Exercice 1 ***8 points***

1. Chaque enregistrement de la relation `Articles` doit mentionner un attribut `Auteur` qui est une clé étrangère de la relation `Auteurs`. Si cette dernière est vide, un SGBD refusera donc tout enregistrement d'un nouvel article, car cela violerait la contrainte de référence : chaque article doit être relié à un auteur unique.

2. Requête à saisir : `INSERT INTO Traitements (article, theme) VALUES  (2, 4)`

3. Requête à saisir : `UPDATE Auteurs SET nom = "Jèraus" WHERE idAuteur = 2`

4. **a.** Le titre des articles parus après le 1^{er} janvier 2022 inclus :

```
SELECT titre
FROM Articles
WHERE dateParution >= 20220101
```

b. Le titre des articles écrits par l'auteur Étienne Zola :

```
SELECT titre
FROM Articles
WHERE auteur = 3
```

c. Le nombre d'articles écrits par l'auteur Jacques Pulitzer (présent dans la table `Auteurs` mais on ne connaît pas son `idAuteur`) :

```
SELECT count(*)
FROM Articles
JOIN Auteurs ON Articles.auteur = Auteurs.idAuteur
WHERE Auteurs.nom = "Pulitzer" AND Auteurs.prenom = "Jacques"
```

d. Les dates de parution des articles traitant du thème « Sport » :

```
SELECT Articles.dateParution
FROM Articles
JOIN Traitements ON Articles.idArticle = Traitements.article
JOIN Themes ON Traitements.theme = Themes.idTheme
WHERE Themes.themes = "Sport"
```

Exercice 2 ***8 points***

PARTIE A : GÉNÉRALITÉS

1. Répartition possible : [26, 4], [17, 13], [15, 11] et [5]. Il faut dans ce cas 4 boîtes.

2. On faut connaître le nombre d'éléments de la liste `repartition`. On suffit donc d'utiliser l'instruction `len(repartition)`.

3. Réponse possible :

```python
def poids_boite(boite):
    poids = 0
    for objet in boite:
        poids += objet

    return poids
```

Partie B : Algorithmes de résolution

B1 : Méthode de la première boîte

4. **a.** On obtient la répartition `[8, 2], [3, 1], [9], [7]`.

   b. On pourrait faire `[8, 2], [3, 7], [9, 1]`. On utiliserait alors 3 boîtes au lieu de 4. La méthode de la première position n'est donc pas optimale.

5. Code possible :

```python
def premiere_position(objets, poids_max):

    repartition = [] # la répartition
    repartition.append([]) # on ajoute une boîte vide

    for objet in objets : # parcours des objets
        ajout = False # permet de savoir si l'objet a été ajouté
        for boite in repartition :
            if poids_boite(boite) + objet <= poids_max :
                # l'objet tient dans cette boite
                boite.append(objet) # on l'ajoute
                ajout = True
                break
        if not ajout : # l'objet ne tient dans aucune des premières boîtes...
            repartition.append([objet]) # on l'ajoute dans une nouvelle boîte

    return repartition
```

B2 : Méthode de la meilleure boîte

6. Considérons des objets de poids [8, 1, 9, 2] et un poids maximal de 10. En appliquant la méthode de la meilleure boîte, on obtient la répartition [8, 1], [9], [2].

   Pourtant, il est possible de faire mieux avec la répartition [8, 2], [9, 1], qui ne fait intervenir que deux boîtes.

   La méthode de la meilleure boîte n'est donc pas optimale.

