Colle n°2

Tous vos fichiers sont à renommer comme d'habitude (en y ajoutant votre nom en majuscule) et à déposer sur Moodle dans la section `Informatique - Devoirs`, tout en bas dans 1ère session. Vous devrez terminer les exercices de cette colle et me remettre l'ensemble des fichiers corrigés sur Moodle au même endroit avant dimanche soir.
Pensez à toutes les bonnes pratiques de programmation : réfléchir sur papier, tester dès que cela est possible, méthode des petits pas, programmation défensive...
Si vous avez terminé en moins d'1h, venez me demander un exercice supplémentaire.

Exercice 1 (Tri par insertion (BELLOCQ, BLOIS, BOCQUET)).
L'objet de cet exercice est de trier dans l'ordre croissant les entiers d'un tableau. Dans le tri par sélection, des permutations successives réorganisent progressivement le tableau. Le plus petit élément est d'abord permuté avec celui situé en première position. Puis le deuxième plus petit élément est permuté avec celui situé en deuxième position, et ainsi de suite. Les programmes seront écrits en OCaml dans un fichier `tri_insertion.ml`

1. L'expression `Random.int a` renvoie un entier pris au hasard entre `0` (inclus) et `a` (exclu). Écrire une fonction

   ```
   genere_tab_alea : int -> int -> int -> int array
   ```

   qui renvoie un tableau de `n` entiers compris entre `a` (inclus) et `b` (exclu). Par exemple, `genere_tab_alea 1 8 10` renvoie 10 entiers compris entre 1 et 8 (exclu).
2. Écrire une fonction `echange : int array -> int -> int -> unit` telle que `echange tab i j` permute les éléments de rang `i` et `j` du tableau `tab` .
3. Écrire une fonction `idx_min : int array -> int -> int -> int` qui renvoie l'indice du plus petit élément situé entre les indices `i` et `j` (inclus) du tableau `tab`.
4. Écrire une fonction itérative `tri_insertion : int array -> unit` qui trie le tableau `tab` avec la méthodologie du tri par insertion.
5. Quelles sont les complexités temporelles au pire et au mieux de cette fonction ?

Exercice 2 (Listes OCaml (BERLIOZ, BERNUCHON, BRICARD)).
L'objet de cet exercice est la manipulation de listes à l'aide de fonctions récursives.
On pourra utiliser `List.hd lst` qui renvoie l'élément situé en tête de la liste `lst`, `List.tl lst` qui renvoie la queue de la liste.
Si vous le souhaitez, vous pourrez également utiliser `List.rev lst` qui renvoie la liste renversée. C'est l'équivalent de la fonction miroir codée en TP.
La fonction `List.length` qui calcule la taille d'une liste n'est pas autorisée. Aucune autre fonction du module `List` n'est autorisée.
Toutes les fonctions de cet exercice seront codées dans un fichier `exercices-listes2.ml`.

1. Écrire une fonction `maxi_lst : int list -> int` qui renvoie le maximum d'une liste d'entiers.
2. Écrire une fonction `square_lst : int list -> int list` qui renvoie une liste dont les termes sont les carrés d'une liste d'entiers passée en argument.
3. Écrire une fonction `shuffle : int list -> int list -> int list` qui reçoit deux listes d'entiers et qui renvoie une liste formée des éléments des deux listes entrelacés en finissant par les éléments de la liste la plus longue. Par exemple, `shuffle [1;3;5] [2;4;6;8;10]` renvoie `[1;2;3;4;5;6;8;10]`
4. Écrire une fonction `split : int -> int list -> int list * int list` qui reçoit un entier `n`, une liste d'entiers et qui renvoie deux listes, la première étant formée des `n` premiers éléments de la liste, la seconde étant formée des éléments restants. Si la liste contient moins de `n` éléments, le résultat sera formé de la liste initiale et d'une liste vide. L'ordre des éléments de la liste initiale doit être préservé dans les deux listes renvoyées. Par exemple `split 3 [1;2;3;4]` renvoie `([1;2;3], [4])` .