Exercice 1 :

1. Écrire une fonction `map : ('a -> 'b) -> 'a array -> 'b array` qui étant donnée `f` une fonction et `t` un tableau, calcule le tableau des images par `f` des éléments de `t`.
   Exemple : `map int_of_float [|-3.5; 0.; 4.2; -0.2|] = [|-3; 0; 4; 0|]`.
   À l'avenir, plutôt que réécrire cette fonction vous pouvez utiliser la fonction `Array.map`.
2. (a) Écrire une fonction récursive `tri_bulles_aux : 'a array -> int -> unit` qui, étant donné un tableau `t` de taille n et un indice $0 \leqslant i < n$, tel que les i plus petits éléments de `t` sont déjà placés aux indices 0 à $i-1$, effectue le tri bulles des éléments suivant (en faisant remonter vers le début les petits éléments, comme des bulles dans de l'eau).
   (b) Prouver la correction et la terminaison de votre fonction, et montrer que sa complexité est $\mathcal{O}\left((n-i)^2\right)$.
   (c) En déduire une fonction `tri_bulles : 'a array -> unit` qui trie un tableau.

Exercice 2 :

On rappelle le principe de la recherche dichotomique dans un tableau trié :

- Si le tableau est vide, l'élément qu'on recherche n'est pas dedans.
- Sinon, on regarde m la médiane du tableau. Si c'est x, l'élément qu'on recherche, on a fini. Si ce n'est pas x, alors on cherche dans les éléments strictement avant la m si $x < m$, et dans les éléments strictement après m si $x > m$.

1. Écrire une fonction itérative `recherche_dicho : int array -> int -> int * bool` qui étant donné un tableau d'entiers **trié** `t` et un entier x, calcule i tel que `t.(`i`)`$= x$, et renvoie `(true, i)`.
   Si cet indice n'existe pas, la fonction renverra `(false, -1)`.
   Cette fonction utilisera dans son code des effets de bords car elle est itérative, mais elle n'aura aucun effet de bord sur `t`.
2. (a) De quels arguments supplémentaire a-t-on besoin pour écrire une fonction récursive qui implémente le même algorithme **sans réaliser de copie de morceaux du tableau** ?
   (b) Écrire une fonction `recherche_dicho_2 : int array -> int -> int * bool` qui utilise une fonction récursive auxiliaire, mais n'utilise aucun effet de bord même sur des variables locales.
   (c) Vérifier que la complexité dans le pire cas de votre fonction est bien $\mathcal{O}(\log(n))$ où n est la taille du tableau.

Exercice 3 : **Tri fusion**

1. Écrire une fonction `partition : 'a list -> 'a list * 'a list` qui prend en argument une liste `l` et renvoie un couple de listes `(l1, l2)`, de mêmes tailles à 1 près, qui contiennent les mêmes éléments que `l`. L'ordre des éléments n'importe pas.
   Par exemple, `partition [1; 2; 3; 4; 5]` peut renvoyer `([5; 3; 1], [4; 2])`, mais ce n'est pas la seule façon de faire la partition.
2. Écrire une fonction `fusion : 'a list * 'a list -> 'a list` qui étant donné un couples de listes `l1` et `l2` toutes deux triées par ordre croissant, calcule une liste `l` contenant les mêmes éléments que `l1` et `l2`, également triée par ordre croissant.
3. En déduire une fonction `tri_fusion : 'a list -> 'a list` qui calcule une liste triée contenant les mêmes éléments que son argument.
4. (a) Écrire, selon un algorithme similaire mais en utilisant des effets de bord, une fonction `tri_fusion : 'a array -> unit` qui trie un tableau.
   (b) Réécrire cette fonction de façon à n'utiliser qu'un seul tableau auxiliaire au cours de son exécution.
   Pour cela on utilisera une fonction auxiliaire avec des arguments supplémentaires : les indices de début et de fin de la partie à trier, le tableau auxiliaire (en plus du tableau à trier), et un booléen indiquant si le résultat doit être placer dans le tableau à trier ou le tableau auxiliaire.