LISTES

Écrivez les procédures récursives suivantes qui agissent sur les listes. Dans tous les cas, si c'est nécessaire, vous ajouterez un ou plusieurs tests pour intercepter les erreurs (par exemple, le maximum d'une liste vide n'existe pas, il faut donc signaler que la liste est vide).

1) Fonctions élémentaires.

a) longueur liste de type `'a list -> int`

b) dernier liste de type `'a list -> 'a`, qui retourne le dernier élément de la liste

c) ajout_fin liste x de type `'a list -> 'a -> 'a list`, qui ajoute en fin de liste un élément

2) Les listes sont supposées être des listes de nombres.

a) maximum liste de type `'a list -> 'a`, qui retourne le plus grand élément de la liste

b) somme liste de type `int list -> int`, qui retourne la somme de tous les éléments de la liste (par convention habituelle en mathématiques, une somme vide vaut 0)

3) Manipulations sur les éléments.

a) appartenir x liste de type `'a -> 'a list -> bool`, qui « dit » si un objet appartient à une liste

b) occurences x liste de type `'a -> 'a list -> int`, qui retourne le nombre d'apparitions d'un objet dans une liste

c) pos_init x liste de type `'a -> 'a list -> int`, qui retourne l'indice de la première apparition de x s'il est dans la liste, -1 sinon

d) supprimer_tous x liste de type `'a -> 'a list -> 'a list`, qui retourne la liste privée de tous les objets x, l'ordre étant conservé

e) ieme k liste de type `int -> 'a list -> 'a`, qui donne le k-ème élément de la liste

f) supprimer k liste de type `int -> 'a list -> 'a list`, qui retourne la liste privée de son k-ème élément

g) inserer x k liste de type `'a -> int -> 'a list -> 'a list`, qui retourne la liste dans laquelle on a inséré en k-ème position l'objet x

h) On suppose que la liste est une liste de nombres triée par ordre croissant : inserer_place x liste de type `'a -> 'a list -> 'a list`, qui retourne la liste dans laquelle on a inséré le nombre x à la bonne place, de sorte que la liste obtenue soit encore triée par ordre croissant

i) appliquer f liste de type `('a -> 'b) -> 'a list -> 'b list`, qui construit la liste des images par une fonction f des éléments de la liste initiale

4) Manipulations globales de listes.

a) miroir liste de type `'a list -> 'a list`, qui calcule l'image miroir de la liste

b) partage liste i de type `'a list -> int -> ('a list * 'a list)`, qui donne les deux sous-listes de la liste coupée après l'élément d'indice $i-1$: si $liste = [a_0; ...; a_{n-1}]$, on obtient $[a_0; ...; a_{i-1}]$ et $[a_i; ...; a_{n-1}]$

c) sous_liste liste i j de type `'a list -> int -> int -> 'a list`, qui donne la sous-liste $[a_i; ...; a_{j-1}]$: elle pourra ne pas être récursive et utiliser la fonction précédente

d) couper liste de type `'a list -> ('a list * 'a list)`, qui retourne deux sous-listes de tailles égales à un élément prés : peu importe l'ordre des éléments dans les deux sous-listes, peu importe aussi la facon dont ils sont sélectionnés dans la liste initiale

5) Avec deux listes.

a) concatene li1 li2 de type `'a list -> 'a list -> 'a list`, qui retourne la concaténation des deux listes : $[a_0; ...; a_{n-1}; b_0; ...; b_{p-1}]$

b) concatene_envers li1 li2 de type `'a list -> 'a list -> 'a list`, qui retourne la concaténation des deux listes, la première étant renversée : $[a_{n-1}; ...; a_0; b_0; ...; b_{p-1}]$ (sans utiliser la fonction miroir)