Consignes générales :

- Pas de respect de l'indentation pdt toute l'épreuve : -0.5 pts à retrnacher de la partie problème
- Si le candidat laisse la fonction vide, et ne fait que des initialisations et des return , à compter seulement dans une seule question, pour les autres questions c'est 0.
- La question Q3 du problème la fonction XOR on la compte correcte dans les deux cas

Exercice (4 pts)

```
## Q1 ( 0.5 pts)

def divStrict(x):
    for i in range(1,x//2+1): # 0.25 boucle for
        if x%i==0:            # 0.25 corps de la boucle
            print(i)
```

Q2 (0.5 pts)

0.25 pour la réponse

0.25 pour la justification

En considérant que la complexité de l'opérateur % est O(1), la complexité de divStrict(x) est $\Sigma_{i=1}^{x/2} O(1) = O(x)$

```
## Q3 (1 pt)

def somDivStrict(x):
    s=0   # 0.25 initialisation
    for i in range(1,x//2+1):  # 0.25 boucle for
        if x%i==0:             # 0.25 corps de la boucle
            s=s+i
    return s     # 0.25 pour le retour

## Q4 (0.5 pt)

def amis(x,y):
    return x==somDivStrict(x) and y==somDivStrict(x)
```

```python
## Q5 (1 pt)

def liste_amis(n):
    L=[] # 0.25 initialisation
    for i in range(1,n+1):     # 0.5 boucle et corps
        for j in range(i,n+1): # de la boucle
            if amis(i,j):
                L.append((i,j))
    return L # 0.25 pour le retour
```

Q6

0.25 pour la réponse

0.25 pour la justification

On considére que la complexité de l'opérateur % et de la fonction append sont O(1).

Tout d'abord, la complexité de somDivStrict(x) est $\Sigma_{i=1}^{x/2} O(1) = O(x)$

Alors la complexité de amis(x,y) est O(x+y)

d'où la complexité de liste_amis(n) est $O(1) + \Sigma_{i=1}^{n} \Sigma_{j=i}^{n} O(i+j) = O(1 + \Sigma_{i=1}^{n}(\frac{(n+i)(n-i+1)}{2} + (n-i+1)i)) = n^3$

Partie I : BD & SQL (20 pts)

Q1) (1.5 pts)

0.5 pour insert

0.5 pour chaque tuple : respect de l'ordre et du typage

INSERT INTO Personnes VALUES (123,"Guissi","Maryam","F")

INSERT INTO Personnes VALUES (138,"Fora","Ahmed","M")

Q2) (2 pts)

0.5 pour projection

0.5 pour selection

1 pour jointure

$\Pi_{code,nom,prénom}(\sigma_{codC=394}(Votes \bowtie_{codP=code} Personnes))$

Q3) (2.5 pts)

0.5 pour select

0.5 pour where

1 pour jointure

0.5 pour order by

SELECT code, nom ,prénom

FROM Personnes

JOIN Votes

ON codP=code

WHERE codC=394

ORDER BY codP

Q4) (2.5 pts)

0.5 pour select

0.5 pour from

1 pour where

0.5 pour order by

SELECT code, nom ,prénom, genre

FROM Personnes

WHERE nom LIKE "_ A _ _ _%"

ORDER BY nom,prenom

Q5) (3 pts)

0.5 pour select

0.5 pour from

0.5 pour la 1ere condition du where

1.5 pour la 2eme condition du where qui contient une requete imbriquee

SELECT code, nom ,prénom

FROM Personnes

WHERE genre="M"

AND code NOT IN (SELECT codC FROM Votes)

Q6 (4.5 pts)

0.5 pour select

1 pour jointure

1 pour group by

1 pour having

1 pour la requete imbriquee

SELECT code, nom ,prénom,genre

FROM Personnes

JOIN Votes

ON codC=code

GROUP BY codC

HAVING 1.0 * count(codP)/(select count(*) from Votes)>0.5

Q7 (4 pts)

0.5 x 3 pour select

2.5 pour la requete imbriquee

SELECT max(c), min(c), avg(c)

FROM (SELECT count(*) as c FROM votes GROUP BY codC)

