13.4 Exercices

Exercice 13.8 ** Le tri par sélection consiste, comme le tri par insertion, à maintenir à chaque itération i une portion du tableau `a[0:i]` déjà triée. En revanche, au lieu de chercher à insérer `a[i]` dans cette portion, on recherche le minimum des éléments de `a[i:n]` et on échange ce minimum avec `a[i]`. La portion du tableau `a[0:i+1]` est alors triée.

1 Écrire une fonction qui réalise le tri par sélection.
2 Démontrer la correction de ce tri.
3 Évaluer sa complexité en temps, en espace, dans le meilleur et dans le pire des cas.

Exercice 13.9 * Montrer que, dans le cas du tri rapide du programme 17 page 321, le nombre maximal d'appels imbriqués à la fonction **`tri_rapide_rec`** ne peut excéder $\log N$.

Exercice 13.10 ** Étant donné un entier k, on dit qu'un tableau est *k-presque trié* :

- si chacun de ses éléments est à au plus k indices de la position où il devrait être ;
- ou bien si au plus k de ses éléments ne sont pas à leur place.

Démontrer qu'à k fixé, le tri par insertion a une complexité en $O(n)$ sur les tableaux k-presque triés.

Exercice 13.11 * Une idée classique pour accélérer un algorithme de tri consiste à effectuer un tri par insertion quand le nombre d'éléments à trier est petit, c'est-à-dire devient inférieur à une constante fixée à l'avance (par exemple 5). Modifier le tri rapide de tableaux pour prendre en compte cette idée. On pourra reprendre la fonction **`tri_insertion`** et la généraliser en lui passant deux indices `g` et `d` pour délimiter la portion du tableau à trier.

Exercice 13.12 ** **Calcul rapide de la médiane.** L'algorithme que l'on écrit ici permet de déterminer la médiane, et même plus généralement le k-ième élément d'un tableau, sans le trier intégralement. Il est linéaire dans le pire des cas.

1 Écrire une fonction qui prend en argument un tableau de cinq éléments et calcule sa médiane.
2 Écrire une fonction qui prend en argument un tableau quelconque, le divise en groupes de cinq éléments et construit le tableau des médianes de chaque groupe de cinq.
3 Modifier la fonction précédente pour qu'elle s'appelle récursivement sur le « tableau des médianes » construit.
4 Enfin, écrire une fonction qui effectue une partition du tableau de départ avec pour pivot la « médiane des médianes » calculée précédemment.
Pour trouver le k-ième élément du tableau, où doit-on le chercher en fonction des tailles des deux sous-tableaux délimités par la partition ? Programmer l'appel récursif correspondant.
5 Pourquoi n'est-ce pas une bonne idée d'extraire les groupes de cinq éléments avec la construction `t[i:i+5]` ? Comment peut-on procéder autrement ?
6 De même, on pourra chercher une façon de construire en place le tableau des médianes.

Exercice 13.13 * Une façon d'optimiser la fonction **`tri_fusion`** consiste à éviter la fusion lorsque, à l'issue des deux appels récursifs, les éléments de la moitié gauche se trouvent être tous plus petits que les éléments de la moitié droite. On le teste facilement en comparant l'élément le plus à droite de la moitié gauche et l'élément le plus à gauche de la moitié droite. Modifier la fonction **`tri_fusion`** en suivant cette idée.

Exercice 13.14 ** Pour éviter la copie de `a` vers `tmp` (avec `tmp[g :d] = a[g :d]`) dans la fonction **`tri_fusion`**, une idée consiste à trier les deux moitiés du tableau `a` tout en les déplaçant vers le tableau `tmp`, puis à fusionner de `tmp` vers `a` comme on le fait déjà. Cependant, pour trier les éléments de `a` vers `tmp`, il faut, inversement, trier les deux moitiés en place puis fusionner vers `tmp`. On a donc besoin de deux fonctions de tri mutuellement récursives. On peut cependant n'en n'écrire qu'une seule, en passant un paramètre supplémentaire indiquant si le tri doit être fait en place ou vers `tmp`. Modifier les fonctions **`tri_fusion`** et **`tri_fusion_rec`** en suivant cette idée.

Exercice 13.15 * Comme pour le tri rapide, on peut terminer le tri fusion par un tri par insertion lorsque le nombre d'éléments à trier devient petit.