Devoir en temps libre n° 4

Ce devoir est à préparer pour le lundi 14 mai 2018 et sera à rendre au début du TP. Vous devez apporter[1] votre code CAML avec vous, sous la forme d'un unique fichier `.ml` dont le nom sera obligatoirement et exactement `<login>.ml` où `<login>` est votre nom d'utilisateur du laboratoire d'informatique (tout en minuscules avec les tirets).

Votre programme devra respecter scrupuleusement les noms et les types indiqués (ou un type plus général). Le code devra compiler sans erreurs ni aucun message d'avertissement. L'encodage du fichier devra être UTF-8 et vous n'utiliserez que des caractères ASCII pour les noms de variables. Il est impératif de veiller au strict respect de ces consignes.

Lorsque l'on pose des hypothèses sur les arguments (par exemple qu'un argument est un entier naturel), il n'est pas nécessaire dans vos fonctions de le vérifier ni de prévoir le comportement si l'hypothèse est violée (par exemple si l'argument est négatif). On pourra cependant lever des exceptions (`failwith "Argument non valable"` par exemple) si on le souhaite.

✔ On prendra bien soin de tester toutes les fonctions sur des cas triviaux, puis un peu plus complexes.

I Tris de tableaux

Dans cette partie, on se propose d'implémenter pour les tableaux le tri par insertion et le tri rapide, que nous avons déjà rencontrés pour les listes. On fera bien attention aux différents cas particuliers (tableaux vides, notamment).

Question 1

Écrire une fonction `echange : 'a array -> int -> int -> unit` telle que `echange tab i j` échange le contenu des cases d'indices `i` et `j` du tableau `tab`. On pourra supposer que les indices sont valables, mais je vous encourage plutôt à lever une erreur si ce n'est pas le cas pour faciliter la détection de bogues[a] par la suite.

a. Défaut de conception ou de réalisation se manifestant par des anomalies de fonctionnement (anglais : *bug*). Journal officiel de la République française du 19 février 1984, p. 51741.

Question 2

Écrire une fonction `tri_insertion : 'a array -> unit` qui trie un tableau en insérant successivement les éléments à la bonne place dans la partie du tableau déjà triée. Le tri devra se faire sur place, c'est-à-dire sans créer un autre tableau.

1. Je vous conseille de vous l'envoyer par mail depuis chez vous *et* de l'apporter sur clef USB.

Question 3

Écrire une fonction `partition : 'a array -> int -> int -> int` telle que l'appel `partition tab i j` sur un tableau $(t_k)_{0\leq k\leq n-1}$ avec $0 \leq i < j \leq n-1$ réarrange le sous-tableau $(t_k)_{i\leq k\leq j}$ suivant le pivot t_i pour avoir les éléments plus petits que t_i avant t_i et ceux plus grands que t_i après t_i. La fonction doit renvoyer l'indice p du pivot dans le tableau réarrangé. Après l'appel, on doit donc avoir $\forall k\,(i \leq k \leq p \Rightarrow t_k \leq t_p) \wedge (p < k \leq j \Rightarrow t_p < t_k)$ dans le tableau modifié (par effets de bord !).

Question 4

Écrire une fonction `tri_rapide_aux` de type `'a array -> int -> int -> unit` tel que l'appel `tri_rapide_aux tab i j` sur un tableau $(t_k)_{0\leq k\leq n-1}$ avec $0 \leq i < j \leq n-1$ trie sur place le sous-tableau $(t_k)_{i\leq k\leq j}$.

Question 5

En déduire la fonction `tri_rapide : 'a array -> unit` qui trie sur place un tableau.

II Autour des permutations

Une permutation σ de $[\![0, n-1]\!]$ est une bijection de cet ensemble dans lui-même. Nous représenterons une permutation σ de $[\![0, n-1]\!]$ par un tableau `sigma` de taille n tel que pour tout $i \in [\![0, n-1]\!]$, l'entier `sigma.(i)` soit égal à $\sigma(i)$.

Question 6

Écrire une fonction `est_permutation : int array -> bool` qui teste si un tableau représente une permutation. On cherchera une solution de complexité linéaire.

Le support d'une permutation σ est l'ensemble des points non fixes de σ, *i.e.* l'ensemble des i tels que $\sigma(i) \neq i$.

Question 7

Écrire une fonction `support : int array -> int list` qui calcule le support d'une permutation sous forme d'une liste d'entiers.

La composée de deux permutations σ et σ' de $[\![0, n-1]\!]$ est l'application $\sigma \circ \sigma'$, au sens habituel des fonctions. On vérifie facilement que la composée de deux permutations de $[\![0, n-1]\!]$ est une permutation de $[\![0, n-1]\!]$.

Question 8

Écrire une fonction `compose : int array -> int array -> int array` qui compose deux permutations.

Question 9

Écrire une fonction `inverse : int array -> int array` qui calcule l'inverse d'une permutation σ, c'est-à-dire la permutation σ^{-1} telle que $\sigma \circ \sigma^{-1} = \sigma^{-1} \circ \sigma = Id_{[\![0,n-1]\!]}$.