Exercice 1 :

Partie A : Réseau dans un lycée

1. Il reste un octet soit 8 bits pour l'adresse réseau (1re adresse), les machines et l'adresse de broadcast (dernière adresse). Ainsi le réseau peut accueillir : $2^8 - 2 = 254$ machines.

2. Conversion décimal – binaire : $217_{(10)} = 128 + 64 + 16 + 8 + 1 = 1101\ 1001_{(2)}$

3. Conversion binaire – décimal : $110010_{(2)} = 32 + 16 + 2 = 50_{(10)}$

4. L'adresse du sous-réseau de la machine 110.217.53.22/24 est 110.217.53.0 donc cette machine n'appartient pas au réseau pédagogique 2.
[Le réseau pédagogique 2 (110.217.52.0/24) admet une plage d'adresse de 110.217.52.0 à 110.217.52.255.]

5. Report des adresses sur le schéma du réseau :

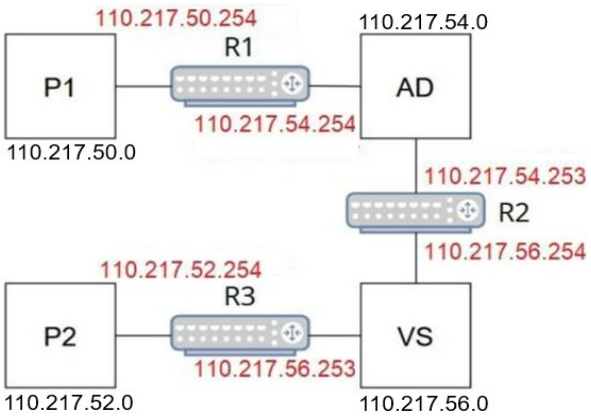

Table de routage du routeur R1 :

Destination	Passerelle	Interface
110.217.50.0	on-link	110.217.50.254
110.217.52.0	110.217.54.253	110.217.54.254
110.217.54.0	on-link	110.217.54.254
110.217.56.0	110.217.54.253	110.217.54.254

6. Table de routage du routeur R1 :

Destination	Passerelle	Interface
110.217.50.0	on-link	110.217.50.254
110.217.52.0	110.217.50.253	110.217.50.254
110.217.54.0	on-link	110.217.54.254
110.217.56.0	110.217.54.253	110.217.54.254

7. Le nombre de saut étant identique, aucune modification n'est nécessaire dans la table de routage du routeur R2.

Partie B : Réseaux et graphes

8. Qu'il y ait ou non un chemin reliant les deux sommets, la fonction risque d'empiler les appels récursifs jusqu'à atteindre la limite de récursion car il n'y a pas de mémoire des sommets déjà visités.

Exemple d'empilement d'appel sans fin pour une graphe avec au moins deux sommets A et B liés entre eux :
`recherche("A", "E")`
appelle `recherche("B", "E")`
qui appelle `recherche("A", "E")`
et ainsi de suite ...

8. Proposition de solution du problème :

```
def recherche_v2(R1, R2, deja_vu=None):
    if deja_vu is None:
        deja_vu = []

    if R1 == R2:
        return True

    if R1 not in deja_vu:
        deja_vu.append(R1)

    for S in adjacents(R1,G):
        if S not in deja_vu:
            if recherche_v2(S, R2, deja_vu):
                return True

    return False
```

Exercice 2 :

1. Il faut savoir si le score est un multiple de 3 :

```
def possible_avec_penalites_seules(score : int) -> bool:
    return score%3 == 0
```

2. Tableau des solutions possibles pour chaque score de 0 à 10 :

score	liste de solution	nombre de solutions
0	[0]	1
1	[]	0
2	[]	0
3	[0, 3]	1
4	[]	0
5	[0, 5]	1
6	[0, 3, 6]	1
7	[0, 7]	1
8	[0, 5, 8], [0, 3, 8]	2
9	[0, 3, 6, 9]	1
10	[0, 3, 10], [0, 5, 10], [0, 7, 10]	3

3. Appliquons $f(n) = f(n-3) + f(n-5) + f(n-7)$ pour n = 10 :
$f(10) = f(7) + f(5) + f(3) = 1 + 1 + 1 = 3$

4. Cas de base de cette fonction récursive :
$f(0) = 1$ $f(1) = 0$ $f(2) = 0$ $f(3) = 1$ $f(4) = 0$ $f(5) = 1$ $f(6) = 1$

5.

```
def nb_solutions(n : int) -> int:
    if n < 0:
        return 0
    elif n in [0, 3, 5, 6]:
        return 1
    elif n in [1, 2, 4]:
        return 0
    else:
        return nb_solutions(n-3) + nb_solutions(n-5) + nb_solutions(n-7)
```

6. Afin de diminuer le nombre d'appels récursifs, il est possible de mémoriser les résultats déjà calculés pour ne pas les refaire. C'est la mémoization.

