Partie I : fonctions sur les arbres binaires de recherche

Pour implémenter des arbres binaires de recherche, on utilise le type suivant :
```
type abr = Vide | Noeud of int * abr * abr ;;
```

Question 1
Ecrire des fonctions `hauteur : abr -> int` et `taille : abr -> int` calculant la hauteur et la taille d'un arbre binaire de recherche.

Question 2
Ecrire une fonction d'insertion de noeud dans un arbre binaire de recherche. Elle aura la signature `insertion : int -> abr -> abr`.

Question 3
Ecrire une fonction de recherche d'un entier dans un arbre binaire de recherche. Elle aura la signature `appartient : int -> abr -> bool`.

Question 4
Ecrire une fonction de suppression d'un entier dans un arbre binaire de recherche. Elle aura pour signature `supprime : int -> abr -> abr`.

Question 5
Donner la complexité des fonctions précédentes.

Question 6
Ecrire des fonctions `infixe : abr - list`, `prefixe : abr -> list` et `postfixe : abr -> list` renvoyant la liste obtenue par le parcours infixe, préfixe ou postfixe d'un arbre binaire de recherche.

Partie II : arbres équilibrés

Pour tout arbre binaire $\mathcal{A}$, on note $h(\mathcal{A})$ sa hauteur. Pour tout noeud x, on note $h(x)$ la hauteur de l'arbre enraciné en x. Pour tout noeud x de sous-arbre gauche $\mathcal{G}$ et de sous-arbre droit $\mathcal{D}$, on appelle déséquilibre de x la valeur $\delta(x) = h(\mathcal{G}) - h(\mathcal{D})$.

Un arbre est dit AVL (initiales de leur inventeur : Adelson-Velsky et Landis) si le déséquilibre de tout noeud est -1, 0 ou 1.

On considère le type
```
type avl = Vide | Noeud of (int*int)*avl*avl ;;
```

Question 6
En notant $n_{min}(h)$ la taille minimale d'un arbre AVL de hauteur h, trouver une relation de récurrence d'ordre 2 sur la suite $(n_{min}(h))_{h\in\mathbb{N}}$. En déduire une estimation asymptotique de $n_{min}(h)$.

Les noeuds des arbres binaires seront implémentés par deux valeurs : leur valeur effective et leur hauteur.

On dira qu'un arbre AVL est valide si les hauteurs mémorisées aux différents noeuds sont bien les les hauteurs des noeuds. Par exemple, `Noeud((4,2),Noeud((1,0),Vide,Vide),Vide)` n'est pas valide.

Question 7
Ecrire une fonction vérifiant qu'un arbre AVL est valide.

Question 8
Ecrire des fonctions de recherche, insertion et suppression dans un arbre binaire de recherche représenté par le type `avl` précédent.

Question 9
Quel problème voyez-vous à l'insertion ou à la suppression dans un arbre AVL ?

Partie III : rotations et rééquilibrage

Etant donné un arbre binaire de recherche `Noeud(x,Noeud(y,gg,gd),d)` ,
on lui associe sa rotation droite `Noeud(y,gg,Noeud(x,gd,d))`.

Question 10
Montrer que la rotation droite d'un arbre binaire de recherche est un arbre binaire de recherche.

Question 11
Ecrire une fonction `rotation_d : avl -> avl` effectuant l'opération de rotation droite sur un arbre AVL.

Question 12
Décrire une opération de "rotation gauche" sur un arbre AVL et l'implémenter en Ocaml.

Question 13
Justifier que, lorsqu'une insertion ou une suppression dans un arbre AVL crée un arbre non AVL, il est possible par une ou deux rotations de retrouver un arbre AVL.

Question 14
Ecrire de nouvelles fonctions d'insertion et de suppression sur les arbres AVL permettant d'insérer et de supprimer des noeuds dans un arbre AVL en conservant la structure d'arbre AVL. On pourra utiliser les fonctions des questions 7, 10 et 11.

Question 15
Etudier les complexités de ces nouvelles fonctions d'insertion et de suppression.

Partie IV : tri

Question 16
Utiliser la structure d'arbre binaire de recherche ou d'arbre AVL pour écrire une fonction de tri sur les listes. On insèrera tous les éléments de la liste dans un arbre binaire de recherche et on effectuera un parcours de cet arbre (on pourra réécrire une fonction de parcours pour les arbres AVL).

Question 17
Discuter de la complexité de cette fonction sur les arbres binaires de recherche et sur les arbres AVL. On étudiera la complexité moyenne et la complexité dans le pire des cas (en précisant éventuellement ces pires cas).