Partie II : calcul numérique (20 pts)

```
import numpy as np
import matplotlib.pyplot as plt
```

```
# Q1 (3 pts)

def decoupe(P):
    # initialisation 1 pt
    n=len(P)
    X=np.zeros(n)
    Y=np.zeros(n)

    for i in range(n): ## 1.5 pts for + contenu
        X[i]=P[i][0]
        Y[i]=P[i][1]
    return X,Y   ## 0.5 pts

P=[ (-1, 2) , (0, 1) , (5, 4) , (1, 3) , (-2, 8) , (-7, 3) , (3, -3) ]
X,Y=decoupe(P )
X
```

```
array([-1.,  0.,  5.,  1., -2., -7.,  3.])
```

```
# Q2 ( 4 pts)

def matrice(X):
    # initialisation 0.5 pt
    n=len(X)
    M=np.zeros((n,n))
    for i in range(n):        ## 3 pts pour remplir
        for j in range(i+1):  ## la matrice
            p=1
            for k in range(j):
                p=p*(X[i]-X[k])
            M[i][j]=p
    return M ## 0.5

## pénalité -0.5 pour les débordement des indices d'intervalles
## pénalité -0.5 pénalité mauvaise initialisation
```

```python
## Q3 (4 pts)

def solution(M,Y):
    n=len(M)
    A=np.zeros(n) ## 0.5 pts
    for i in range(n):  ## 1 pts
        ## 1.5 pour le calcul de la somme
        s=0
        for j in range(0,i):
            s=s+(M[i][j]*A[j])
        A[i]=(Y[i]-s)/M[i][i] ## 0.5
    return A # 0.5


## Q4 (4 pts)

def valeur(x,A,X):
    s=0 ## 0.5
    n=len(A)

    for i in range(n): ## 0.5

        ## 2 pts calcul du produit
        p=1
        for j in range(i):
            p=p*(x-X[j])

        s=s+A[i]*p ## 0.5

    return s #0.5

## Q5 (5 pts)
def courbe(P):
    X,Y=decoupe(P) # 0.5

    M=matrice(X) # 0.5
    A=solution(M,Y) # 0.5
    a,b=min(X),max(X) # 1

    x=np.linspace(a,b,500) # 0.5
    y=[valeur(i,A,X) for i in x] #1 pts

    plt.plot(x,y,'k') #0.5
    plt.scatter(X,Y)  #0.25
    plt.grid()      #0.25
    plt.show()
```

