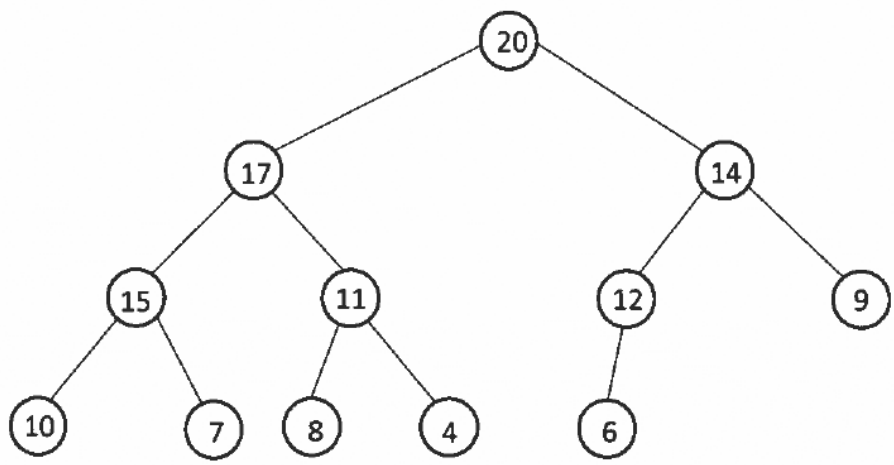
Exercice 2 (5 points)

Cet exercice porte sur les arbres binaires, notamment les arbres binaires de recherche et sur la programmation en langage Python.

Partie A

Dans cet exercice, on considère que la hauteur de l'arbre vide est 0.

On considère l'arbre binaire suivant, noté `A` dans la suite de l'exercice.



L'arbre binaire A

1. Déterminer, sans justifier, la hauteur et la taille de l'arbre binaire `A`.

2. On utilise l'interface suivante pour un arbre binaire :

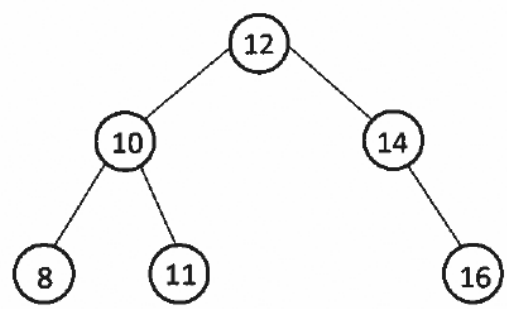
Structure de données abstraite : Arbre Binaire
Utilise : `Entier, Booléen`
Opérations : • `arbre_vide()` vaut un arbre vide • `est_vide(A)` vaut `True` si l'arbre binaire `A` est vide ou `False` sinon. • `cons(v, SAG, SAD)` vaut un arbre binaire dont la valeur de la racine est `v`, le sous-arbre gauche `SAG` et le sous-arbre droit `SAD`. • `gauche(A)` vaut le sous arbre-gauche de l'arbre binaire `A`. • `droit(A)` vaut le sous arbre-droit de l'arbre binaire `A`. • `racine(A)` vaut la valeur du nœud racine de l'arbre binaire `A`.

Cette interface a été implémentée en langage Python.

a) Dessiner l'arbre binaire obtenu avec l'instruction suivante :

```
1 cons(7,
2       cons(4,
2            cons(2, arbre_vide(), arbre_vide()),
3            cons(6,
4                 cons(5, arbre_vide(), arbre_vide()),
5                 arbre_vide())),
6       cons(9,
7            arbre_vide(),
8            cons(8, arbre_vide(), arbre_vide())))
```

b) Déterminer la ou les instructions en langage Python permettant d'obtenir l'arbre binaire suivant :



3.

a) Déterminer l'ordre des valeurs obtenu lorsqu'on parcourt l'arbre `A` à l'aide d'un parcours infixe.

b) La fonction `parcours` incomplète ci-après prend en argument un arbre binaire et renvoie un tableau contenant les valeurs de l'arbre, dans l'ordre obtenu lors d'un parcours infixe.

```
1 def parcours(arbre):
2       if est_vide(arbre):
3            return []
4       else:
5            ...
```

Déterminer parmi les trois propositions suivantes, quelle instruction doit-être écrite à la ligne 5.

