TP tris (à faire sur 2 séances)

Ce TP est à déposer à la fin de **chacune** des deux séances.

Avant d'écrire un invariant de boucle, il est fortement conseillé de dessiner sur une feuille le tableau partiellement trié après un nombre quelconque d'itérations.

Exercice 1 : tri par sélection

1. Spécifier puis programmer le sous-programme `minimum` renvoyant l'indice du plus petit élément d'un sous-tableau t[m..n[, soit le tableau `t` restreint aux cases dont les indices sont compris entre `m` et `(n-1)`.

   ```
   let minimum (t : array int) (m n : int) : int (* à compléter *)
   ```

   Utiliser les jeux de tests (et la commande `why3 execute` expliquée en commentaire) fournis dans `tp6.mlw` pour tester votre code de `minimum` et vous assurer que votre programme renvoie le résultat attendu.

2. Spécifier le sous-programme `tri_selection` de tri par sélection d'un tableau d'entiers. En utilisant le programme `minimum`, programmer `tri_selection`.

   ```
   let tri_selection (t : array int) : unit (* à compléter *)
   ```

   Utiliser les jeux de tests fournis dans `tp6.mlw` pour tester votre code de `tri_selection` et vous assurer que votre programme renvoie le résultat attendu.

Le principe du tri par sélection dans l'ordre croissant d'un tableau `t` dont les éléments sont indicés de 0 à N-1 est le suivant.

- Rechercher le plus petit élément de t, puis l'échanger avec l'élément `t[0]`.
- Rechercher ensuite le 2e plus petit élément parmi les éléments restants de t et l'échanger avec `t[1]`.
- Itérer le procédé jusqu'à ce que le tableau soit trié.

Indications :

- Les noms des prédicats de la bibliothèque de why3 à utiliser pour spécifier le tri sont rappelés dans `tp-tris.mlw`.
- L'échange de deux éléments utilisera le sous-programme `swap` vu au cours 3.

  ```
  val swap (a: array int) (i: int) (j: int) =
  requires { 0 <= i < length a /\ 0 <= j < length a }
  ensures { exchange (old a) a i j }
  ```

- Dans l'invariant de boucle, utiliser le prédicat (`sorted_sub t i j`) spécifiant que le sous-tableau t[i .. j[est trié.

Exercice 2 : tri par insertion

Programmer, tester et prouver un programme effectuant un tri par insertion (cf. http://fr.wikipedia.org/wiki/Tri_par_insertion).

```
let tri_insertion (t : array int) : unit (* à compléter *)
```

1. Commencer par le programme `tri_insertion_v0` qui est une version alternative à celle de wikipédia plus simple à prouver.
2. En s'inspirant de `tri_insertion_v0`, spécifier, programmer, tester et prouver la version plus optimisée de wikipédia.

Exercice 3 : tri à bulles

Programmer et prouver un programme effectuant un tri à bulles.

```
let tri_bulles (t : array int) : unit (* à compléter *)
```