CORRIGÉ ÉPREUVE D'INFO X PSI 2016

```python
# PARTIE I

# Question 1
def deplacerParticule(particule, largeur, hauteur):
    x, y, vx, vy = particule
    if x + vx <= 0 or x + vx >= largeur:
        vx = -vx
    if y + vy <=0 or y + vy >= hauteur:
        vy = -vy
    return x + vx, y + vy, vx, vy

# Partie II

# Question 2

# Version utilisant la définition d'une liste en compréhension
def nouvelleGrille(largeur, hauteur):
    return[[None for j in range(hauteur)] for i in range(largeur)]

# Version plus basique
def nouvelleGrille2(largeur, hauteur):
    grille = []
    for i in range(largeur):
        ligne = []
        for j in range(hauteur):
            ligne.append(None)
        grille.append(ligne)
    return grille


# Question 3
def majGrilleOuCollision(grille):
    largeur, hauteur = len(grille), len(grille[0])
    nouvelle_grille = nouvelleGrille(largeur, hauteur)
    for i in range(largeur):
        for j in range(hauteur):
            if grille[i][j] is not None:
                nouvelleParticule = deplacerParticule(grille[i][j], largeur, hauteur)
                x, y = int(nouvelleParticule[0]), int(nouvelleParticule[1])
                if nouvelle_grille[x][y] is not None:
                    return None
                nouvelle_grille[x][y] = nouvelleParticule
    return nouv_grille


# Question 4
def attendreCollisionGrille(grille, tMax):
    t = 0
    while t < tMax and grille is not None:
        grille = majGrilleOuCollision(grille)
        t += 1
    if grille is None:
        return t
    return None
```

```
# Question 5
"""
- Complexité de deplacerParticule : O(1) ;
- complexité de majGrilleOuCollision : O(largeur*hauteur) ;
- complexité de attendreCollisionGrille : O(largeur*hauteur*tMax).
"""


# PARTIE III

# Question 6
def detecterCollisionEntreParticules(p1, p2):
    distance = (p1[0] - p2[0])**2 + (p1[1] - p2[1])**2
    return distance <= 4*rayon**2


# Question 7
def maj(particules):
    largeur, hauteur, listeParticules = particules
    nouvelleListe = []
    for p in listeParticules:
        nouvelleListe.append(deplacerParticule(p, largeur, hauteur))
    return largeur, hauteur, nouvelleListe


# Question 8
def majOuCollision(particules):
    nouvelleParticules = maj(particules)
    listeParticules = nouvelleParticules[2]
    n = len(listeParticules)
    for i in range(n-1):
        for j in range(i+1, n):
            if detecterCollisionEntreParticules(listeParticules[i], listeParticules[j]):
                return None
    return nouvelleParticules


# Question 9
def attendreCollision(particules, tMax):
    t = 0
    while t < tMax and particules is not None:
        particules = majOuCollision(particules)
        t += 1
    if particules is None:
        return t
    return None

"""
- complexité de detecterCollisionEntreParticules : O(1) ;
- complexité de maj : O(n) ;
- complexité de majOuCollision : O(n) + O(n**2) = O(n**2)
   ( car n(n-1)/2 boucles);
- complexité de attendreCollision : O(tMax * n**2).
"""

# Question 10
"""
Entre les instants t et t+1, une particule se déplace au plus de vMax, donc
deux particules se rapprochent au maximum de 2*vMax.
Pour que ces deux particules puissent alors entrer en collision, il faut qu'elles
se trouvent alors à une distance inférieure ou égale à 2*rayon, donc il faut que
```

```
    leur distance à l'instant t soit inférieure à 2*(vMax + rayon).
    """

    # Question 11
    def majOuCollisionX(particules):
        nouvelleParticules = maj(particules)
        listeParticules = nouvelleParticules[2]
        n = len(listeParticules)
        distanceMax = 2 * (rayon + vMax)
        for i in range(n - 1):
            xi = listeParticules[i][0] # abcisse de la particule i
            j = i + 1
            while j < n and listeParticules[j][0] - xi <=  distanceMax:
                if detecterCollisionEntreParticules(listeParticules[i], listeParticules[j]):
                    return None
                j += 1
        return nouvelleParticules


    # PARTIE IV

    # Question 12
    def scm(s):
        liste_scm = []
        n = len(s)
        d = 0
        for f in range(n-1):
            if s[f+1] < s[f]:
                liste_scm.append( (d, f) )
                d = f + 1
        liste_scm.append( (d, n-1) )
        return liste_scm

    # Question 13

    def fusionner(s, r1, r2):
        d1, f1 = r1
        d2, f2 = r2
        L = []
        i1, i2 = d1, d2
        while i1 <= f1 and i2 <= f2:
            if s[i1] <= s[i2]:
                L.append(s[i1])
                i1 += 1
            else:
                L.append(s[i2])
                i2 += 1
        while i1 <= f1:
            L.append(s[i1])
            i1 += 1
        while i2 <= f2:
            L.append(s[i2])
            i2 += 1
        # pour ces deux derniers cas, on peut faire plus rapide en utilisant directement
        # L.extend, mais il semble que cette méthode est interdite
        for i in range(len(L)):
            s[d1 + i] = L[i]
        # là encore on peut faire plus rapide en écrivant s[d1:f2+1]=L
```

```python
# Question 14
def depileFusionneRemplace(s, pile):
    r2 = (d2, f2) = pile.pop()
    r1 = (d1, f1) = pile.pop()
    fusionner(s, r1, r2)
    pile.append( (d1, f2) )


# Question 15
def alphaTri(s):
    def longueur(segment):
        return segment[1] - segment[0] + 1
    liste_scm = scm(s)
    nb_scm = len(liste_scm)
    # première phase
    pile = [ liste_scm[0] ]
    for i in range(1, nb_scm):
        pile.append( liste_scm[i] )
        while len(pile) > 1 and longueur(pile[-2]) < 2*longueur(pile[-1]) # attention à l'ordre des
            depileFusionneRemplace(s, pile)
    # deuxième phase
    while len(pile) > 1:
        depileFusionneRemplace(s, pile)
```