*BubbleSort*), et le tri-fusion (*MergeSort*).

- Rappelez les grands principes de ces deux algorithmes (sans nécessairement donner leur code ou leur pseudo-code), et explicitez leurs avantages et leurs inconvénients en termes de complexité en temps, et de complexité mémoire.

Nous avons étudié pendant les TD, un autre algorithme de tri : le tri par tas (*HeapSort*).

- Dans quelle mesure peut-on dire que ce troisième algorithme combine les avantages des deux algorithmes précédents, sans en avoir les inconvénients ?

L'objectif de la suite de ce problème est d'étudier en détails, un autre algorithme très célèbre, et peut-être même encore plus utilisé que le tri par tas : l'algorithme du Tri_Rapide (*QuickSort*), et ses variantes. On donne ci-dessous le pseudo-code de cet algorithme. A est un tableau de n nombres entiers :

<pre> Partition(A, p, r) x ← A[r] i ← p pour j de p à r-1 faire si A[j] ≤ x alors Echange(A, i, j) i ← i+1 Echange(A, i, r) Renvoyer i </pre>	<pre> Tri_Rapide_Recursif(A, p, r) si p < r alors q ← Partition(A, p, r) Tri_Rapide_Recursif(A, p, q-1) Tri_Rapide_Recursif(A, q+1, r) Tri_Rapide(A) Tri_Rapide_Recursif(A, 0, Taille(A)-1) </pre>
---	--

- Ecrivez le pseudo-code de la fonction `Echange(A, i, j)`, qui échange le contenu des éléments `i` et `j` du tableau `A`. Quelle est sa complexité ?
- Explicitez le rôle de la fonction `Partition()`. Illustrez chacune des étapes de son fonctionnement sur le tableau `A` représenté ci-dessous, à $n=9$ éléments, lors de l'appel `q ← Partition(A, 0, 8)`. Complétez le contenu du tableau `A` pour chaque valeur de `j`, ligne par ligne, à la fin de l'itération numéro `j`. La case numéro `j` est grisée. Vous indiquerez la valeur de `i` en entourant le chiffre de la case numéro `i`.

	A[0]	A[1]	A[2]	A[3]	A[4]	A[5]	A[6]	A[7]	A[8]
État initial	8	5	10	2	3	4	12	4	6
j=0	8	5	10	2	3	4	12	4	6
j=1	5	8	10	2	3	4	12	4	6
j=2	5	8	10	2	3	4	12	4	6
j=3									
j=4									
j=5									
j=6									
j=7									
Etat final									

5. Dans l'exemple de la question 4, quelle est la valeur de retour de la fonction, qui est affectée à q et quel est le contenu de la case $A[q]$ du tableau ? A la fin de l'exécution de la fonction, que peut-on dire des valeurs des éléments du tableau placés avant l'indice q ? et de ceux placés après l'indice q ? Quels sont les paramètres des deux appels récursifs `Tri_Rapide_Recurusif()` qui suivent l'appel à la fonction $q \leftarrow \text{Partition}(A, 0, 8)$? Justifiez le fait que l'élément $A[q]$ a trouvé sa place définitive dans le tableau, et ne changera plus de position dans la suite de l'algorithme.
6. Quelle est la complexité, en fonction de n , de l'appel de la fonction $q \leftarrow \text{Partition}(A, 0, n-1)$?

D'un point de vue de la complexité en temps, le scénario idéal se produit lorsque la fonction `Partition()` partage approximativement le tableau A de taille n , en deux sous-tableaux de taille $n/2$. Pour que cela se produise, il faut que l'élément x , que l'on appelle le *pivot*, initialisé ici à $A[r]$, corresponde à la médiane du tableau, c'est à dire un élément qui comprendrait (à un près, si cela ne tombe pas juste) autant d'éléments qui lui sont inférieurs, que d'éléments qui lui sont supérieurs.

7. En supposant que ce scénario idéal se produit à chaque appel récursif, écrire l'équation récurrente qui exprime la complexité $T(n)$ du tri rapide de n éléments en fonction de $T(n/2)$. Quelle est la solution de cette équation ? (vous pouvez démontrer ce résultat, ou utiliser le théorème central vu en cours pour répondre à cette question).
8. Le pire des cas se produit quand l'appel à la fonction `Partition()` partage à chaque étape, le tableau A en un tableau de taille $n-1$, et un autre tableau vide. Écrire alors $T(n)$ en fonction de $T(n-1)$, et résoudre l'équation récurrente dans ce pire des cas.

