Master 1 Informatique — Ducci 2
Mercredi 10 janvier 2018

Enseignants :
Mehdi Ayadi / Carlos Crispim / Serge Miguet
Université Lumière Lyon 2

Examen de Complexité

Durée : 1h30

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

Tris par comparaison : questions de cours

Nous avons étudié et comparé pendant le cours et les travaux dirigés, six algorithmes de tri dont les opérations élémentaires sont la comparaison entre deux éléments du tableau, et l'échange de deux éléments du tableau.

1. Reproduisez le tableau (Figure 1) sur votre copie, et remplissez-le en utilisant la notation de Landau. On désignera par n la taille du tableau. Justifiez par quelques phrases vos réponses dans le texte, sans pour autant détailler chacun des algorithmes ni les preuves complètes de l'analyse de leur complexité.

Complexité / Algorithme	En Moyenne	Pire des cas	Meilleur des cas	Complexité Mémoire
Tri par bulles				
Tri par insertion				
Tri par sélection				
Tri fusion (mergesort)				
Tri rapide (quicksort)				
Tri par tas (heapsort)				

Figure 1 : Complexité des tris

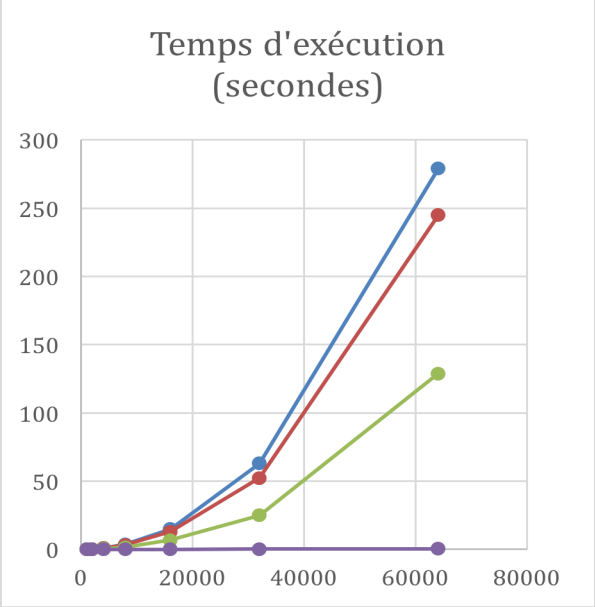

Figure 2 : Temps d'exécution des tris

On mesure les temps d'exécution (Figure 2) en secondes (en ordonnée) des quatre premiers algorithmes en fonction de la taille du tableau n (en abscisse).

2. Essayez de prédire à quel algorithme correspond quelle courbe
3. Pourquoi l'une des courbes semble-t-elle être « plate » ?
4. Comment peut-on estimer expérimentalement la valeur de la constante cachée dans la notation O( ) exprimant la complexité des différents algorithmes de tri ?
5. Rappelez les arguments permettant d'affirmer que tout algorithme de tri qui utilise des comparaisons comme opérations élémentaires, a une complexité de $O(n.\log n)$.
6. En faisant référence au problème du « crêpier psychorigide » étudié en TD, montrer que la complexité algorithmique d'un problème peut dépendre de ce que l'on considère comme étant une opération élémentaire.

La suite de ce problème est consacrée à l'étude d'un algorithme de tri utilisant d'autres opérations élémentaires que la comparaison et l'échange entre deux éléments du tableau. On pourra alors obtenir, dans certain cas, un algorithme ayant une complexité meilleure que celle évoquée en question 5.

Tri par dénombrement

Le tri par dénombrement suppose que chacun des n éléments du tableau d'entrée est un entier de l'intervalle 0 à k, k étant également un nombre entier.
Le principe du tri par dénombrement est de déterminer, pour chaque élément x de l'entrée, le nombre d'éléments inférieurs à x. Cette information peut servir à placer l'élément x directement à sa position dans le tableau de sortie. Par exemple, s'il existe 17 éléments strictement inférieurs à x, alors x se trouvera en sortie à la position 18. Ce schéma devra être légèrement modifié pour gérer la situation dans laquelle plusieurs éléments ont la même valeur, puisqu'on ne veut pas tous les placer à la même position.

7. Proposer un algorithme qui parcourt le tableau A[1..n] et calcule le tableau C[0..k], dans lequel C[i] correspond au nombre de fois ou l'élément i apparaît dans le tableau A (pensez à l'initialisation de C). Par exemple, avec n=8 et k=5, si A est le tableau [2, 5, 3, 0, 2, 3, 0, 3], C[0] vaut 2 car 0 apparaît deux fois dans A, C[1] vaut 0 car 1 n'apparaît pas dans A, C[2] vaut 2 car 2 apparaît 2 fois dans le tableau A.
8. Que vaut le tableau C complet dans l'exemple précédent ?
9. Quelle est la complexité de cet algorithme ?
10. Donnez un deuxième algorithme qui parcourt le tableau C ainsi construit, et qui remplace C[i] par le nombre d'éléments inférieurs ou égaux à i dans tableau A. Dans notre exemple, C[0] vaut 2 (il y a deux fois 0 dans A), C[1] vaut 2 (il n'y a pas de 1, donc les deux 0 sont les deux seuls éléments inférieurs ou égaux à 1), et C[2] vaut 4 (car il existe 4 éléments dans A qui sont inférieurs ou égaux à 2 : 0, 0, 2 et 2).
11. Que vaut le nouveau tableau C complet dans l'exemple précédent ?
12. Quelle est la complexité de ce deuxième algorithme ?
13. On construit maintenant un tableau B, de la même taille que A, dans lequel les éléments de A sont triés. Si on prend un élément x quelconque dans A, on peut le placer directement à l'indice C[x] du tableau B, puisqu'il y a exactement C[x] éléments inférieurs ou égaux à x. Il faut par contre ensuite décrémenter C[x], pour qu'un nouvel élément de A qui aurait la valeur x se place dans la case précédente du tableau B. Par exemple, la première case du tableau A est 2. C[2] vaut 4. On peut donc mettre le 2 à l'indice 4 de B, mais il faut décrémenter C[2], qui passe donc à la valeur 3, pour que le prochain 2 que l'on rencontre dans A soit placé à l'indice 3 de B. Ecrivez un troisième algorithme qui implémente cette idée, et donnez sa complexité.
14. Dans l'exemple précédent, donnez les valeurs des tableaux B et C à chacune des étapes de ce troisième algorithme.
15. Quelle est la complexité algorithmique finale du tri par dénombrement, en fonction de n et k ? Quelle est sa complexité en mémoire ?
16. En déduire que si $k = O(n)$, on obtient un algorithme de tri linéaire en la taille du tableau.
17. A partir de quelles valeurs de k cet algorithme devient-il moins intéressant que les meilleurs algorithmes de tris par comparaison ?