2021 métropole candidat libre sujet 2

Exercice 1

1. La troisième instruction utilise la même clé primaire que la première (128) or une clé primaire doit être unique.

2. La clé étrangère idEleve assure l'existence d'une clé primaire identique dans la table Eleves.

3. `SELECT titre FROM Livres WHERE auteur = "Molière";`

4. Elle renvoie le nombre d'élèves dans la classe T2.

5. `UPDATE Emprunts SET dateRetour = "2020-09-30" WHERE idEmprunt = 640`

6. Elle renvoie les nom et prénom (sans doublons) des élèves de la classe T2 ayant emprunté un livre.

7.

```
SELECT nom, prenom
FROM Eleves
JOIN Emprunts on Eleves.idEleve = Emprunts.idEleve
JOIN Livres ON Emprunts.isbn = Livres.isbn
WHERE titre = "Les misérables";
```

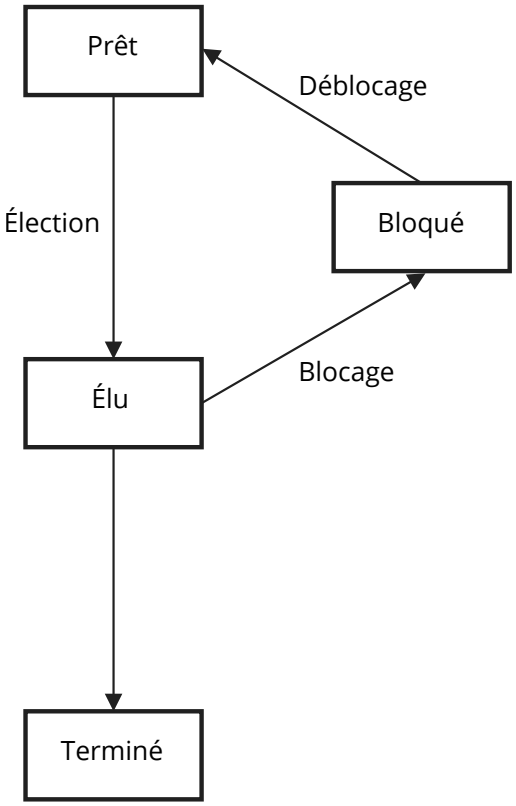
ou

```
SELECT nom, prenom
FROM Eleves
JOIN Emprunts on Eleves.idEleve = Emprunts.idEleve
WHERE isbn = 9782070409228;
```

Exercice 2

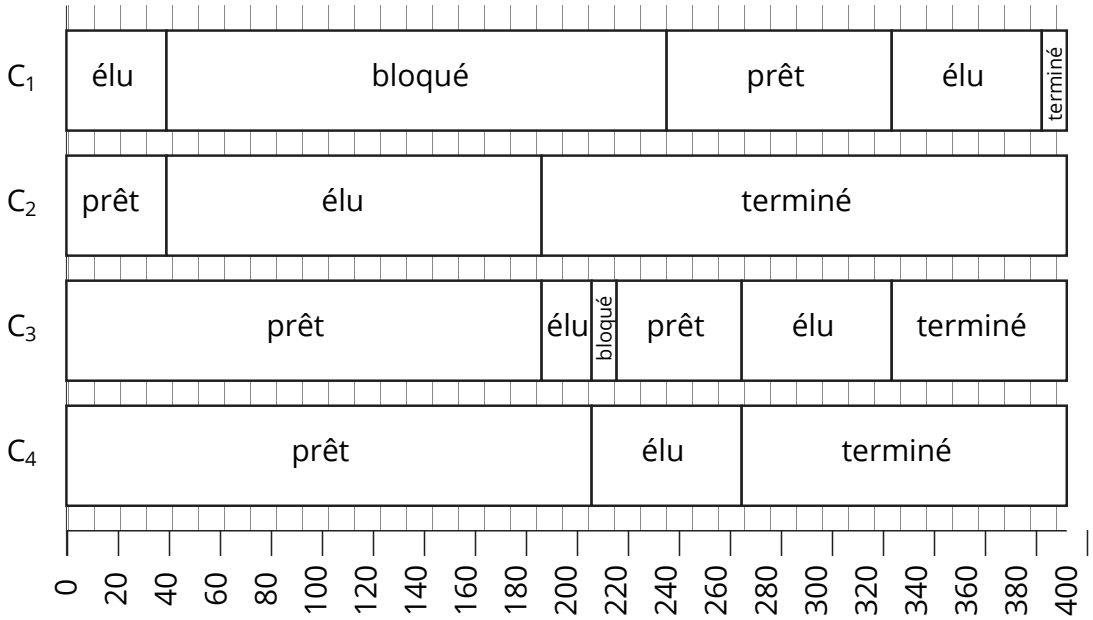
1. a. Un processus est dans l'état élu s'il est exécuté par le processeur.

1. b.



2. a. Premier entré, premier sorti.

2. b.



3. a. Il peut y avoir un interblocage entre les deux programmes. En effet, le programme 1 verrouille le fichier 1 puis a ensuite besoin du ficher 2 alors que le programme 2 verrouille le fichier 2 puis a ensuite besoin du ficher 1. Chaque programme peut être bloqué en attendant la ressource occupée par l'autre programme.

