TP 1 MANIPULATION DE LISTES

```python
## Exercice 1

# Question 1

def H(n) :
    q = n
    while q%5 == 0:
        q = q // 5
    while q%3 == 0:
        q = q // 3
    while q%2 == 0:
        q = q // 2
    return q == 1


# Question 2

def LH1(N) :
    n = 1
    nb = 0
    L = []
    while nb < N :
        if H(n) :
            nb += 1
            L.append(n)
        n += 1
    return L


# Question 3

def LH2(N) :
    t = [1]
    for k in range(N-1):
        m = t[-1]
        new2, new3, new5 = False, False, False
        i = 0
        while not (new2 and new3 and new5):
            if not new2 and 2*t[i] > m:
                new2 = True
                n2 = 2*t[i]
            if not new3 and 3*t[i] > m:
                new3 = True
                n3 = 3*t[i]
            if not new5 and 5*t[i] > m:
                new5 = True
                n5 = 5*t[i]
            i += 1
        t.append(min(n2, n3, n5))
    return t
```

```
# Question 4

def g(L2, L3, L5):
    n = min(L2[0], L3[0], L5[0])
    if n == L2[0]:
        L2.pop(0)
    if n == L3[0]:
        L3.pop(0)
    if n == L5[0]:
        L5.pop(0)
    L2.append(2*n)
    L3.append(3*n)
    L5.append(5*n)
    return n

L2, L3, L5 = [1], [1], [1]
LH = []
for no in range(15) :
    LH.append(g(L2, L3, L5))


# Question 5

from time import time

debut = time()
h = 1
for k in range(1, 10**6):
    if H(k):
        h = k
duree = time() - debut
print("méthode 1: ",duree)


debut = time()
t = [1]
while t[-1] < 10**6:
    m = t[-1]
    new2, new3, new5 = False, False, False
    i = 0
    while not (new2 and new3 and new5):
        if not new2 and 2*t[i] > m:
            new2 = True
            n2 = 2*t[i]
        if not new3 and 3*t[i] > m:
            new3 = True
            n3 = 3*t[i]
        if not new5 and 5*t[i] > m:
            new5 = True
            n5 = 5*t[i]
        i += 1
    t.append(min(n2, n3, n5))
duree = time() - debut
print("méthode 2: ",duree)


debut = time()
L2, L3, L5 = [1], [1], [1]
LH = []
Nmax = 10**6
LH.append(g(L2,L3,L5))
while LH[-1] < Nmax :
    LH.append(g(L2,L3,L5))
duree = time()-debut
print("méthode 3: ",duree)
```

```
## Exercice 2

# Question 1

def cardinal(A):
    c = 0
    for el in A:
        c += el
    return(c)


# Question 2

def intersection(A, B):
    inters = []
    for k in range(len(A)):
        if A[k]==1 and B[k]==1:
            inters.append(1)
        else:
            inters.append(0)
    return(inters)


# Question 3

def difference(A, B):
    dif = []
    for k in range(len(A)):
        if A[k] == 1 and B[k] == 0:
            dif.append(1)
        else:
            dif.append(0)
    return dif


# Question 4.a

def decoder(E,C):
    P = []
    for k in range(len(E)):
        if C[k] == 1:
            P.append(E[k])
    return P


# Question 4.b

def coder(E, P):
    C = [0 for k in range(len(E))]
    for e in P :
        C[E.index(e)] = 1
    return C
''' ou alors on utilise in, ou une recherche linéaire'''


# Question 4.c

def incrementer(C):
    C = C.copy() # pour ne pas modifier l'original
    no = len(C)-1
    while C[no] == 1 :
        C[no] = 0
        no -= 1
    if no == -1:
        print("Avertissement : dépassement")
    else:
        C[no] = 1
    return C
```

```
# Question 4.d

def parties(E):
    C = [0]*len(E)
    LP = [[]]
    for no in range(1,2**(len(E))) :
        C = incrementer(C)
        LP.append(decoder(E,C))
    return LP


# Question 4.e
def p_parties1(E,p):
    if p == 0:
        return []
    n = len(E)
    C = n*[0]
    LP = []
    for no in range(1,2**n) :
        C = incrementer(C)
        if cardinal(C)==p:
            LP.append(decoder(E,C))
    return LP

for i in range(5) :
    print(p_parties1([2,3,5,7],i))


# Question 5.a

def reunion(C1, C2):
    C = C1.copy()
    for k in range(len(C)):
        if C2[k] == 1:
            C[k] = 1
    return C


# Question 5.b

def estRec(R) :
    codeU = R[0]
    for C in R[1:] :
        codeU = reunion(codeU, C)
    return codeU == len(codeU)*[1]


# Question 5.c
''' Il suffit de vérifier que si l'on enlève une partie,
    alors R n'est plus un recouvrement.'''

def estRecMin(R) :
    if not estRec(R) :
        print("R n'est pas un recouvrement ! il ne peux donc être minimal.")
        return False
    for i in range(len(R)) :
        Rprime = R[:i] + R[i+1:]
        if estRec(Rprime):
            return False
    return True
```

```
# Question 5.d
''' Il suffit de vérifier, lorsque R est un recouvrement,
    que la somme des cardinaux des éléments de R vaut n
    et que les éléments de R sont non vides.'''

def estPartition(R) :
    if not estRec(R) :
        print("R n'est pas un recouvrement.")
        return False
    cardinaux = [sum(A) for A in R]
    if 0 in cardinaux :
        print("Il y a une partie vide")
        return False
    return sum(cardinaux) == len(R[0])


# Question 5.e
def partition(R) :
    n = len(R[0])
    reste = n * [1]
    P = []
    for e in R :
        p = []
        nonvide = False
        for b0,b1,i in zip(e,reste,range(n)) :
            if b0 == 1 and b1 == 1 :
                p.append(1)
                reste[i] = 0
                nonvide = True
            else :
                p.append(0)
        if nonvide : P.append(p)
    if sum(reste) != 0 : P.append(reste)
    return P
```