7. `solutions_possibles(11)` appelle :
`solutions_possibles(8)` `[0, 5, 8], [0, 3, 8]` avec une pénalité supplémentaire
`solutions_possibles(6)` `[0, 3, 6]` avec un essai supplémentaire
`solutions_possibles(4)` `[]` avec un essai transformé supplémentaire
pour trouver les solutions : `[0, 5, 8, 11], [0, 3, 8, 11], [0, 3, 6, 11]`

8.

```
def solutions_possibles(score):
    if score < 0:
        resultat = []
    elif score == 0:
        resultat = [[0]]
    else:
        resultat = []
        for coup in [3, 5, 7]:
            liste = solutions_possibles(score - coup)
        for solution in liste:
            solution.append(score)
            resultat.append(solution)
    return resultat
```

Exercice 3 : Partie A :

1.
```
participants["PHILIPSEN Jasper"]
classement_general[participants["PHILIPSEN Jasper"]]
temps_etapes[participants["PINOT Thibaut"]][3]
```

2.
```
def calcul_temps_total(d):
    s = 0
    for t in temps_etapes[d]:
        s += t
    return s
```

Ou

```
def calcul_temps_total(d):
    return sum(temps_etapes[d])
```

3.
```
classement = []

for numero_dossard in temps_etapes:
    element = (numero_dossard, calcul_temps_total(numero_dossard))
    classement.append(element)
    pos = len(classement) - 2
    while pos >= 0 and element[1] < classement[pos][1]:
        classement[pos + 1] = classement[pos]
        pos = pos - 1
        classement[pos + 1] = element

for i in range(len(classement)):
    classement_general[classement[i][0]] = i + 1
```

Le boucle `while` permet d'insérer l'élément au bon endroit dans la liste `classement` en trouvant sa position `pos + 1` de telle sorte à ce que ce soit trié correctement.

4.
```
tableau_final = []
difference_temps = 0
premier = True
for ligne in tableau_temps:
    coureur = [ligne[0]]
    coureur.append(ligne[1])
    if premier:
        temps_premier = ligne[2]
        coureur.append(temps_premier)
        premier = False
    else:
        difference_temps = ligne[2] - temps_premier
        coureur.append(difference_temps)
    tableau_final.append(coureur)
```

Exercice 3 : Partie B :

5. Une clé primaire doit permettre d'identifier de manière unique chaque enregistrement dans une table. Le couple de valeur (numDossard, NumEtape) permet cela.

6. La requête renvoie les noms des coureurs de l'équipe Cofidis.

7.
```
SELECT Date FROM Etapes
WHERE Type = 'contre-la-montre' ORDER BY Date ;
```

8.
```
SELECT Equipes.directeurSportif FROM Equipes
JOIN Coureurs ON Equipes.nomEquipe = Coureurs.Equipe
WHERE nomCoureur = 'BARDET Romain' ;
```

9. La contrainte de référence provoque une erreur car la première requête ajoute un enregistrement faisant référence à une étape (5) qui n'existe pas encore dans la base de données.

10. La solution est d'inverser les deux requêtes pour créer l'étape avant d'y faire référence :
```
INSERT INTO Etapes VALUES(5, 'Montagne', 'Pau', 'Laruns', 05/07/2023);
INSERT INTO Temps VALUES (1, 5, 14267);
```

11. En considérant que la base de donnée ne contient que les données de l'édition 2023 du Tour de France :
```
SELECT SUM(Temps.tempsRéalisé) AS tempsTotal FROM Temps
JOIN Coureurs ON Temps.numDossard = Coureurs.numDossard
WHERE Coureurs.nomCoureur = 'BARDET Romain' ;
```