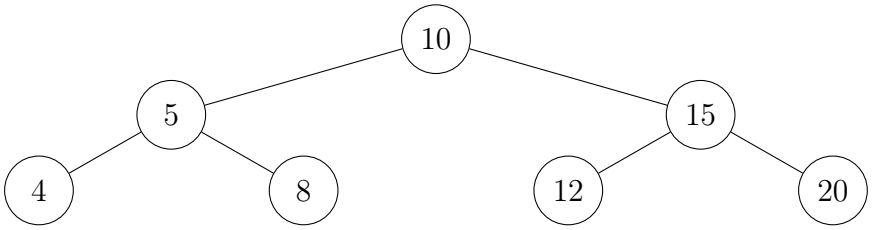
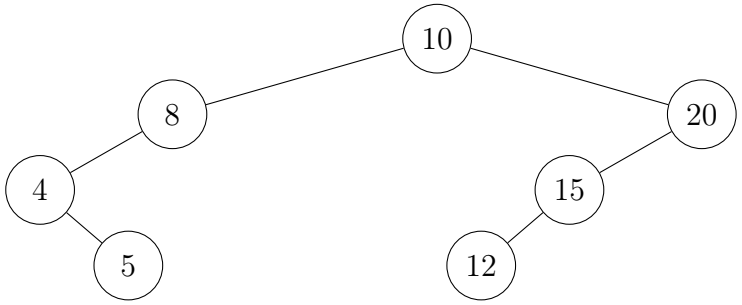
Devoir surveillé n°6 – NSI
Correction

Exercice 1

1. La taille de l'arbre est 7. Sa hauteur est 4.
2. Un arbre bien construit avec les mêmes clé serait :



   L'idée est de choisir à chaque niveau la valeur médiane parmi celles qui restent à placer.
3. La représentation de l'arbre est la suivante :



4. La méthode `hauteur` de la classe `Arbre` est la suivante :

```
def hauteur(self):
    return self.racine.hauteur()
```

5. Voici la méthode `taille` de la classe `Noeud` :

```
def taille(self):
    if self.gauche == None and self.droit == None :
        return 1
    if self.gauche == None :
        return 1+self.droit.taille()
    elif self.droit == None :
        return 1+self.gauche.taille()
    else :
        return 1 + self.gauche.taille() + self.droit.taille()
```

   Le code suivant est incorrect, car la méthode n'existe que pour les éléments de la classe `Noeud`.

```
def taille(self):
    # Méthode incorrecte
    if self is None:
        return 0
    else:
        return 1 + self.gauche.taille() + self.droit.taille()
        # self.gauche et self.droit pouvant être None,
        # la méthode n'existe pas et n'est donc pas applicable
```

Celle de la classe `Arbre` est alors :

```
def taille(self):
    return self.racine.taille()
```

6. (a) La taille minimale d'un arbre binaire de recherche « bien construit » de hauteur h est $t\_min = 2^{h-1}$.
   En effet, si sa taille t était strictement inférieure, autrement dit si $t \leqslant 2^{h-1} - 1$, on pourrait réarranger les nœuds de l'arbre de sorte qu'il soit de hauteur $h-1$ (mais est-ce évident ?), ce qui impliquerait que l'arbre de départ n'est pas « bien construit ».

   (b) La méthode `bien_construit` demandée est alors la suivante :

```
def bien_construit(self):
    t = self.taille()
    h = self.hauteur()
    return t >= 2**(h - 1)
```

Exercice 2

1. Les valeurs entières que l'on peut stocker dans le nœud fils gauche du noeud04 sont les nombres de 27 à 28 (inclus).
2. (a) La liste est `[26,3,1,15,13,19,42,29,32,30,37]`.

   (b) Il s'agit d'un parcourt préfixe (la racine traitée avant ses fils).
3. Pour afficher les nombres par valeur croissante dans un arbre binaire de recherche, il faut effectuer un parcours en profondeur infixe.
   Ainsi :

```
def Parcours(A): # A est un arbre binaire de recherche
    if not A is None:
        Parcours(A.gauche)
        print(A.cle)
        Parcours(A.droit)
```