Proposition 1 :

```
return [racine(arbre)] + parcours(gauche(arbre)) +
          parcours(droit(arbre))
```

Proposition 2 :

```
return parcours(gauche(arbre)) + [racine(arbre)] +
          parcours(droit(arbre))
```

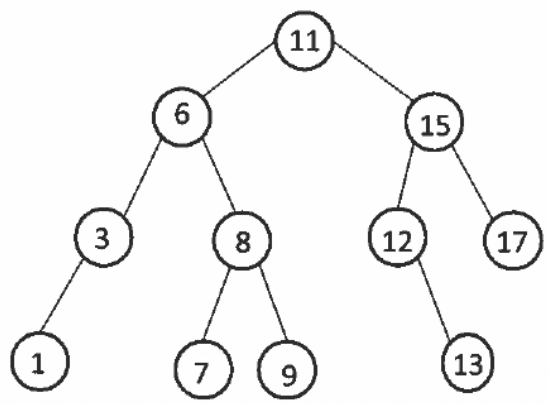
Proposition 3 :

```
return parcours(gauche(arbre)) +
          parcours(droit(arbre)) + [racine(arbre)]
```

Partie B

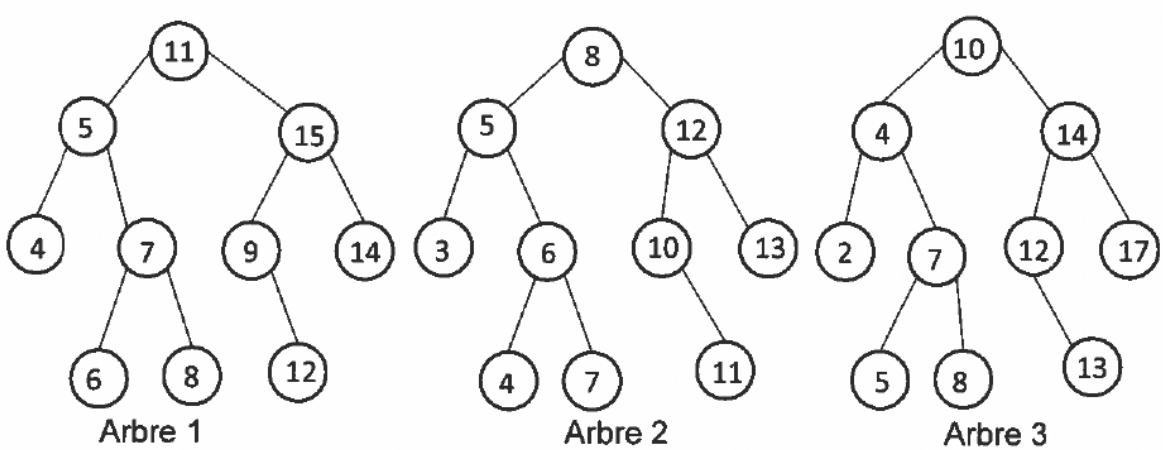
Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud possède une valeur, telle que chaque nœud du sous arbre gauche ait une valeur inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous arbre droit ait une valeur supérieure à celle-ci.

Par exemple, l'arbre suivant, noté `B` dans la suite de l'exercice, est un arbre binaire de recherche.



L'arbre binaire de recherche B

4. Déterminer en justifiant, parmi les arbres suivants, ceux qui ne sont pas des arbres binaires de recherche



5. Déterminer, en justifiant, quel parcours d'un arbre binaire de recherche permet d'obtenir les valeurs de l'arbre classées par ordre croissant.

6. On ajoute à l'interface des arbres binaires une fonction `inserer_dans_ABR` qui prend en arguments un arbre binaire de recherche et une valeur, et qui ajoute une

feuille contenant la valeur, à sa place dans l'arbre binaire de façon à ce que la structure d'arbre binaire de recherche soit toujours respectée.

a) Dessiner l'arbre `B` obtenu après exécution de l'instruction suivante : `inserer_dans_ABR(B, 16)`

b) Dessiner l'arbre binaire de recherche que l'on obtient, en partant de l'arbre vide, et en insérant les valeurs suivantes dans cet ordre :
5, 3, 7, 8, 4, 1, 9, 2 et 6.

7. On veut écrire une fonction `tri` qui prend en argument un tableau de nombres `T` et qui renvoie un tableau trié en utilisant l'algorithme suivant :

- on construit un arbre binaire de recherche en insérant, à partir d'un arbre vide, les éléments du tableau `T` ;
- on construit le tableau que l'on obtient en parcourant cet arbre binaire de recherche grâce à la fonction `parcours` de la partie A.

Écrire la fonction `tri` en Python, en utilisant cet algorithme :

```
1 def tri(T):
2   """ Trie le tableau de nombres T."""
```