Proposition de correction

Exercice 1

Q1.a

Oui : la fonction A() s'appelle elle même à la ligne 5

Q1.b

dépassement de la pile d'exécution au bout de 1000 appels

Q2.a

```
def A(n):
    if n <= 0 or choice([True, False]) :
            return "a"
    else:
            return "a" + A(n – 1)  + "a"
```

Q2.b

dans le cas où choice([True, False]) renvoi True : la fonction se termine

dans le cas où choice([True, False]) renvoi False: n est décrémenté à chaque appel et n → -∞. La condition de continuation n ≥ 0 n'est plus vérifiée et la fonction termine.

Q3

- B(0) : bab
- B(1) : bab, bbabb
- B(2) : bab, baaab, bbabb, bbbabbb

Q4.a

```
def regleA(chaine):
    n = len(chaine)
    if n >= 2:
            return chaine[0] == "a" and chaine[n-1] == "a" and regleA(raccourcir(chaine))
    else:
            return chaine == "a"
```

Q4.b

```
def regleB(chaine):
    n = len(chaine)
    if n > 2:
            sous_chaine = raccourcir(chaine)
```

```
        return chaine[0] == "b" and chaine[n-1] == "b" and \
                ( regleA(sous_chaine) or regleB(sous_chaine) )
else:
        return chaine == "a"
```

Exercice 2

Q1

Numéro du périphérique	Adresse	Opération	Réponse de l'ordonnanceur
0	10	écriture	OK
1	11	lecture	OK
2	10	lecture	ATT
3	10	écriture	ATT
0	12	lecture	OK
1	10	lecture	OK
2	10	lecture	OK
3	10	écriture	ATT

Q2

Le périphérique sera toujours en ATTente

Q3.a

Tour	ordre	Périph. 0 (W @ 10)	Périph. 1 (R @ 10)
1	0 ; 1 ; 2 ; 3	OK	ATT
2	1 ; 2 ; 3 ; 0	ATT	OK
3	2 ; 3 ; 0 ; 1	OK	ATT
4	3 ; 0 ; 1 ; 2	OK	ATT

Q3.b

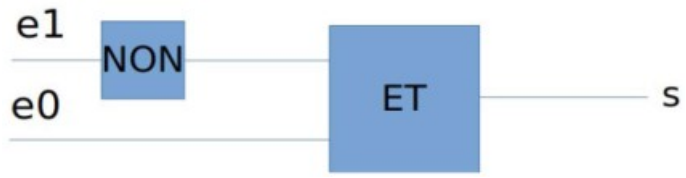
25 %

Q4

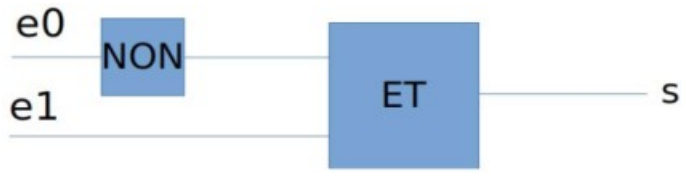
Tour	Numéro du périphérique	Adresse	Opération	Réponse de l'ordonnanceur	ATT_L	ATT_E
1	0	10	écriture	OK	vide	vide
1	1	10	lecture	ATT	(1,10)	vide

1	2	11	écriture	OK	(1,10)	vide
1	3	11	lecture	ATT	(1,10), (3,11)	vide
2	1	10	lecture	OK	(3,11)	vide
2	3	11	lecture	OK	vide	vide
2	0	10	écriture	ATT	vide	(0,10)
2	2	12	écriture	OK	vide	(0,10)
3	0	10	écriture	OK	vide	vide
3	1	10	lecture	ATT	(1,10)	vide
3	2	11	écriture	OK	(1,10)	vide
3	3	12	lecture	OK	(1,10)	vide

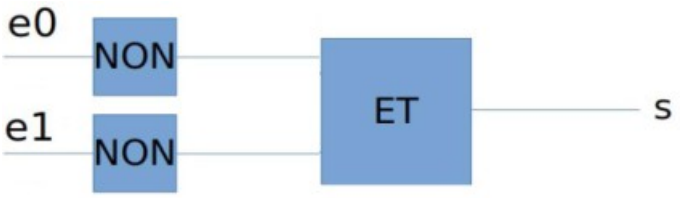
Q5.a



Q5.b



Q5.c



Exercice 3

Q1.a

SELECT identifiant, nompage

FROM Visites

ORDER BY identifiant

Q1.b

SELECT DISTINCTROW ip

```
FROM Visites
ORDER BY ip
```

Q1.c

```
SELECT DISTINCTROW nompage
FROM Visites
WHERE ip = '192.168.1.91'
ORDER BY nompage, ip
```

Q2.a

Visites

Q2.b

Pings

Q2.c

La clé étrangère identifiant de la table Pings doit exister en tant que clé primaire dans la table Visites.

Q3

```
INSERT INTO Pings
VALUES(1534, 105)
```

Q4.a

```
UPDATE Ping
SET duree = 120
WHERE identifiant = 1534
```

Q4.b

Le délai d'acheminement de l'information sur le réseau dépend de l'état du réseau.

Q4.c

Si une requête de mise à jour n-1 arrive après la requête n, le temps ne sera pas correct.

