Proposition de correction

Exercice 1

Q1

Deux clés primaires sont identiques pour les enregistrements 1 et 3

Q2

La clef étrangère sur la relation Eleves doit d'abord être définie.

Q3

SELECT titre FROM Livre WHERE auteur = 'Molière'

Q4

Donne le nombre d'élèves de la classe de T2.

Q5

UPDATE Emprunts SET dateRetour = '2020-09-30' WHERE idEmprunt = 640

Q6

Renvoie les nom et prénom de la classe de T2 qui ont emprunté un livre.

Q7

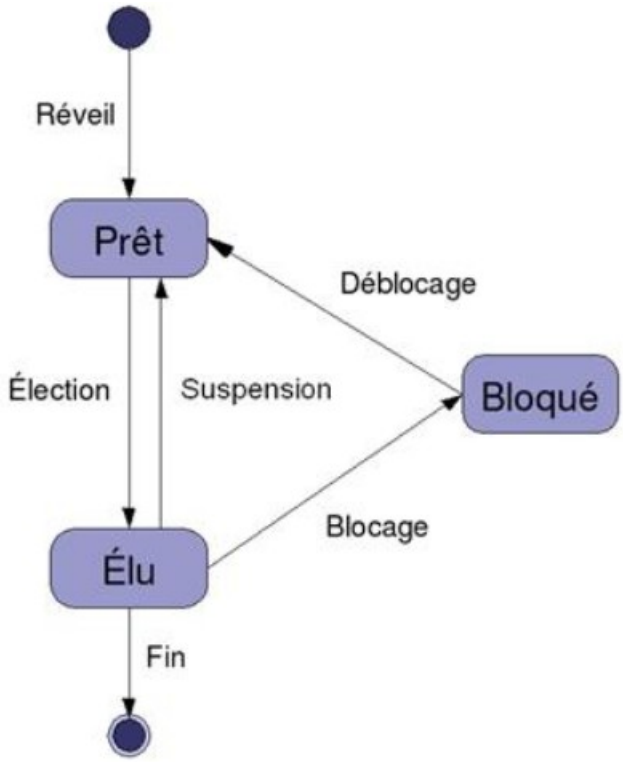
SELECT Eleves.nom, Eleves.prenom
FROM Eleves, Emprunts, Livres
WHERE Livres.titre = 'Les misérables' AND Livres.isbn = Emprunts.isbn AND Emprunts.idEleve = Eleves.idEleve

Exercice 2

Q1a

Le processus est en exécution (actif).

