Correction du bac Polynésie **2021/2022**
T^le NSI

Exercice 1 : récursivité (4 points)

1.
   a. Oui cette fonction est récursive car elle contient un appel à elle-même dans son code.
   b. Au bout d'un grand nombre de $False$ consécutifs, une erreur d'exécution sera déclenchée car la limite de taille de la pile des appels récursifs en mémoire sera atteinte.
2.
   a.
```
def A(n):
    """retourne une chaine de caractères"""
    if n<=0 or choice([True,False]):
        return "a"
    else :
        return "a"+A(n-1)+"a"
```
   b. Dans l'algorithme précédent, les valeurs successives de n forment une suite décroissante d'entiers minorée par 0 dans le pire des cas. Donc la fonction se termine toujours avec un $return$ "a".
3. $B(0)$ retourne "bab".
   $B(1)$ retourne "bab" ou "$bbabb$".
   $B(2)$ retourne "bab" ou "$baaab$" ou "$bbabb$" ou "$bbbabbb$".

4.
   a.
```
def regleA(chaine):
    """retourne un booléen"""
    n = len(chaine)
    if n>= 2 :
        return chaine[0]=="a" and chaine[n-1]=="a" and regleA(raccourcir(chaine))
    else :
        return chaine == "a"
```
   b.
```
def regleB(chaine):
    """retourne un booléen"""
    n = len(chaine)
    if n >= 2 :
        return chaine[0]=="b" and chaine[n-1]=="b" and (regleA(raccourcir(chaine)) or regleB(raccourcir(chaine)))
    else :
        return chaine == "b"
```

Exercice 2 : architecture (4 points)

1.

N° 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**

2. Le périphérique 1 ne pourra jamais lire l'adresse 10 car le périphérique 0 sera systématiquement en écriture sur la même adresse. Le périphérique 1 est bloqué.

3.

a.

Tour 1		Tour 2		Tour 3		Tour 4	
Périphérique 0	**OK**	Périphérique 0	**ATT**	Périphérique 0	**OK**	Périphérique 0	**OK**
Périphérique 1	**ATT**	Périphérique 1	**OK**	Périphérique 1	**ATT**	Périphérique 1	**ATT**

b. Le périphérique 0 lit une valeur sur trois du périphérique 1 (un tiers des valeurs).

4.

Tour	N° du périphérique	Adresse	Opération	Réponse de l'ordonnanceur	*ATT_L*	*ATT_E*
1	0	10	Ecriture	OK	Vide	Vide
1	1	10	Lecture	ATT	(1,10)	Vide
1	2	11	Ecriture	**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**	**Ecriture**	**ATT**	**Vide**	**(0,10)**
2	**2**	**12**	**Ecriture**	**OK**	**Vide**	**(0,10)**
3	0	10	Ecriture	**OK**	Vide	Vide
3	1	10	Lecture	**ATT**	**(1,10)**	Vide
3	2	11	Ecriture	OK	(1,10)	Vide
3	3	12	Lecture	**OK**	**(1,10)**	**Vide**

5.

a.

b.

c.

Exercice 3 : Base de données (4 points)

1.
   a.
```
SELECT ip, nompage
FROM Visites;
```
   b.
```
SELECT DISTINCT ip
FROM Visites;
```
   c.
```
SELECT nompage
FROM Visites
WHERE ip = "192.168.1.91";
```
2.
   a. L'attribut $identifiant$ est la clé primaire de la table $Visites$.
   b. L'attribut $identifiant$ est la clé étrangère de la table $Pings$.
   c. L'identifiant doit être unique (contrainte d'entité) et tous les identifiants de la table $Pings$ doivent correspondre à un identifiant de la table $Visites$ (contrainte de référence).
3.
```
INSERT INTO Pings
VALUES (1534,105);
```
4.
   a.
```
UPDATE Pings
SET duree = 120
WHERE idenfiant = 1534;
```
   b. Les requêtes arrivent au serveur dans un ordre quelconque car elles n'empruntent pas forcément les mêmes chemins (entre les routeurs).
   c. Il est préférable de faire une insertion plutôt qu'une mise à jour car une requête émise après une autre pourrait ne pas être mise à jour si elle arrive après la requête de mise à jour.
5.
```
SELECT DISTINCT nompage
FROM Vistes JOIN Pings ON Visites.identifiant = Pings.identifiant
WHERE duree>60;
```

Exercice 4 : les piles (4 points)

1.
```
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
```
2.

a. Une fois dépilée, les éléments seront $4-3-2-1$ donc non croissants donc l'instruction $A.est\_triee()$ retourne $False$.

b. Comme la pile n'est pas triée, la fonction s'arrête au premier $return$ et donc $A = [1,2]$.

