DS 2

Informatique pour tous, deuxième année

Julien Reichert

Durée : 2 heures maximum.

Attention : La question de cours est sur six points, et un maximum de quatorze points peut être obtenu pour les autres exercices, valant chacun six points (barème prévisionnel).

1 Question de cours

Exercice 1 : Écrire un des algorithmes de tri au programme (insertion, fusion ou rapide), au choix dans sa version en place ou non en place. Prouver sa terminaison lorsqu'il est récursif ou contient une boucle conditionnelle ; calculer sa complexité dans le pire des cas par une analyse rigoureuse, ainsi que sa complexité dans le meilleur des cas en précisant pour quel genre de liste la complexité asymptotique est extrémale. Prouver aussi la correction de l'algorithme.

2 Autour du tri par insertion

Exercice 2.1 : Écrire une version du tri par insertion procédant par dichotomie pour trouver l'endroit où placer chaque élément successif de la liste à trier. L'algorithme peut agir au choix en place ou sur de la mémoire extérieure.

Exercice 2.2 : Déterminer la complexité asymptotique de l'algorithme précédent, d'abord en nombre d'affectations, puis en nombre de comparaisons. Il suffit de donner la formule de récurrence et d'utiliser le cours du chapitre précédent.

3 Tri sur un ensemble de taille deux ou trois

Remarque : La deuxième question a pour solution standard une version de l'algorithme du drapeau néerlandais, ou « tri Dijkstra », du nom de son inventeur.

Exercice 3.1 : Écrire un programme nommé `tri_ceratops` qui trie une liste en argument dont on sait qu'elle ne contient que des chaînes de caractère valant `"carnivore"` ou `"herbivore"`. La complexité devra être linéaire **et ceci devra être justifié** et le tri sera de préférence en place. Évidemement, `"carnivore"` $>$ `"herbivore"`.

Exercice 3.2 : Écrire un programme nommé `tri_cholome` qui trie une liste en argument dont on sait qu'elle ne contient que des chaînes de caractère valant `"comestible"`, `"toxique"` ou `"mortel"`. On utilise les mêmes critères que précédemment. Ici, `"mortel"` $>$ `"toxique"` $>$ `"comestible"`.

Bien évidemment, le tri dénombrement n'étant pas en place, il ne donnera pas la totalité des points. Le but est de faire des échanges de manière pertinente.

4 Diviser en trois pour (moins bien) régner

Exercice 4.1 : Écrire un programme nommé `tri_chotomie` qui effectue une variante du tri rapide où on utilise deux pivots, sans contrainte sur la façon dont le programme choisit les pivots. On pourra donc tenir compte ou non des optimisations présentées en ce qui concerne la médiane des médianes.

Exercice 4.2 : Donner la formule de récurrence permettant de calculer la complexité asymptotique de ce programme. En déduire cette complexité et discuter de l'intérêt de faire une division en trois parts plutôt qu'en deux.

Exercice 4.3 : Reprendre les deux exercices précédents en effectuant cette fois une variante du tri fusion où une liste est coupée en trois sous-listes équilibrées.

5 Tri avec des piles

Exercice 5.1 : Écrire un algorithme de tri travaillant sur une structure de pile [1] et tel que la mémoire autorisée soit uniquement constituée de piles (les affectations de variables d'autres types sont donc interdites). La complexité temporelle peut être quadratique, mais on impose le moins de piles possible.

Exercice 5.2 : Prouver la correction de l'algorithme précédent et déterminer sa complexité asymptotique.

1. La structure est au choix, mais il est recommandé de se servir de listes et d'utiliser les méthodes `pop` et `append` et des fonctions `sommet` et `est_vide` supposées déjà écrites, chacune coûtant une instruction élémentaire.