Partie III : Problème (60 pts)

```python
## Q1 (2.5 pts)

def NON(X):
    s=""  # 0.5
    for b in X: # 0.5
        if b=='1': # 1 pts pour if-else
            s=s+'0'
        else:
            s=s+'1'
    return s # 0.5


## Q2 (3 pts)

def ET(X,Y):
    s="" # 0.5
    for i in range(32): # 0.5
        if X[i]=='1' and Y[i]=='1': # 1.5 pts pour if-else
            s=s+'1'
        else:
            s=s+'0'
    return s # 0.5


## Q3 (3pts)

def XOR(X,Y):
    s="" # 0.5
    for i in range(32): # 0.5
        if X[i]==Y[i]: # 1.5 pour if-else
            s=s+'0'
        else:
            s=s+'1'
    return s # 0.5


## Q4 (2.5 pts)

def decal_droite(X,p):

    return p*'0'+X[:-p]
```

```python
## Q5 (2.5 pts)

def rotation_droite (X, p) :
    return X[-p:]+X[:-p]




## Q6 (6 pts)

def addition(X,Y):
    ## 0.5 initialisation
    s=""
    r="0"
    for i in range(31,-1,-1): # 1 pts boucle for

        a=int(X[i])+int(Y[i])+int(r) # 1 pts

        # 3 pts pour le calcul du S et r
        s=str(a%2)+s
        r=str(a//2)

    return s # 0.5


## Q7 (5.5 pts )

def bin_fract(x,n):
    rac_n=x**(1/n) ## 0.75
    y=rac_n-int(rac_n)  ## 0.75
    s="" ## 0.5
    for i in range(32): # 3 pts boucle + contenu
        y=y*2
        if y>=1:
            s=s+'1'
            y=y-1
        else:
            s=s+'0'
    return s # 0.5
```

```python
## q8 (3 pts)

def liste_init(t,n):
    global L
    R=[] # 0.5 pts initialisation
    for i in range(t): # 1 pts
        R.append(bin_fract(L[i],n)) # 1 pts
    return R # 0.5




## Q9 ( 2 pts )

# 1 pts
R=liste_init(8,2)
h0=R[0]
h1=R[1]
h2=R[2]
h3=R[3]
h4=R[4]
h5=R[5]
h6=R[6]
h7=R[7]

# 1 pts
K=liste_init(64,3)


## q10 (3.5 pts)

def binaire(x,t):
    s='' # 0.5 pts

    #2 pts pour la conversion binaire
    while x!=0:
        s=str(x%2)+s
        x=x//2
     # 0.5 pts pour compléter par des 0
    return '0'*(t-len(s))+s # 0.5 pts

# remarque : si le candidat utilise la fonction predefinie
# bin , on sera strict lors dans son utilisation, sinon
# si le candidat ne place pas les bits à chaque fois
# à gauche , on pénalise par 0.5
```

```python
## Q11 (3 pts)


def binaire_txt(T):
    s="" #0.5 pts
    for c in T: # 1 pts
        s=s+binaire(c,8) # 1 pts
    return s # 0.5 pts


## Q12 (4.5 pts)

def remplissage(B):
    n=len(B) # 0.5 pts
    B=B+'1' # 0.5 pts
    while len(B)%512!=448: # 2 pts
        B=B+'0'
    B=B+binaire(n,64) # 1 pts
    return B # 0.5


## Q13 (5 pts)

def decoupe(B):
    p=len(B)//512

    W=[] # 0.5 pts
    for i in range(p): # 1 pts for + 3 pts contenu
        sschaine=B[512*i:512*(i+1)]
        w=[]
        for j in range(16):
            w.append(sschaine[32*j:32*(j+1)])
        W.append(w)
    return W #0.5
```

```python
## Q14 (7 pts)

def compression(W):
    global h0,h1,h2,h3,h4,h5,h6,h7 #0.5 pts

    for w in W: # 0.5 pts
        for i in range(48): # 1 pts
            w.append('0'*32)

        for i in range(16,64): # 1.5 pts
            A=XOR(rotation_droite(w[i-15],7),rotation_droite(w[i-15],18))
            S0=XOR(A,rotation_droite(w[i-15],3))
            B=XOR(rotation_droite(w[i-2],17),rotation_droite(w[i-2],19))
            S1=XOR(A,rotation_droite(w[i-12],10))
            w[i]=w[i-16]+S0+w[i-7]+S1

        a,b,c,d,e,f,g,h=h0,h1,h2,h3,h4,h5,h6,h7 #0.5

        for i in range(64): # 2 pts contenu du for
            S0=XOR(XOR(rotation_droite(e,6),rotation_droite(e,11)),rotation_droite(e,25))
            ch=XOR(ET(e,f),ET(NON(e),g))
            tmp1=addition(addition(addition(h,s0),addition(ch,K[i])),w[i])
            S1=XOR(XOR(rotation_droite(a,2),rotation_droite(a,13)),rotation_droite(a,22))
            maj=XOR(XOR(ET(a,b),ET(a,c)),ET(b,c))
            tmp2=addition(S1,maj)
            h=g
            g=f
            f=e
            e=addition(d,tmp1)
            d=c
            c=b
            b=a
            a=addition(tmp1,tmp2)

    #1 pts pour modification des hi
    h0=addition(a,h0)
    h1=addition(b,h1)
    h2=addition(c,h2)
    h3=addition(d,h3)
    h4=addition(e,h4)
    h5=addition(f,h5)
    h6=addition(g,h6)
    h7=addition(h,h7)
```

```python
## Q15 (7 pts)


def hache_final():
    global h0,h1,h2,h3,h4,h5,h6,h7

    H= h0+h1+h2+h3+h4+h5+h6+h7 # 0.5 pts

    dico={"0000":"0",...,"1111":"F"} # 4 pts conversion du binaire au hexa

    # 2 pts pour le haché final
    s=""
    for i in range(0,len(H),4):
        s=s+dico[H[i:i+4]]

    return s #0.5
```