Proposition de correction

Exercice 1

Partie A

Q1

TCP : protocole de transport chargé de découper et d'assembler les paquets

IP : protocole de routage chargé de trouver la bonne route

Q2a

200.100.10.0

Q2b

200.100.10.1 à 200.100.10.254, soit 256 – 2 = 254 hôtes

Partie B

Q1

machine A : 172.16.0.0

machine F : 10.0.0.0

Q2

Pour le réseau 1, les machine possèdent toutes la même adresse réseau 172.16.0.0 et font partir du même réseau.

Par contre seules les machines F à I appartiennent au réseau 10.0.0.0. La machine J appartient au réseau 8.0.0.0.

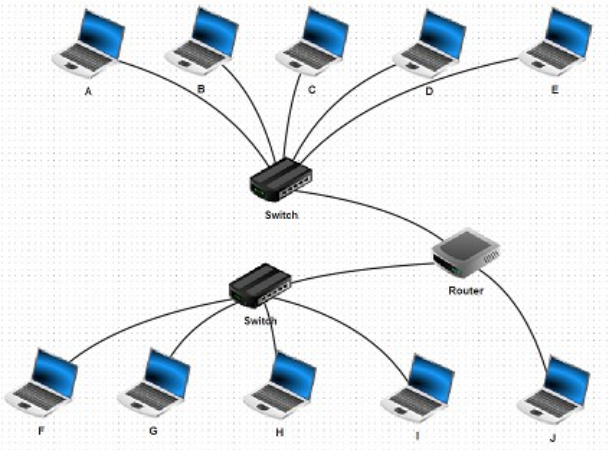
Q3

réseau 1 : $256^2 - 2$

Q4

il est nécessaire d'utiliser un routeur qui va servir de passerelles entre les réseaux.

Un concentrateur va également permettre de relier les machines d'un même réseau.



Exercice 2

Partie A

Q1

Vrai

Q2

logarithmique

Q3

- fin → $1/2^n$ taille(liste) – 1, donc fin → -1 quand n → +∞
- deb → $1/2^n$ taille(liste) + 1, donc deb → +1 quand n → +∞

la condition de continuation deb <= fin n'est plus réalisée

Partie B

Q1

liste est passée en paramètre, on ne connaît pas à priori la taille de liste

Q2

algorithme quotient(numérateur : entier, diviseur : entier) : entier

début

quotient : entier := 0

tant que (((quotient + 1) * diviseur) < numérateur) **faire**

quotient := quotient + 1

renvoyer quotient

fin

Q3

Variables			Condition	Valeur renvoyée
deb	**Fin**	**m**	**deb <= fin**	
0	6	3	true	
4	6	5	true	
4	4	4	false	True

Q4

```
def rechercheDicho(elem, liste):
    deb = 0
    fin = len(liste)-1
    m   = (deb + fin) // 2
```

```
    while deb <= fin :
        if liste[m] == elem :
            return True, m
        elif liste[m] > elem :
            fin = m-1
        else :
            deb = m+1

        m = (deb + fin) // 2

    return False, -1
```

Partie C

Q1

Programme qui s'appelle lui même

Q2

```
def rechercheDicho(elem : str, liste : list, deb : int, fin : int) -> bool:
    """ recherche dichotomique d'un élément dans une liste
    renvoie True si l'objet a été trouvé, False sinon."""
    if deb > fin :
        return False

    m = (deb + fin) // 2
    if liste[m] == elem :
        return True
    elif liste[m] > elem :
        return rechercheDicho(elem, liste, deb, m-1)
    else :
        return rechercheDicho(elem, liste, m+1, fin)
```

Exercice 3

Partie A

Q1

Table : Aeroport

Attribut : codeIATA

Q2

a) → 1

b) → 3

Partie B

Q1

contrainte d'intégrité référentielle : la clef étrangère BCN n'existe pas dans la table Aeroport