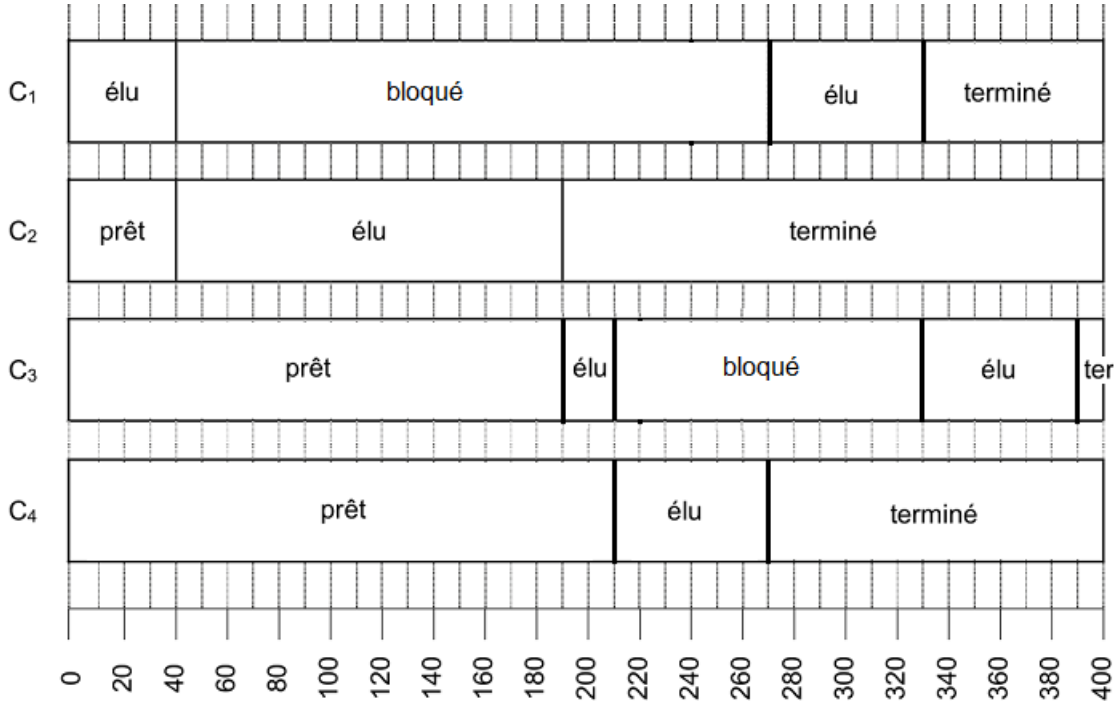
Q1b



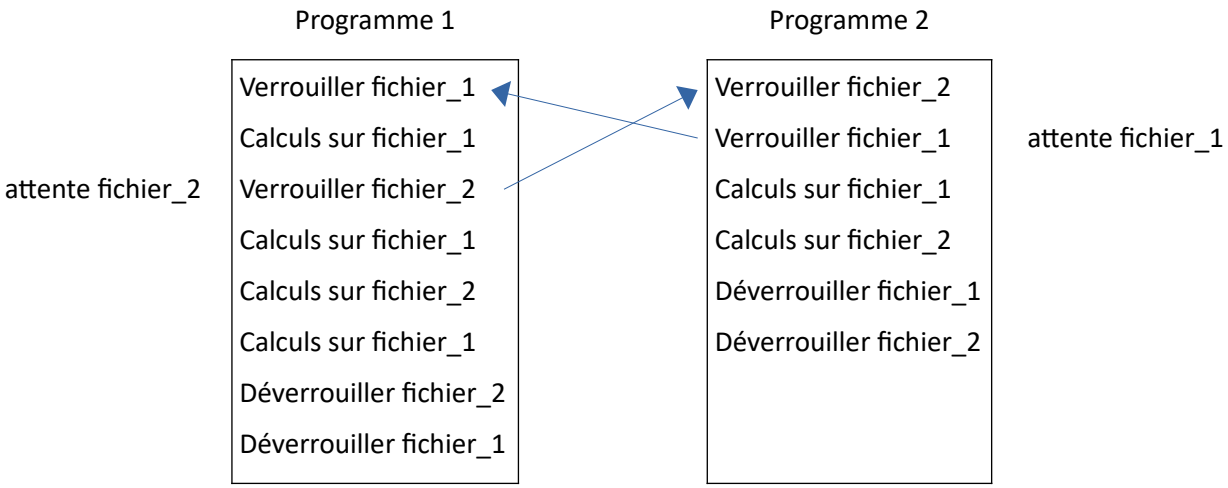
Q2a

Premier entré, premier sorti (FIFO)

Q2b



Q3a



Supposons l'ordonnancement suivant :

- Verrouiller fichier_1 (P1)
- Calculs sur fichier_1 (P1)
- Verrouiller fichier_2 (P2)
- Verrouiller fichier_1 (P2) → en attente de libération
- Verrouiller fichier_2 (P1) → en attente de libération

Les 2 programmes s'attendent mutuellement : il y a interblocage (deadlock)

Q3b

Verrouiller fichier_1

Verrouiller fichier_2

Calculs sur fichier_1

Calculs sur fichier_2

Déverrouiller fichier_1

Déverrouiller fichier_2

Exercice 3

Q1a

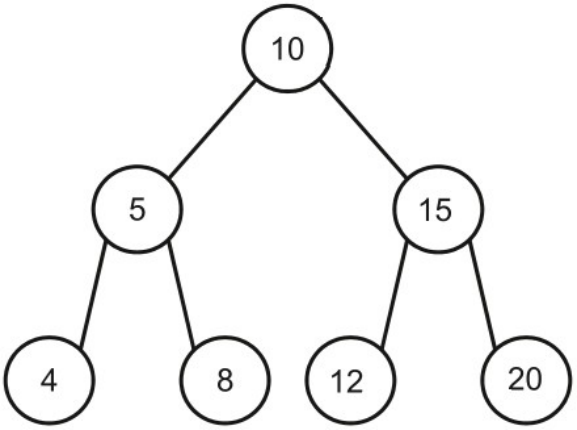
Nombre de nœuds = 7 (taille max pour une hauteur h : $2^h - 1 = 1$).