3. b. Une solution à l'interblocage est de libérer les ressources avant d'en occuper une autre. Ainsi le programme 2 peut être modifié comme suit :

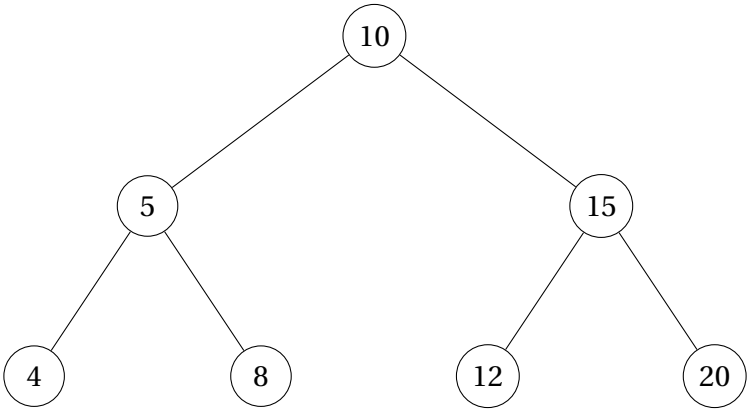
```
Verrouiller fichier_2
Calculs sur fichier_2
Déverrouiller fichier_2
Verrouiller fichier_1
Calculs sur fichier_1
Déverrouiller fichier_1
```

Exercice 3

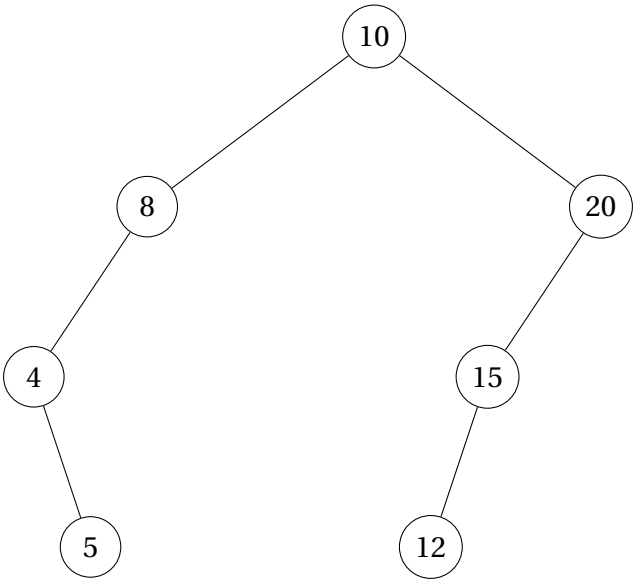
1. a. 7

1. b. 4

2.



3.



4.

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

5.
Pour la classe Nœud :

```
def taille(self):
    t = 1
    if self.gauche is not None:
        t = t + self.gauche.taille()
    if self.droit is not None:
        t = t + self.droit.taille()
    return t
```

Pour la classe Arbre :

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

6. a. Il faut que les niveaux précédents soit pleins $(2^{h-1}-1)$ et qu'il y ait un nœud en plus (1) :

$$t_{\min} = 2^{h-1}$$

6. b.

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

Exercice 4

1. Il faut utiliser une variable pour effectuer un échange. La valeur qui était dans `lst[i2]` est perdue ici.

```
def echange(lst, i1, i2):
    a = lst[i2]
    lst[i2] = lst[i1]
    lst[i1] = a
```

2. 0, 1, 9 et 10.

3. a. La condition d'arrêt est `ind <= 0`. Or l'appel récursif se fait avec le paramètre `ind - 1`. Donc la condition d'arrêt sera nécessairement vraie à un moment.

3. b. $n-1$ car les appels vont de $n-2$ à 0.

3. c.

```
[0, 1, 2, 3, 4]
[0, 1, 4, 3, 2]
[3, 1, 4, 0, 2]
[3, 1, 4, 0, 2]
[1, 3, 4, 0, 2]
```

3. d.

```
def melange_iter(lst):
    n = len(lst)
    for i in range(n-1, 0, -1): # de n-1 à 1
        print(lst)
        j = randint(0, i)
        echange(lst, i, j)
```

Exercice 5

1. a. Il faut prendre tout le tableau.

1. b. Il faut prendre l'élément le plus grand car la sous-séquence ne pas être vide.

2. a.

```
def somme_sous_sequence(lst, i, j):
    s = 0
    for k in range(i, j+1):
        s = s + lst[k]
    return s
```

2. b. C'est la somme des entiers de 1 à 10 : 55.

2. c.

```
def pgsp(lst):
    n = len(lst)
    somme_max = lst[0]
    i_max = 0
    j_max = 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 = s
                i_max = i
                j_max = j
    return somme_max, i_max, j_max
```

3. a.

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

3. b.

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

3. c.
Cette solution utilise la mémoïsation : elle se sert des résultats précédents pour gagner en efficacité. En conséquence, cette solution n'effectuera que 9 comparaisons, elle sera donc plus rapide que la première. Cette solution est en O(n) contrairement à la première qui est en $O(n^2)$.