TP de Programmation Fonctionnelle

On considère la définition de type `Ocaml` suivante qui permet de représenter des arbres binaires étiquetés par des entiers :

```
# type arbre = Vide | Noeud of int * arbre * arbre ;;
```

Voici quelques exemples de valeurs de ce type :

```
# Vide ;;
# Noeud(8,Vide,Vide) ;;
# Noeud(1,Vide,Vide) ;;
# Noeud(5,Noeud(1,Vide,Vide),Noeud(8,Vide,Vide)) ;;
# Noeud(8,Vide,Noeud(1,Vide,Vide)) ;;
```

On rappelle qu'un Arbre Binaire de Recherche (ABR) est un arbre binaire dont les nœuds sont étiquetés par des entiers et qui possède les propriétés suivantes :

i) l'étiquette de la racine est supérieure à toutes les étiquettes du sous-arbre gauche et inférieure à toutes les étiquettes du sous-arbre droit.

ii) tout sous-arbre de cet arbre est lui aussi un ABR.

Dans les exemples de valeurs `Ocaml` de type `arbre` donnés plus haut, les quatre premiers sont des ABR mais pas le dernier. Dans toute la suite du TP, en revanche, on ne considérera que des ABR.

1) Définir une fonction `appartient` de type `int -> arbre -> bool` qui indique si l'entier appartient ou non à l'ABR.

2) Définir une fonction `arbre_2_liste` de type `arbre -> int list` qui transforme un ABR en une liste triée.

3) Un première manière d'insérer un entier `n` dans un ABR `a` qui ne le contient pas consiste à parcourir `a` jusqu'à l'endroit où devrait se trouver `n` et à l'y insérer.

Définir une telle fonction `inserer_bas` qui aura pour type `int -> arbre -> arbre`.

4) Un deuxième manière d'insérer un entier `n` dans un ABR `a` qui ne le contient pas consiste à :

i) couper d'abord `a` en deux sous-arbres distincts `g` et `d` qui contiennent respectivement les éléments inférieurs à `n` et les éléments supérieurs à `n`.

ii) construire ensuite un nouvel ABR de racine `n`, de fils gauche `g` et de fils droit `d`.

Définir d'abord une fonction `couper` de type `int -> arbre -> arbre * arbre` qui réalise **i)**.

Définir ensuite une fonction `inserer_haut` de type `int -> arbre -> arbre` qui réalise **ii)**.

5) Déduire de tout ce qui précède une fonction de tri de type `int list -> int list` qui utilise un ABR. (On construira d'abord un ABR à partir de la liste initiale en utilisant l'une ou l'autre des deux fonctions d'insertion avant de transformer cet ABR en une liste triée grâce à `arbre_2_liste`).