Programmation récursive, listes

1. Ecrire une fonction `dernier : 'a list -> 'a` qui donne le dernier élément d'une liste et une fonction `avantdernier : 'a list -> 'a` qui donne l'avant-dernier.
Ecrire une fonction `nieme : int -> 'a list -> 'a` qui renvoie le n-ième élément d'une liste.
Ecrire une fonction `listmake : int -> 'a -> 'a list` renvoyant une liste de n éléments égaux (équivalent de `Array.make` pour les listes).
Ecrire une fonction calculant la somme des éléments d'une liste d'entiers.
Ecrire une fonction `maxlist: 'a list -> 'a` qui renvoie le plus grand élément d'une liste. La liste passée en argument pourra contenir tous les types acceptés par la fonction intégrée `max`. De même écrire une fonction `minmaxlist: 'a list -> 'a * 'a` qui renvoie le couple formée par le plus petit et plus grand élément d'une liste en un seul passage sur la liste.

2. Les fonctions suivantes `List.exists` et `List.for_all` sont prédéfinies en Caml. Ecrire ces fonctions en procédant par récursivité et par filtrage.
- `exists : ('a -> bool) -> 'a list -> bool` qui détermine s'il existe un élément de la liste vérifiant une propriété donnée.
- `for_all : ('a -> bool) -> 'a list -> bool` qui détermine si tous les éléments de la liste vérifient une propriété donnée.
Tester ces deux fonctions avec la propriété "être positif".

3. La fonction `List.rev : 'a list -> 'a list` prédéfinie en Caml renvoie le miroir d'une liste (liste dans laquelle l'ordre des éléments a été inversé).
a. En procédant par récursivité et filtrage, définir une fonction `miroir` réalisant cette transformation (vous pouvez utiliser l'opérateur de concaténation @). Montrer que le coût de cette fonction est quadratique.
b. Rédiger une version de cette fonction de coût linéaire

4. Ecrire une fonction `test_monotone : 'a list -> bool` qui renvoie le booléen vrai si la liste passée en argument est monotone, avec une complexité linéaire en la taille de la liste.

5. Ecrire une fonction `rotg` de compléxité linéaire qui fait tourner une liste d'un cran vers la gauche (par exemple, `rotg [1; 2; 3; 4]` renverra la liste `[2; 3; 4; 1]`).
En utilisant les fonctions précédentes, écrire la fonction `rotd` qui tourne une liste d'un cran vers la droite (sans utiliser de filtrage).

6. On souhaite représenter un ensemble par une liste, chaque élément de l'ensemble ne devant apparaître qu'une seule fois dans la liste, à un emplacement arbitraire.
a. Définir une fonction `intersection` qui calcule l'intersection de deux ensembles. Évaluer son coût en fonction des cardinaux de ces ensembles.
b. Définir de même l'union et la différence symétrique de deux ensembles.
c. Rédiger enfin une fonction `egal` qui détermine si deux ensembles sont égaux. Évaluer le coût de chacune de ces fonctions.