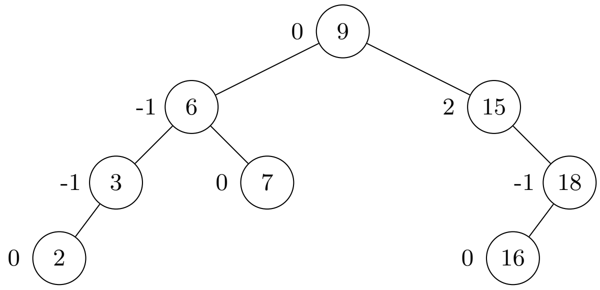
7. Code possible :

```python
# On "remonte" cette boîte à sa position triée
while i > 0 and poids_boite(repartition[i]) > poids_boite(repartition[i-1]) :
    repartition[i], repartition[i-1] = repartition[i-1], repartition[i]
    i = i-1
```

Exercice 3 *8 points*

Partie A

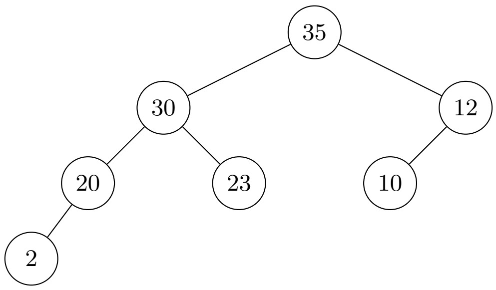
1. a. On obtient :



b. Cet arbre n'est pas équilibré car le nœud de valeur 15 a une balance de 2.

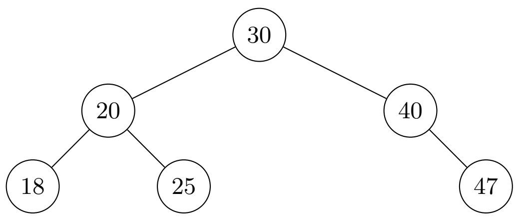
2. a. On obtient `[0, 45, 40, 48, 17, 43, None, 49, 14, 19]`

b. On obtient :



3. a. La fonction `myst` permet de calculer la hauteur d'un arbre. En effet, si l'arbre est vide ou si la valeur de sa racine est `None`, on renvoie 0. Dans le cas contraire, on renvoie 1 plus de maximum des résultats des sous-arbres gauches et droits (indices `2*i` et `2*i+1`). On calcule ainsi la hauteur de l'arbre.

b. `myst(arbre, 1)` renvoie 3, qui est la hauteur de l'arbre



4. Code possible :

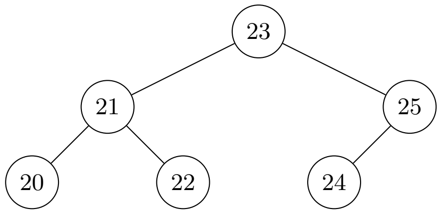
```
def est_equilibre(arbre, i):
   if i >= len(arbre) or arbre[i] is None:
        return True
    else:
        balance = myst(arbre, 2*i+1) - myst(arbre, 2*i)
        reponse = balance in [-1, 0, 1]
        return reponse and est_equilibre(arbre,2*i) and est_equilibre(arbre, 2*i+1)
```

Partie B

5.
- Parcours *préfixe* : 45, 40, 17, 14, 19, 43, 48, 49
- Parcours *infixe* : 14, 17, 19, 40, 43, 45, 48, 49

- Parcours *postfixe* : 14, 19, 17, 43, 40, 49, 48, 45

6. On obtient :



7. Code possible :

```python
def infixe(arbre):
    pile = []
    visites = []
    n = 1
    repetition = True
    while repetition :
        while n < len(arbre) and arbre[n] is not None :
            pile.append(n)
            n = 2*n
        if len(pile) == 0 :
            repetition = False
        else :
            n = pile.pop()
            visites.append(arbre[n])
            n = 2*n+1
    return visites
```

8. Code possible :

```python
def construire_ABR(i, ordre):
    while len(nouveau) != i+1:
        nouveau.append(None)

    i_milieu = len(ordre) // 2
    nouveau[i] = ordre[i_milieu]

    gauche = ordre[:i_milieu]
    if gauche != []:
        construire_ABR(2*i, gauche)

    droite = ordre[(i_milieu+1):]
    if droite != []:
        construire_ABR(2*i+1, droite)
```