3.
```
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
```
4.

a.

Itération n°	1	2	3	4
$B =$	$[9,-7,8]$	$[9,-7]$	$[9]$	$[\,]$
$q =$	$[4]$	$[4,8]$	$[4,8,-7]$	$[4,8,-7,9]$

b. $B = [9,-7,8,4]$ et $q = [\,]$.

c. Par exemple $B = [5,1,2,3]$ devient $B = [3,1,2]$

5.

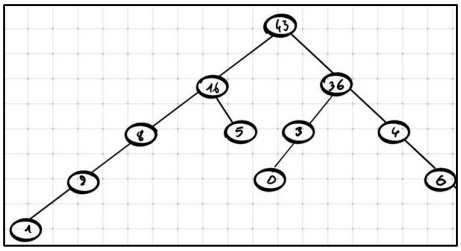
a. Avant la ligne 3 : $B = [1,6,4,3,7,2]$ et $q = [\,]$.
Avant la ligne 5 : $B = [\,]$ et $q = [\,]$.
A la fin de l'exécution de la fonction : $B = [1,2,3,4,6,7]$ et $q = [\,]$.

b. Cette méthode range dans l'ordre croissant la pile B. Une fois dépilés, les éléments de la pile seront classés du plus petit au plus grand.

Exercice 5 : les arbres binaires (4 points)

1.

a. La hauteur est 2.

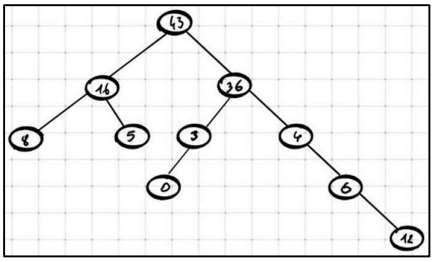
b. 

2.

```python
def hauteur(A):
    """retourne la hauteur de l'arbre A"""
    assert not est_vide(A),"Arbre non vide"
    if est_vide(A.sous_arbre_gauche) and est_vide(A.sous_arbre_droit):
        return 0
    elif not est_vide(A.sous_arbre_droit):
        return 1 + hauteur(A.sous_arbre_droit)
    elif not est_vide(A.sous_arbre_gauche):
        return 1 + hauteur(A.sous_arbre_gauche)
    else :
        return 1 + max( hauteur(A.sous_arbre_gauche),hauteur(A.sous_arbre_droit))
```

3.

a. Comme $hauteur(R) = 1 + max(hauteur(G), hauteur(D))$, alors $4 = 1 + max(2,\ hauteur_D)$ et donc $hauteur_D = 3$ donc D n'est pas un arbre vide.

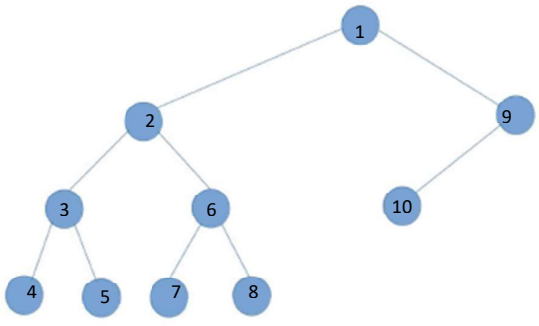
b. 

4.

a. On a $h = 2$ et $n = 4$ donc $2 + 1 \leq 4 \leq 2^{2+1} - 1$ et donc l'inégalité $3 \leq 4 \leq 7$ est vraie ici.

b. Il suffit de créer un arbre filiforme : pour construire l'arbre on ajoute les valeurs dans le dernier fils gauche uniquement par exemple.

c. On remplit l'arbre au maximum étage par étage pour que toutes les feuilles aient la même hauteur. Il sera donc parfait.

5. 

6.

```python
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)
```