Q1b

profondeur maximale du nœud racine = 4

NB : par convention, ici un arbre réduit à un seul nœud est de hauteur 1

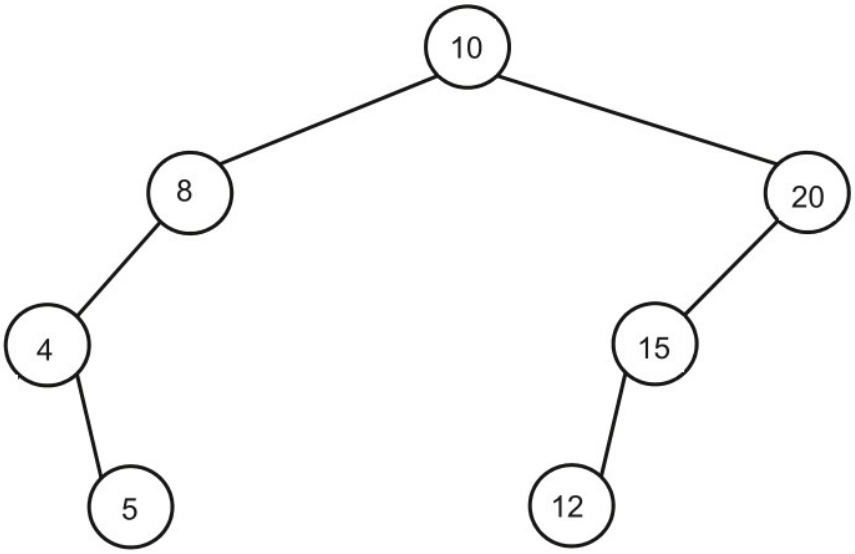
Q2



Taille : 7

Hauteur : 3 (par convention h(Δ) = 1)

Q3



Taille : 7

Hauteur : 4

Q4

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

Q5

Classe Nœud :

```
def taille(self) -> int:
    if self.gauche == None and self.droite == None:
        return 1
    elif self.gauche == None:
        return 1 + self.droite.taille()
    elif self.droite == None:
        return 1 + self.gauche.taille()
    else:
        return 1 + self.gauche.taille() + self.droite.taille()
```

Classe Arbre :

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

Q6a

$2^{(h-1)} - 1 < taille \leq 2^h - 1$

doù $t\_min = 2^{(h-1)} - 1$

Q6b

```
def bien_construit(self) -> bool:
    return 2**(self.racine.hauteur() - 1) - 1 < self.racine.taille() <= 2**self.racine.hauteur() - 1
```

Exercice 4

Q1

La valeur lst[i2] est perdue lorsque le tableau d'indice i2 est affecté avec la valeur lst[i1].

Q2

0, 1, 9, 10

Q3a

A chaque appel de melange(), le paramètre ind est décrémenté de 1. Donc ind → -∞.

La condition de continuation ind > 0 n'est alors plus vérifiée et melange() n'est plus appelée lorsque ind = 0.

Q3b

n

Q3c

list	ind	j
[0, 1, 2, 3, 4]	4	2
[0, 1, 4, 3, 2]	3	1
[0, 3, 4, 1, 2]	2	2
[0, 3, 4, 1, 2]	1	0
[3, 0, 4, 1, 2]	0	

Q3d

```
def melange(lst : list, ind : int) -> list:
    """ mélange de Fischer Yates """
```

```
    for i in range(ind, 0, -1):
        j = randint(0, i)
        lst[i], lst[j] = lst[j], lst[i]
    return lst
```

Exercice 5

Q1a

La liste de tous les éléments du tableau.

Q1b

L'élément max de la liste.

Q2a

```
def somme_sous_sequence(lst : list, i : int, j : int) -> int:
    """ renvoie la somme de la sous-séquence délimitée par les indices i et j (inclus). """
    if 0 <= i < len(lst) and i <= j <= len(lst):
        somme = 0
        for k in range(i, j+1):
            somme += lst[k]
        return somme
    return None
```

Q2b

1/2 n(n+1) = 55, complexité quadratique

Q2c

```
def pgsp(lst : list) -> tuple:
    n = len(lst)
    somme_max, a, b = lst[0], 0, 0
    for i in range(n):
        for j in range(i, n):
            s = somme_sous_sequence(lst, i, j)
            if s > somme_max :
                somme_max, a, b = s, i, j
    return somme_max, a, b
```

Q3a

i	0	1	2	3	4	5	6	7
lst[i]	-8	-4	6	8	-6	10	-4	-4
S(i)	-8	-4	6	14	8	18	14	10

Q3b

```
def pgsp2(lst):
    sommes_max = [lst[0]]
    for i in range(1, len(lst)):
        if sommes_max[i-1] <= 0:
            sommes_max = sommes_max + [ lst[i] ]
        else:
            sommes_max = sommes_max + [ lst[i] + sommes_max[i-1] ]
    return max(sommes_max)
```

Q3c

La complexité de la fonction pgsp() est quadratique.

La complexité de la fonction pgsp2() est linéaire.