Examen de Complexité

Durée : 1h30

Documents autorisés. Connexion à Internet interdite (smartphones et ordinateurs interdits)

Questions de cours

Vous pouvez justifier vos réponses en donnant les algorithmes qui résolvent les problèmes proposés, et en explicitant la raison pour laquelle on ne peut pas trouver mieux.

Étant donné un tableau T de n éléments arbitraires :

- 1) Quelle est la complexité, *dans le pire des cas*, de la recherche de l'élément x dans le tableau T :
 - a. Si T est quelconque ?
 - b. Si T est trié du plus petit au plus grand élément ?
- 2) Mêmes questions pour la complexité de ce problème, *dans le meilleur des cas* ?
- 3) Quelle est la complexité, *dans le pire des cas*, de la recherche du plus petit élément y de T ?
 - a. Si T est quelconque ?
 - b. Si T est trié du plus petit au plus grand élément ?
- 4) Mêmes questions pour la complexité de ce problème, *dans le meilleur des cas* ?
- 5) Quelle est la complexité algorithmique du problème du tri du tableau T , en ne s'autorisant, pour seules opérations sur les éléments du tableau que la comparaison et l'échange de deux éléments du tableau ?

Nous avons étudié et comparé pendant le cours et les travaux dirigés, six algorithmes de tri qui n'effectuent que des comparaisons et des échanges : Tri par bulles, Tri fusion, Tri par insertion, Tri rapide, Tri par sélection, Tri par tas. Pour trier des tableaux de n éléments, certains de ces algorithmes ont une complexité algorithmique en $O(n^2)$, d'autres une complexité en $O(n \log n)$. Certains ont une complexité mémoire de $O(n)$, d'autres ont une complexité mémoire de $O(1)$.

Saurez-vous identifier (en justifiant vos choix) ces six algorithmes par ce jeu de devinettes ?

- a) J'ai une complexité algorithmique *en moyenne* de $O(n \log n)$, mais *dans le pire des cas* de $\Theta(n^2)$. Qui suis-je ?
- b) J'ai une complexité algorithmique de $O(n \log n)$ et une complexité mémoire de $O(1)$. Qui suis-je ?
- c) J'ai une complexité algorithmique de $O(n \log n)$, mais j'ai besoin d'un tableau auxiliaire de taille $O(n)$ pour pouvoir fonctionner. Qui suis-je ?
- d) Je fais au plus $n-1$ échanges, quel que soit le tableau qui m'est donné en entrée. Qui suis-je ?
- e) Je fais systématiquement exactement $n(n-1)/2$ comparaisons, quel que soit le tableau qui m'est donné en entrée. Je fais un nombre d'échanges compris entre 0 et $n(n-1)/2$ en fonction du tableau qui m'est donné en entrée. Qui suis-je ?
- f) Si on me donne en entrée un tableau trié, je ne fais que $n-1$ comparaisons, et aucun échange. Qui suis-je ?

Tri par paquets

Lorsque l'on fait certaines hypothèses sur les données que l'on doit trier, et que l'on se donne d'autres instructions que les comparaisons et les échanges, on peut obtenir des algorithmes plus efficaces que les algorithmes de tri généralistes. C'est le cas de l'algorithme de tri par paquets, dans

lequel on va faire une hypothèse sur la distribution des données. On utilisera ensuite la valeur d'une donnée source pour placer cette donnée à un endroit particulier d'un tableau destination (opération élémentaire qui n'existe pas dans les algorithmes de tri par comparaison et échange). La suite de ce problème se propose d'étudier cet algorithme.

On suppose disposer d'un tableau A contenant n nombres aléatoires compris entre 0 et 1. On suppose que ces nombres suivent une distribution uniforme, c'est à dire que tous les intervalles de même longueur contiennent approximativement le même nombre de points. Dans l'exemple suivant, on a tiré n=10 nombres aléatoires, compris entre 0 inclus et 1 exclus. Le tableau A initial vaut $\langle 0.78, 0.17, 0.39, 0.26, 0.72, 0.94, 0.21, 0.12, 0.23, 0.68 \rangle$. On découpe l'intervalle $[0, 1[$ en n=10 sous-intervalles de longueurs égales $[0, 0.1[$, $[0.1, 0.2[$, et ainsi de suite jusqu'à $[0.9, 1[$, et on place chaque nombre du tableau A dans le sous-intervalle qui lui est associé. On crée pour cela un tableau B dont les éléments sont des listes de nombres issus du tableau A, et tombant dans le même sous-intervalle.

		Étape 1 : remplir les	Étape 2 : trier les listes	Étape 3 : concaténer les
		listes de B	de B	listes de B
	A	B	B	B
0	0.78	0	0	0
1	0.17	1	1	1
2	0.39	2	2	2
3	0.26	3	3	3
4	0.72	4	4	4
5	0.94	5	5	5
6	0.21	6	6	6
7	0.12	7	7	7
8	0.23	8	8	8
9	0.68	9	9	9

- 1) Illustrez l'action du tri par paquets sur le tableau $A = \langle 0.79, 0.13, 0.16, 0.64, 0.39, 0.20, 0.89, 0.53, 0.71, 0.42 \rangle$
- 2) Donnez un algorithme s'exécutant en $O(n)$ et permettant d'effectuer l'étape 1¹.
- 3) Pour l'étape 2, on suppose que $B[i]$ contient une liste de taille n_i . L'hypothèse sur l'uniformité de la distribution permet en moyenne, d'assimiler n_i à une constante indépendante de n. On peut prendre n'importe quel algorithme de tri quadratique (par exemple, le tri par insertion, qui fonctionne très bien pour des petits tableaux). Sous cette hypothèse, quelle est la complexité du tri de l'une des lignes du tableau B ? Et de l'étape 2 dans son ensemble ?
- 4) Donnez un algorithme linéaire permettant d'effectuer l'étape 3.
- 5) En conclure que l'algorithme complet s'exécute en moyenne en $O(n)$.
- 6) Quel est le temps d'exécution de l'algorithme, dans le cas le plus défavorable ? quelle modification simple permettrait à l'algorithme de garder son temps d'exécution moyen linéaire, tout en obtenant un temps d'exécution de $O(n \log n)$ dans le pire des cas ?

[bonus] Si on gère les listes de B dans des tableaux créés au début de l'exécution de l'algorithme, et que l'on veut permettre de traiter le pire des cas, quelle est la complexité en mémoire de cette implémentation ? Pouvez-vous proposer une ou plusieurs autres structures de données, qui limiteraient à $O(n)$ la complexité mémoire de l'algorithme ?

¹ On pourra utiliser la fonction $\text{int}(x)$ qui permet de calculer la partie entière du nombre à virgule x.