Exercice 1 : Base de données – Naissances – Patientes – Accouchement

1. a. idMere ne peut pas être une clé primaire de la relation Naissances car la mère peut accoucher plusieurs fois ou avoir des jumeaux (deux naissances avec le même idMere).
1. b. (date, rang) peuvent être une clé primaire parce qu'ensemble elles permettent d'identifier de façon unique chaque enregistrement dans la table.
1. c. (poids, taille) ne peuvent pas être une clé primaire car deux bébés peuvent avoir la même taille et le même poids.

2. Elle provoque une erreur parce que la clé étrangère idMere de la relation Naissances est dépendante de d'attribut NumPatientes de la relation Patientes. Cela enfreint la contrainte de référence.

3.
```
INSERT INTO Patientes
VALUES (13862, 'Belanger', 'Ninette', 'La Rochelle');
```

4.
```
UPDATE Naissances SET prenom = 'Laurette'
WHERE idMere = 13860 AND prenom = 'Lorette';
```

5.
```
SELECT nom, prenom FROM Patientes
WHERE commune = 'Aigrefeuille d'Aunis';
```

6.
```
SELECT AVG(poids) FROM Naissances
JOIN TypesAccouchement ON acc = idAcc
WHERE libelleAcc = 'césarienne';
```

7.

Berthelot	Michelle
Samson	Marine
Baugé	Gaëlle
Baugé	Gaëlle

Exercice 2 : Programmation – Algorithme de tri de patients

1.
```
attente.append((50,4))
```

2. a. L'algorithme de tri implémenté est le tri par sélection.
(En effet, il consiste à rechercher le plus prioritaire dans la liste puis de l'échanger avec le premier de la liste et ainsi de suite)

2. b. La complexité en temps des tris par insertion et par sélection est quadratique $O(n^2)$.
(En effet, il faut parcourir n fois la liste pour chacun des n élément de la liste.)

3. a. (Les crochets semblent indiquer qu'il faut générer la liste par compréhension.)
```
def quitte(attente):
    return [a for a in attente if a[1] != 1]
```

3. b.

Solution 1 :	Solution 2 :

`def maj(attente):`
`  t = []`
`  for a in attente:`
`  t. append((a[0], a[1]-1))`
`  return t`
	`def maj(attente):`
	`  return [(a, b-1) for a, b in attente]`

4. a.
```
def priorite(attente, p):
    for a in attente:
        if a[0] == p:
            return a[1]
```

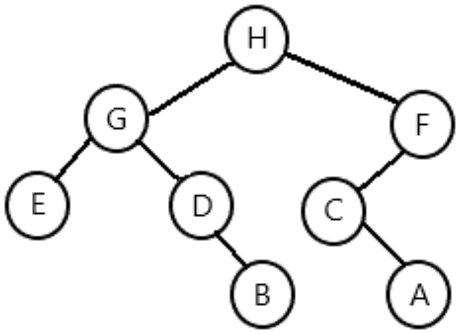
4. b.
```
def revise(attente, p):
    nouvelle = []
    n = priorite(attente, p)
    for (patient, prio) in attente:
        if patient == p:
            nouvelle.append((patient, 1))
        elif prio < n:
            nouvelle.append((patient, prio+1))
        else :
            nouvelle.append((patient, prio))
    return nouvelle
```

Exercice 3 : Arbres binaires – Décisions médicales

1. Sa taille est de 11 et sa hauteur de 5.

2. a. La représentation graphique correspond à l'arbre 2.