Q2

contrainte d'intégrité de clé : la clef F-KI452 existe déjà dans la table Avion

Q3

contrainte d'intégrité de domaine : le valeur « environ 200 » n'est pas de type entier

Partie C

Q1

supprime tous les vols de la table vol dont la date de départ est antérieure au 11 janvier 2021

Q2

INSERT INTO Type VALUES ("A310",250, "Airbus") ;

Q3

SELECT DISTINCTROW Type.nomT

FROM Type, Vol, Avion

WHERE Vol.dateVol = "10/01/2021"

AND Avion.numA = Vol.numAvion

AND Type.nomT = Avion.type

ORDER BY Type.nomT ASC

Exercice 4

Partie A

Q1

- _nom : str (nom de la chambre), accesseur get_nom()
- _occupation : list (tableau de 365 booléens), accesseur get_occupation(), mutateur reserver()

Q2

assert 0 < date < 366

Q3

```
def AnnulerReserver(self, date : int):
    assert 0 < date < 366
    self._occupation[date - 1] = False
```

Partie B

Q1

GiteBN.ajouter_chambres("Ch1")

Q2

```
def ajouter_chambres(self, nom_ch : str) -> bool:
    if nom_ch not in self.get_nchambres():
        self._chambres.append(Chambre(nom_ch))
        return True
    return False
```

Q3a

Tableau d'objet Chambre

Q3b

Ch2

Q3c

- get_chambres() : tableau d'objet Chambre
- get_nchambres() : tableau de nom de chambre

Q4a

la liste des chambres inoccupées en date du jour j

Q4b

def mystere(self, date : **int**) -> **list**:

- attribut : self._chambres
- méthode : get_occupation()

Exercice 5

Q1a

Un arbre binaire de recherche est un arbre binaire dans lequel chaque nœud possède une clé, telle que chaque nœud du sous-arbre gauche ait une clé inférieure ou égale à celle du nœud considéré, et que chaque nœud du sous-arbre droit possède une clé supérieure ou égale à celle-ci.

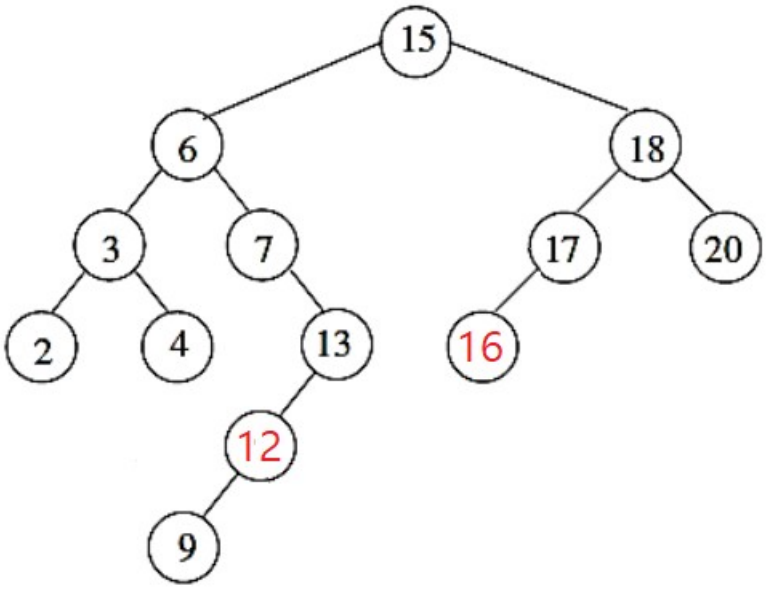
Q1b

15

Q1c

4

Q2



Q3

2 3 4 6 7 9 13 15 17 18 20

on obtient une liste triée

Q4

Recherche(A, x) :

Si EstVide(A) alors Faux
Si Racine(A) = x alors Vrai
Si x<Racine(A) alors Sag(A, x)
Sinon Sad(A, x)