```
## Préliminaires
def str_to_list(chaine):
    """ Convertit une chaine de caractère entre liste. Les
  caractères hors de la plage [0;255] sont ignorés """
    res = []
    for c in chaine:
        if ord(c) < 256:
            res.append(ord(c))
    return res

def list_to_str(liste):
    """ Convertit une liste d'entiers entre 0 et 255 en chaine """
    return "".join(chr(i) for i in liste)



## Question 1
def occurrences(l):
    r = [0] * 256
    for i in l:
        r[i] += 1
    return r

## Question 2
def min(l):
    r = occurrences(l)
    m = 0
    # m est l'indice du nombre minimum dans r.
    for i in range(256):
        if r[i] < r[m]:
            m = i
    return m

## Question 3
def taille_codage(l):
    # res est la longueur totale.
    # On devra mettre le marqueur, donc res commence à 1.
    res = 1
    c = l[0]
    long = 1
    # c est le caractère courant (pour détecter les répétitions)
    # long est la longueur de la répétition
    for i in range(1,len(l)):
        if l[i] != c:
            if long == 1: # Le caractère précédent ne se répète pas
                res += 1
            if long > 1: # Il y avait une répétition
                res += 3
            c = l[i]
            long = 1
        else:
            long += 1
    # Une fois la boucle terminée, il reste la fin à vérifier.
    if long == 1: # Le caractère précédent ne se répète pas
        res += 1
    if long > 1: # Il y avait une répétition
        res += 3
    return res

## Question 4
def codage(l):
    marqueur = min(l)
    res = [marqueur]
    # res est la liste à renvoyer.
    c = l[0]
    long = 1
    # même principe qu'au-dessus
    for i in range(1,len(l)):
        if l[i] != c:
```



```
            if long == 1: # Le caractère précédent ne se répète pas
                res.append(c)
            if long > 1: # Il y avait une répétition
                res.append(marqueur)
                res.append(long-1)
                res.append(c)
            c = l[i]
            long = 1
        else:
            long += 1
    # Une fois la boucle terminée, il reste la fin à vérifier.
    if long == 1: # Le caractère précédent ne se répète pas
        res.append(c)
    if long > 1: # Il y avait une répétition
        res.append(marqueur)
        res.append(long-1)
        res.append(c)
    return res

## Question 5
def decodage(lprime):
    i = 1
    marqueur = lprime[0]
    res = []
    # i est l'indice de lecture dans la liste.
    # res est la liste l qu'on reconstruit.
    while i < len(lprime):
        # On parcourt lprime.
        if lprime[i] != marqueur:
            # On a un caractère simple
            res.append(lprime[i])
            i += 1
        else:
            # On a une répétition
            # On récupère la longueur et le caractère
            long = lprime[i+1] + 1
            c = lprime[i+2]
            # On les rajoute à la suite
            for k in range(long):
                res.append(c)
            i += 3
    return res

## Question 6
def comparer_rotations(l,i,j):
    n = len(l)
    # k est la lettre à comparer dans rot[i] et rot[j]
    for k in range(n):
        # dans rot[i], on parle alors de la lettre l[(k+i) mod n]
        # dans rot[j], il s'agit de l[(k+j) mod n]
        if l[(k+i)%n] > l[(k+j)%n]:
            return 1
        if l[(k+i)%n] < l[(k+j)%n]:
            return -1
    return 0

## Question pas demandée
def rotation(l,d):
    # On n'utilisera pas cette fonction (lourde)
    # autrement que pour tester
    n = len(l)
    res = []
    for i in range(n):
        res.append(l[(i+d) % n])
    return res


def tri_rotations(l):
    # Ici, pour ne pas m'embêter, je fais un tri par insertion.
```

```
    # Cela ne convient pas pour la complexité, mais tant pis.
    n = len(l)
    r = [i for i in range(n)]
    for i in range(1,n):
        while i>0 and comparer_rotations(l,r[i],r[i-1]) == -1:
            r[i],r[i-1] = r[i-1],r[i]
            i -= 1
    return r

## Pour vérifier que tout marche bien
c = "concours"
l = str_to_list(c)
print([list_to_str(rotation(l,i)) for i in tri_rotations(l)])

## Question 7
def codage_bw(l):
    n = len(l)
    r = tri_rotations(l)
    cle = -1
    res = []
    # n est la taille de l. r le tableau des rotations trié.
    # cle est la cle finale
    # res est la liste codée
    for i in range(len(r)):
        # On regarde la rotation r[i]
        # Son dernier caractère est donc l[(r[i]-1) mod n]
        res.append(l[(r[i]-1) % n])
        if r[i] == 1:
            cle = i
    return res,cle

## Testons les choses
l = str_to_list("concours")
print(list_to_str(codage_bw(l)[0]))
l = str_to_list("concours de l ecole polytechnique")
print(list_to_str(codage_bw(l)[0]))

## Question 8
"""
La fonction tri_rotations fait au maximum appel à n ln(n) fois la fonction
comparer_rotations. Celle−ci s'execute dans le pire des cas en O(n) opérations.
D'où une complexité O(n² ln(n)) pour tri_rotations.
La suite des opérations dans codage_bw est en O(n) (une boucle for dans laquelle on
effectue un nombre borné d'opérations).
Au total, la complexité est en O(n² ln(n)).
"""

## Question 9
frequences = occurrences

## Question 10
def tri_cars_de(lprime):
    r = occurrences(l)
    res = []
    for i in range(256):
        for k in range(r[i]):
            res.append(i)
    return res

## Question 11
def trouver_indices(lprime):
    n = len(lprime)
    tri_cars = tri_cars_de(lprime)
    index = 0
    # index est la position à partir de laquelle on recherche une lettre dans lprime
.
    indices = []
    # indices est le tableau solution
    c = ""
```



```
    # c est la lettre recherchée dans lprime
    for i in range(n):
        if tri_cars[i] != c:
            # Si on cherche une nouvelle lettre, on repart du début du mot.
            index = 0
        c = tri_cars[i]
        # On cherche la lettre
        while lprime[index] != c:
            index += 1
        # Quand on sort de la boucle, index pointe la bonne lettre.
        indices.append(index)
        index += 1 # On l'incrémente pour ne pas retomber sur la même lettre.
    return indices

##
lprime = codage_bw(l)[0]
print(trouver_indices(lprime))

##
def decodage_bw(lprime,cle):
    n = len(lprime)
    indices = trouver_indices(lprime)
    res = []
    courant = cle
    # courant désigne le caractère courant.
    for i in range(n):
        res.append(lprime[courant])
        courant = indices[courant]
    return res

## Vérification...
l = str_to_list("concours")
print("concours",l)
lprime,cle = codage_bw(l)
print(lprime,cle)
print(list_to_str(decodage_bw(lprime,cle)))
```