Proposition de correction

Exercice 1

Q1

```
def echange(tab : list, i : int, j : int):
    """ réalise sur place l'échange des valeurs dans tab
    @param tab -- une liste de valeurs
    @param i, j --  deux indices i, j
    """
    if (i >= 0) and (j < len(tab)):
        tab[i], tab[j] = tab[j], tab[i]
```

Q2

```
def triStooge(tab, i, j):
    if tab[i] > tab[j]:
        echange(tab, i, j)
    if (j - i) > 1:
        k = (j - i + 1) // 3
        triStooge(tab, i, i+k)
        triStooge(tab, i+k, j-k)
        triStooge(tab, j-k, j)
```

Q3

récursif : la fonction s'appelle elle même

Q4

k = (5 – 0 + 1) // 3 = 6 // 2 = 3

Q5

3 + (3 x 3) + (3 x 3 x 3) = 39

Q6

1 : triStooge(A, 1, 4)

2 : triStooge(A, 2, 3)

3 : triStooge(A, 3, 5)

Q7

Appel	Valeur de A avant l'appel	Valeur de A après l'appel
triStooge(A, 0, 3)	[5, 6, 4, 2]	[2, 4, 5, 6]
triStooge(A, 1, 3)	[2, 6, 4, 5]	[2, 4, 5, 6]
triStooge(A, 1, 3)	[2, 4, 6, 5]	[2, 4, 5, 6]
triStooge(A, 0, 0)	[2, 4, 5, 6]	[2, 4, 5, 6]

Q8

Tri par fusion, complexité en O(n.ln2(n)) < O(n^(8/3)), n > 0

Exercice 2

Q1

Dufour Marc

Martin Sophie

Q2

SELECT nom_medic

WHERE prix < 3

ORDER BY nom_medic

Q3

INSERT INTO client

VALUES(1, 'Durand', 'Nathalie', 269054958815780)

Q4

- id_client : permet de faire la relation avec la table client
- id_medic : permet de faire la relation avec la table medicament

Q5

- 2 x 3 = 6 < 8 : 1 boite
- 1 x 4 x 7 = 28 < 3 x 10 : 3 boites

Q6

UPDATE medicament

SET quantite = quantite – 3

WHERE id_medic = 4

Q7

1 x 3,50 = 3,50€

3 x 5,50 = 16,50€

soit 20,00€

Q8

SELECT medicament.nom_medic

FROM medicament, ordonnance

WHERE ordonnance.id_ordo = 6

AND medicament.id_medic = ordonnance.id_medic

Exercice 3

Partie A

Q1

192.168.1.3/24

1 octet pour attribution @IP, soit 192.168.1.0/24 à 192.168.1.255/24 en éliminant @RZO, @BC et @IP déjà attibuées

Q2

- [‘Alice’, ‘Charlie’, 10]
- [‘Bob’, ‘Alice’, 5]

Q3

La méthode creer_bloc_0() initialise le bloc précédent à None

Q4

Blockchain().creer_bloc_0()

Q5

```
ma_blockchain = Blockchain()

liste_transactions = [
   Transaction('Alice', 'Charlie', 50),
   Transaction('Charlie', 'Bob', 30)
   ]

ma_blockchain.tete = Bloc(liste_transactions, ma_blockchain.tete)

liste_transactions = [
   Transaction('Bob', 'Charlie', 20),
   Transaction('Bob', 'Charlie', 20),
   Transaction('Charlie', 'Alice', 30)
   ]
```

```
ma_blockchain.tete = Bloc(liste_transactions, ma_blockchain.tete)
```

Q6

100 + 30 – 20 – 20 = 90

Q7

```
def ajouter_bloc(self, liste_transactions : list) :
    self.tete = Bloc(liste_transactions, self.tete)
```

Q8

adresse de broadcast : 192.168.1.255

Q9

```
def calculer_solde(self, utilisateur : str) -> float:
    if self.bloc_precedent is None: # cas de base
        solde = 0
    else:
        # appel récursif : calcul du solde au bloc précédent
        solde = self.bloc_precedent.calculer_solde(utilisateur)

    for transaction in self.liste_transactions:
        if transaction.expediteur == utilisateur:
            solde = solde - transaction.montant
        elif transaction.destinataire == utilisateur:
            solde = solde + transaction.montant

    return solde
```

Q10

ma_blockchain.tete.calculer_solde("Alice")

Partie B

Q11

Recherche de toutes les valeurs possibles

Q12

"0" puisque bloc_precedent vaut None

Q13

2^256

Q14

```
def minage_bloc(self) -> str:
    """
    modifie le nonce d'un bloc pour que son hash commence par '00'
    énumérant tous les entiers naturels en partant de 0.
    """
```

```
        self.nonce = 0
        self.hash  = self.calculer_hash()
        while self.hash[:2] != "00" :
            self.nonce = self.nonce + 1
            self.hash  = self.calculer_hash()
        return self.hash
```