Exercice 1 : Ordre lexicographique

On rappelle qu'en OCaml la fonction `(<=) : 'a -> 'a -> bool` permet de comparer deux éléments de même type. L'*ordre lexicographique* sur des listes d'éléments de type `'a` est l'ordre du dictionnaire, où les listes jouent le rôle de mots dont les lettres sont les éléments contenus dans la liste. Par exemple `[1; 2; 2]` est lexicographiquement plus petit que `[1; 3]`, `[1; 2]` est lexicographiquement plus petit que `[1; 2; 2]`. `[1; 2; 2]` est lexicographiquement plus petit que `[1; 3; 0]`.

Q. 1 Définir une fonction `lexico_inf : 'a list -> 'a list -> bool` prenant en paramètres deux listes et testant si la première est, ou non, strictement inférieure à la seconde, pour l'ordre lexicographique.

Q. 2 Comment rendre cette fonction paramétrique en l'opérateur de comparaison des éléments dans les listes ?

Exercice 2 : Sous listes contiguës

Q. 1 Écrire une fonction `sous_listes_contigues: 'a list -> 'a list list` prenant en paramètre une liste `l` et calculant l'ensemble de ses sous-listes non vides d'éléments contigus dans la liste `l`.

```
sous_listes_contigues [1; 2; 3];;
- : int list list = [[1]; [1; 2]; [1; 2; 3]; [2]; [2; 3]; [3]]
```

Exercice 4 : Ordre lexicographique

On rappelle qu'en OCaml la fonction `(<=) : 'a -> 'a -> bool` permet de comparer deux éléments de même type. L'*ordre lexicographique* sur des tableaux d'éléments de type `'a` est l'ordre du dictionnaire, où les tableaux jouent le rôle de mots dont les lettres sont les éléments contenus dans la liste. Par exemple `[|1; 2; 2|]` est lexicographiquement plus petit que `[|1; 3|]`, `[|1; 2|]` est lexicographiquement plus petit que `[|1; 2; 2|]`. `[1; 2; 2|]` est lexicographiquement plus petit que `[1; 3; 0|]`.

Q. 1 Définir une fonction `lexico_inf : 'a array -> 'a array -> bool` prenant en paramètres deux tableaux et testant si la première est, ou non, strictement inférieure à la seconde, pour l'ordre lexicographique.

Q. 2 Comment rendre cette fonction paramétrique en l'opérateur de comparaison des éléments dans les tableaux ?

Exercice 5 : Nombre d'occurrences

Q. 1 Définir une fonction `histogramme (donnees: int list) (pas: int) (max: int): int array` prenant en paramètres une liste d'entiers (`donnees`) dont les éléments sont tous dans l'intervalle entier $[\![0, \texttt{max}-1]\!]$, un pas entier (`pas`) et une valeur entière `max` et retournant un tableau `t` de taille $\lceil \frac{\texttt{max}}{\texttt{pas}} \rceil$ contenant dans sa case d'indice `i` un entier indiquant le nombre d'entiers de la liste `donnees` se trouvant dans l'intervalle entier $[\![i \times \texttt{pas}, (i+1) \times \texttt{pas} - 1]\!]$.

Exercice 7 : Remplissage d'un tableau

Q. 1 Définir une fonction `remplissage : int array -> int list -> unit` prenant en paramètres un tableau d'entiers `t` et une liste *non vide* `l` et remplaçant, dans le tableau `t` les occurrences du nombre -1 par des valeurs piochées dans la liste `l`. Les valeurs de la liste `l` devront être utilisées dans l'ordre, si on atteint la fin de la liste, on recommence. Par exemple si `t` est le tableau `[|1; 2; -1; 0; -2; -1; -1|]` l'appel `(remplissage t [4; 5; 6; 8])` l'appel devra modifier le tableau `t` en la valeur `[|1; 2; 4; 0; -2; 5; 6|]`. `(remplissage t [4; 5])` devra modifier le tableau `t` en la valeur `[|1; 2; 4; 0; -2; 5; 4|]`.

Exercice 8 : Fusion de listes en tableau

Q. 1 Définir une fonction `fusion : 'a list -> 'a list -> 'a array` prenant en paramètres deux listes *triées par ordre croissant* et retournant un tableau `t`, *trié par ordre croissant* et dont les éléments sont ceux se trouvant dans l'union des listes en paramètres.

Exercice 9 : Sous-séquences monotones

Q. 1 Définir une fonction `sseq_monotones : 'a array -> int list` prenant en paramètres un tableau d'éléments et retournant la liste des longueurs des sous-séquences monotones du tableau en paramètre. La longueur d'une séquence monotone est le nombre d'éléments se trouvant dans la séquence décrémenté de 1. Par exemple le tableau `[|1; 5; 8; 4; 6; 9; 8; 7; 6; 1; 9|]` contient les séquences monotones : `[|1; 5; 8|]`, `[|8; 4|]`, `[|4; 6; 9|]`, `[|9; 8; 7; 6; 1|]` et finalement `[|1; 9|]` de longueurs respectives : 2, 1, 2, 4, 1. Ainsi le résultat attendu pour cette entrée est la liste `[2; 1; 2; 4; 1]`. *Indication :* On prendra soin de définir des fonctions auxiliaires et d'expliciter leurs comportements.