Même si l'appel à la fonction `Partition()` partage systématiquement le tableau en deux parties inégales (par exemple, 90% et 10% des cases du tableau), on peut montrer que la complexité de l'algorithme reste intéressante, et plus proche de la complexité de la question 7 que de celle de la question 8. En moyenne, sur un tableau quelconque, on estime que les proportions sont plutôt de l'ordre de 80% et 20%, pour la taille des deux sous-tableaux.

9. Montrez en revanche que si on donne en entrée à l'algorithme un tableau trié (qui n'est donc pas un tableau quelconque), on se trouve à chaque étape dans le cas défavorable évoqué en question 8.

Nous disposons donc d'une version de l'algorithme Tri-Rapide, dont la complexité est $\Theta(n \cdot \log n)$ en moyenne, mais $\Theta(n^2)$ dans le pire des cas. Nous allons nous intéresser, dans la suite de ce problème, à la sélection de la médiane en temps linéaire, qui permet d'obtenir un algorithme de complexité $\Theta(n \cdot \log n)$ même dans le pire des cas.

10. Quelle est la médiane du tableau A de la question 4 ?
11. Montrez que si l'on savait où se trouve la médiane dans le tableau, il suffirait d'un seul appel supplémentaire à la fonction `Echange()` pour se trouver dans la situation favorable de la question 7.
12. Une première amélioration à l'algorithme consiste à le « randomiser », c'est à dire à choisir comme pivot, lors du partitionnement non pas forcément la dernière case du tableau, mais une case aléatoire à l'intérieur du tableau. Montrer comment modifier `Partition()` pour obtenir la fonction `Partition_Randomisee()` qui implémente cette idée (on pourra utiliser la fonction `Rand(a, b)` qui renvoie un nombre entier aléatoire compris entre a et b inclus).

En cas de malchance, on obtient (très rarement) un algorithme en $\Theta(n^2)$, mais il n'existe plus *une entrée particulière* qui génère ce comportement, mais seulement *une séquence malchanceuse* de tirages au sort pour laquelle cela se produit. En moyenne, sur un grand nombre d'exécutions, la complexité est de $\Theta(n \cdot \log n)$.

Par la suite, plutôt que traiter le problème de la détermination de la médiane, nous allons traiter le problème de la sélection de l'élément de rang i , c'est à dire de l'élément x tel qu'il existe exactement i éléments inférieurs ou égaux à x dans le tableau.

13. Pour quelle valeur de i faudra-t-il appeler cet algorithme pour obtenir la médiane du tableau ?
14. Montrer qu'en utilisant un algorithme bien connu, on peut résoudre ce problème avec une complexité $\Theta(n \log n)$.

On va maintenant chercher à résoudre le problème de la sélection de l'élément de rang i en temps linéaire.

15. Proposer un algorithme de complexité $O(n)$ qui résolve le problème de la sélection pour $i=0$, (donc, trouver le plus petit élément de A) et pour $i=1$ (donc, trouver le deuxième plus petit élément de A). Indication : chercher simultanément les deux plus petits.

Lorsque l'on appelle la fonction `Partition_Randomisee()`, on obtient en sortie le rang q du pivot. Si par hasard q égal à i , on a résolu le problème de la sélection de l'élément de rang i . Sinon, en comparant q et i , on peut savoir dans quel sous-tableau (à gauche ou à droite de q) rechercher le $i^{\text{ème}}$ plus petit élément.

16. En utilisant cette idée, proposer une version récursive de la fonction `Selection_Randomisee(A, p, r, i)`, qui renvoie le $i^{\text{ème}}$ plus petit élément du sous-tableau de A , dont les indices sont compris entre p et r . Attention au rang dans le sous-tableau qui n'est en général, pas égal au rang dans le tableau complet !
17. En reprenant l'hypothèse de la question 7, écrivez l'équation récurrente, qui relie $T(n)$ à $T(n/2)$, et résolvez-là dans le cas le plus favorable (qui est également le plus proche du cas moyen). Dans le cas le plus défavorable, reliez $T(n)$ à $T(n-1)$, et déduisez-en la complexité dans le pire des cas.

Pour les mêmes raisons que précédemment, on obtient maintenant un algorithme linéaire *en moyenne* pour résoudre le problème de la sélection du $i^{\text{ème}}$ élément d'un tableau. Mais il reste possible (quoique très improbable) que cet algorithme effectue un grand nombre d'opérations.

Sachez qu'il existe un algorithme déterministe et linéaire même dans le pire des cas, pour la sélection du $i^{\text{ème}}$ élément, mais ce problème est déjà bien assez long pour un examen de 1h30, et nous avons eu pitié de vous...