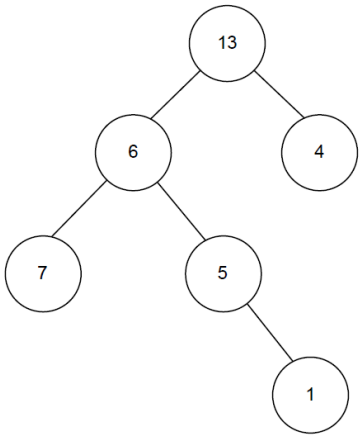
EXERCICE 1 (4 points)

Cet exercice porte sur les arbres binaires, les arbres binaires de recherche, la programmation orientée objet et la récursivité.

1. On considère l'arbre ci-dessous :



a. Justifier que cet arbre est un arbre binaire.

b. Indiquer si l'arbre ci-dessus est un arbre binaire de recherche (ABR). Justifier votre réponse.

2. On considère la classe `Noeud`, nous permettant de définir les arbres binaires, définie de la manière suivante en Python :

```
1   class Noeud:
2       def __init__(self, g, v, d):
3           """crée un noeud d'un arbre binaire"""
4           self.gauche = g
5           self.valeur = v
6           self.droit = d
```

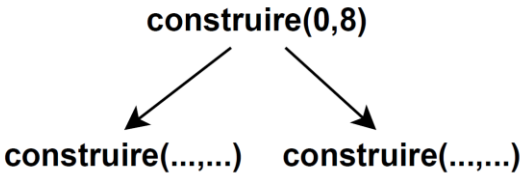
On considère également la fonction `construire` ci-dessus qui prend en paramètre deux entiers `mini` et `maxi` et qui renvoie un arbre.

```
1   def construire(mini, maxi):
2       """mini, maxi:  entiers respectant mini <= maxi"""
3       assert isinstance(mini, int) and isinstance(...,...) and ...
4       if maxi - mini == 1 or maxi - mini == 0:
5           return Noeud(None, mini, None)
6       elif maxi - mini == 2:
7           return Noeud(None, (mini+maxi)//2, None)
8       else:
9           sag = construire(mini, (mini+maxi)//2)
10          sad = construire((mini+maxi)//2, maxi)
11          return Noeud(sag, (mini+maxi)//2, sad)
```

La fonction `isinstance(obj, t)` renvoie `True` si l'objet `obj` est du type `t`, sinon `False`.

a. Recopier et compléter sur votre copie la ligne 3 de l'assertion de la fonction `construire` de manière à vérifier les conditions sur les paramètres `mini` et `maxi`.

b. On exécute l'instruction `construire(0,8)`. Déterminer quels sont les différents appels récursifs de la fonction `construire` exécutés. On représentera ces appels sous forme d'une arborescence, par exemple :



.....

c. Dessiner l'arbre renvoyé par l'instruction `construire(0,8)`.

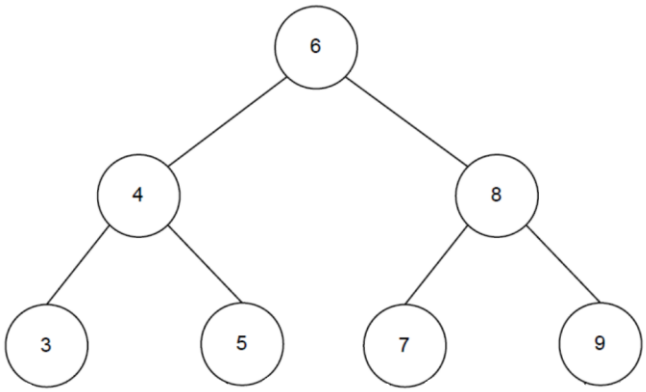
d. Dessiner l'arbre renvoyé par l'instruction `construire(0,3)`.

e. Donner le résultat d'un parcours infixe sur l'arbre obtenu à la question 2.c. Expliquer pourquoi ce parcours permet d'affirmer que l'arbre est un arbre binaire de recherche.

f. La fonction récursive `maximum` ci-dessous prend en paramètre un arbre binaire de recherche `abr` et renvoie la valeur maximale de ses nœuds. Recopier et compléter les lignes 5 et 7 de cette fonction.

```
1   def maximum(abr):
2       if abr is None:
3           return None
4       elif abr.droit is None:
5           return .........
6       else :
7           return .........
```

3. On donne l'arbre binaire de recherche `abr_7_noeuds` suivant :



On donne également ci-dessous la fonction `mystere` qui prend en paramètres un arbre binaire de recherche `abr`, un entier `x` et une liste `liste`.

```
1   def mystere(abr, x, liste):
2       """
3       abr -- arbre binaire de recherche
4       x -- int
5       liste -- list
6       Renvoie -- list"""
7       if abr is None:
8           return []
9       else:
10          liste.append(abr.valeur)
11          if x == abr.valeur:
12              return liste
13          elif x < abr.valeur:
14              return mystere(abr.gauche, x, liste)
15          else:
16              return mystere(abr.droit, x, liste)
```

a. Donner les résultats obtenus lorsque l'on exécute les instructions `mystere(abr_7_noeuds,5,[])`, puis `mystere(abr_7_noeuds,6,[])` puis `mystere(abr_7_noeuds,2,[])`.

b. Décrire quel peut être le rôle de la fonction `mystere`.