```
"""
Author: Pascal Padilla
Source: correction de l'exercice 2 du sujet 17 des épreuves pratiques NSI 2022

Remarque (anecdotique):
    * il y a une pratique à éviter dans Python (cf ligne 109)
"""


# ajout pour les tests unitaires (facultatif)
from doctest import testmod


class Noeud:
    '''
    Classe implémentant un n¿ud d'arbre binaire
    disposant de 3 attributs :
    - valeur : la valeur de l'étiquette,
    - gauche : le sous-arbre gauche.
    - droit : le sous-arbre droit.
    '''
    def __init__(self, v, g, d):
        self.valeur = v
        self.gauche = g
        self.droite = d


class ABR:
    '''
    Classe implémentant une structure
    d'arbre binaire de recherche.
    '''

    def __init__(self):
        '''Crée un arbre binaire de recherche vide'''
        self.racine = None


    def est_vide(self):
        '''Renvoie True si l'ABR est vide et False sinon.'''
        return self.racine is None


    def parcours(self, tab = []):
        '''
        Renvoie la liste tab complétée avec tous les
        éléments de l'ABR triés par ordre croissant.
        '''
        if self.est_vide():
            return tab
        else:
            self.racine.gauche.parcours(tab)

            # ajoute la valeur de la racine au tableau
            tab.append(self.racine.valeur)

            # parcours récursif à droite
            self.racine.droite.parcours(tab)
            return tab


    def insere(self, element):
        '''Insère un élément dans l'arbre binaire de recherche.'''
        if self.est_vide():
            self.racine = Noeud(element, ABR(), ABR())
        else:
            if element < self.racine.valeur:
                self.racine.gauche.insere(element)
            else :
                self.racine.droite.insere(element)


    def recherche(self, element):
```

```
        '''
        Renvoie True si element est présent dans l'arbre
        binaire et False sinon.
        '''
        if self.est_vide():
            # si l'ABR est vide, element ne peut pas s'y trouver !
            return False
        else:
            if element < self.racine.valeur:
                # recherche récursive dans le sous ABR de gauche
                return self.racine.gauche.recherche(element)
            elif element > self.racine.valeur:
                # recherche récursive dans le sous ABR de droite
                return self.racine.droite.recherche(element)
            else:
                # si element n'est ni > ni < à valeur
                # c'est qu'ils sont égaux !
                return True


# tests de l'énoncé avec des assertions:
a = ABR()
a.insere(7)
a.insere(3)
a.insere(9)
a.insere(1)
a.insere(9)
assert a.parcours() == [1, 3, 7, 9, 9]
assert a.recherche(4) == False
assert a.recherche(3) == True


# décommenter pour voir le programme planter ;)
# print(a.parcours())
# print(a.parcours())


# explication : la fonction parcours possède un tableau en argument
# qui a une valeur par défaut. Et ça, c'est pas bien :
# NE PAS donner à un TABLEAU en argument une valeur initiale
#
# en effet, après une première exécution de la méthode
# 'parcours', la variable tab n'est plus jamais effacée
# et donc
# lors d'un deuxième appel, les éléments du tableau sont ajoutés
# au tab déjà créé
# pour corriger cela, il faut faire en sorte que la valeur
# initiale donnée à tab soit ignorée. Pour cela, il suffit d'appeler
# la fonction parcours de la façon suivante: 'a.parcours([])'.
```