Q5.

```
SELECT DISTINCTROW Visites.nompage
FROM Visites JOIN Pings
ON Visites.identifiant = Pings.identifiant
WHERE Pings.duree > 60
ORDER BY Pings.duree DESC, Visites.nompage ASC
```

Exercice 4

Q1

```
def est_triee(self):
    if not self.est_vide() :
            e1 = self.depiler()
    while not self.est_vide():
            e2 = self.depiler()
            if e1 > e2 :
                    return False
            e1 = e2
    return True
```

Q2.a

False

Q2.b

[1, 2]

Q3

```
def depileMax(self):
    assert not self.est_vide(), "Pile vide"
    q = Pile()
    maxi = self.depiler()
    while not self.est_vide() :
            elt = self.depiler()
            if maxi < elt :
                    q.empiler(maxi)
                    maxi = elt
            else :
                    q.empiler(elt)
    while not q.est_vide():
            self.empiler(q.depiler())
    return maxi
```

Q4.a

1. B [9, -7, 8]

   q [4]

2. B [9, -7]

   q [4, 8]

3. B [9]

   q [4, 8, -7]

4. B []

q [4, 8, -7, 9]

Q4.b

B [9, -7, 8, 4]

q []

Q4.c

[3, 1, 2]

Q5.a

Ligne 3

B [1, 6, 4, 3, 7, 2]

q []

ligne 5

B []

q [7, 6, 4, 3, 2, 1]

fin fonction

B [1, 2, 3, 4, 6, 7]

q []

Q5.b

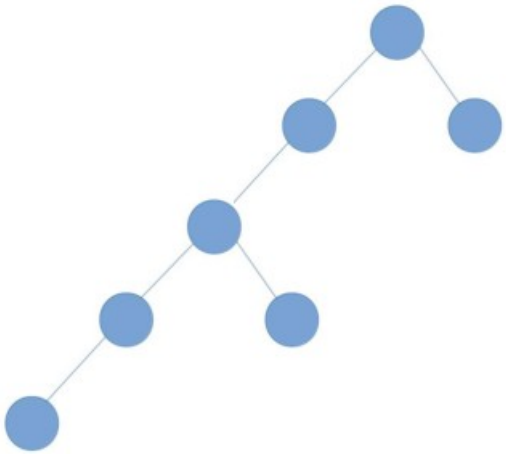
tri équivalent à un tri par insertion (recherche max dans un sous ensemble)

Exercice 5

Q1.a

2 (convention h(racine) = 0)

Q1.b



Q2

```
Algorithme hauteur(A)
    test d'assertion : A est supposé non vide
    si sous_arbre_gauche(A) vide et sous_arbre_droit(A) vide alors
            renvoyer 0
    sinon si sous_arbre_gauche(A) vide alors
            renvoyer 1 + hauteur(sous_arbre_droit(A))
    sinon si sous_arbre_droit(A) vide alors
            renvoyer 1 + hauteur(sous_arbre_gauche(A))
    sinon
            renvoyer 1 + max(hauteur(sous_arbre_gauche(A)), hauteur(sous_arbre_droit(A)))
```

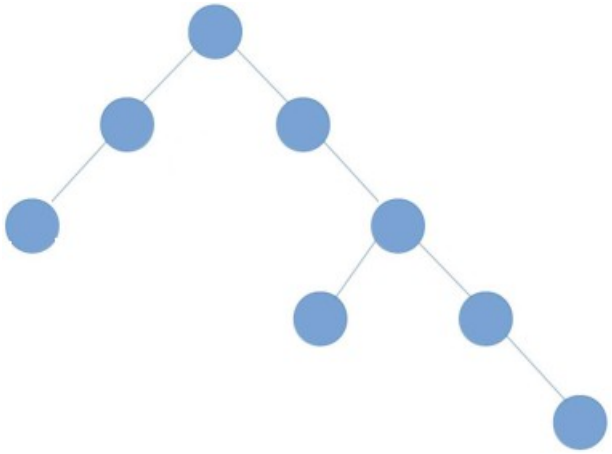
Q3.a

hauteur(R) = 4 = max(hauteur(G), hauteur(D)), avec hauteur(G) = 2

d'où : max(2, hauteur(D)) = 4

il vient : hauteur(D) = 4 ≠ 0

Q3.b



Q4.a

soit $h+1 \leqslant n \leqslant 2^{h+1}-1$

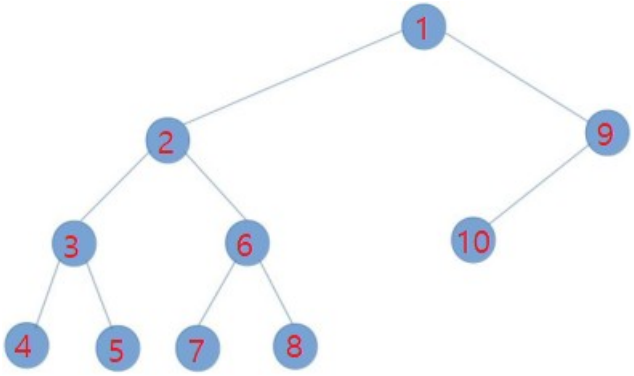
pour n = 4 et h = 2, on obtient $2+1 \leqslant 4 \leqslant 2^{2+1}-1$

il vient : $3 \leqslant 4 \leqslant 7$

Q4.c

en complétant au maximum chaque niveau de l'arbre

Q5.



Q6

```
def fabrique(h, n):
    def annexe(hauteur_max):
            if n == 0 :
                    return arbre_vide()
            elif hauteur_max == 0:
                    n = n - 1
                    return arbre(arbre_vide(), arbre_vide())
            else:
                    n = n - 1
                    gauche = annexe(hauteur_max - 1)
                    droite   = annexe(hauteur_max - 1)
                    return arbre(gauche, droite)
    return annexe(h)
```