2. b. Voir ci-contre :



3. a. Il s'agit du parcours suffixe de l'arbre (les deux sous-arbres avant la racine) :

Affichage ligne par ligne : d, b, g, f, a

3. b.

```
def parcours_maladies(arb):
    if arb == {}:
        pass
    if arb['sag'] == {} and arb['sad'] == {}:
        print(arb['etiquette'])
    else:
        parcours_maladies(arb['sag'])
        parcours_maladies(arb['sad'])
```

4.

```
def symptomes(arb, mal):
    if arb['sag'] != {}:
        symptomes(arb['sag'], mal)

    if arb['sad'] != {}:
        symptomes(arb['sad'], mal)

    if arb['etiquette'] == mal:
        arb['surChemin'] = True
        print('symptômes de', arb['etiquette'], ':')

    else :
        if arb['sad'] != {} and arb['sad']['surChemin']:
            print(arb['etiquette'])
            arb['surChemin'] = True

        if arb['sag'] != {} and arb['sag']['surChemin']:
            print('pas de ', arb['etiquette'])
            arb['surChemin'] = True
```

Exercice 4 : Gestion des processus et des ressources par un système d'exploitation

PARTIE A :

1.

P1	P1	P2	P3	P3	P3	P2	P2	P2	P4	P4	P4	P4	P1	

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15

2.

Processus	Temps d'exécution	Instant d'arrivée	Temps de séjour	Temps d'attente
P1	3	0	14 – 0 = 14	14 – 3 = 11
P2	4	2	9 – 2 = 7	7 – 4 = 3
P3	3	3	6 – 3 = 3	3 – 3 = 0
P4	4	5	13 – 5 = 8	8 – 4 = 4

3. Le temps d'attente d'un processus peut être nul s'il est exécuté dès qu'il arrive et s'il n'est pas interrompu par un autre processus.

PARTIE B :

1. Les processus s'attendent mutuellement car :
L'analyseur d'échantillon attend D4 car D4 est utilisée par le SGBD.
Le SGBD attend D5 car D5 est utilisée par le tableur.
Le tableur attend D1 car D1 est utilisée par l'analyseur d'échantillon.
Le traitement de texte attend D3 car D3 est utilisée par le tableur.

2. Cette situation s'appelle un interblocage (deadlock en anglais).

3. L'analyseur d'échantillon libère D1, donc le tableur peut prendre D1.
Le tableur libère D3, donc le traitement de texte peut prendre D3.
Le tableur ayant aussi libéré D5, le SGBD peut prendre D5.
Le SGBD libère D4 donc l'analyseur d'échantillon peut prendre D4.
L'ordre possible d'exécution est : Tableur – Traitement de texte – SGBD – Analyseur d'échantillon.

Exercice 5 : Réseaux et protocoles de routage

PARTIE A : Adressage

1. Service de radiologie :
L'adresse réseau est : 192.168.1.0 .
Le masque du réseau est 255.255.255.0 (ou /24 en notation CIDR).

2. Les adresses des trois interfaces de réseaux de R5 :
192.168.5.254
175.89.50.254
44.197.5.1

3. a. La 1ère et la dernière adresse IP pouvant être attribuée à une machine sont : 192.168.1.1 à 192.168.1.254.En effet, pour rappel l'adresse 192.168.1.0 est l'adresse du réseau et l'adresse 192.168 .1.255 est l'adresse de diffusion (broadcast).

3. b. Il y a donc 254 machines adressables sur ce réseau.

PARTIE B : Étude du protocole RIP

1. SP → R5 → R1 → R0 → RL R

2. SP → R5 → R4 → R2 → R0 → RL R

PARTIE C : Protocole OSPF

1. Calcul du coût de la liaison R2 vers R3 : C = $10^9/400.10^6$ = 2,5 donc C = 3.

2. Bande Passante = 10^9 / 5 = 2.10^8 b/s = 200Mb/s
(Toutes les valeurs entre 200Mb/s et 10^9 / 4 = $2,5.10^8$ b/s = 250Mb/s sont possibles.)

3. SP → R5 → R4 → R2 → R1 → R0 → RL R
pour un coût total C = 1 + 4 + 1 + 2 = 8.

4. SP → R5 → R4 → R3 → R0 → RL R
pour un coût total C = 1 + 5 + 4 = 10.