α – tri

Question 1 *Donner le résultat de l'appel de la fonction* `scm(s)` *avec l'argument* `s =[2, 2, 1, 8, 1, 7, 9, 2, 2, 4, 4, 0, 7, 7, 9]`.

Correction
`[(0, 1), (2, 3), (4, 6), (7, 10), (11, 14)]`

Question 2 *Donner la complexité algorithmique de la fonction* `scm`.

Correction
La complexité est en $\mathcal{O}(n)$.

Question 3 *En utilisant par exemple une représentation avec indentation, donner le résultat de l'appel suivant (et les résultats intermédiaires) :* `fusionner(s,r[0],r[1])`.

Correction

```
fusionner(s,(0, 2),(3, 4))
     i=0 et s=[5, 3, 4, 8, 1, 2, 7, 9, 0, 10, 0]
     i=1 et s=[1, 5, 3, 4, 8, 2, 7, 9, 0, 10, 0]
   fusionner(s,(0, 1),(2, 4)) vérifie que s[0]<s[2]
   fusionner(s,(1, 1),(2, 4))
     i=0 et s=[1, 8, 5, 3, 4, 2, 7, 9, 0, 10, 0]
     i=1 et s=[1, 4, 8, 5, 3, 2, 7, 9, 0, 10, 0]
     i=2 et s=[1, 3, 4, 8, 5, 2, 7, 9, 0, 10, 0]
   fusionner(s,(1, 3),(4, 4)) vérifie que s[1]<s[4]
   fusionner(s,(2, 3),(4, 4)) vérifie que s[2]<s[4]
   fusionner(s,(3, 3),(4, 4))
     i=0 et s=[1, 3, 4, 5, 8, 2, 7, 9, 0, 10, 0]
     s=[1, 3, 4, 5, 8, 2, 7, 9, 0, 10, 0].
```

Question 4 *Donner la complexité algorithmique de la fonction* `fusionner` *dans le pire des cas puis la complexité de la fonction* `tri`. *Donner le nom d'un algorithme plus performant dans le pire des cas. Préciser sa complexité dans le meilleur des cas et dans le cas moyen.*

Correction La complexité de la fonction `fusionner` dépend de la taille de `r1` et de `r2` que nous notons `n1` et `n2`. Dans le pire des cas, quand le second élément de `r1` est toujours plus grand que le premier élément de `r2`, le calcul de complexité donne (n+1)*n/2. Ainsi la complexité de la fonction `fusionner` est en $\mathcal{O}(n^2)$. La fonction `tri` présente un boucle for avec n appels de la fonction `fusionner` soit une complexité en $\mathcal{O}(n^3)$.

Détection de collisions

Listes non triées

Question 5 *Écrire une fonction* `detecterCollisionEntreParticules(p1,p2)` *qui prend en paramètre deux particules et renvoie* `True` *si les particules sont en collision à l'instant considéré ou* `False` *sinon.*

Correction

```
def detecterCollisionEntreParticules(p1, p2):
    x1, y1, vx1, vy1 = p1
    x2, y2, vx2, vy2 = p2
    return (x2-x1)**2+(y2-y1)**2 <= (2*rayon)**2
```

Question 6 *Écrire une fonction* `maj(particules)` *qui prend en paramètre un ensemble de particules (un triplet comme indiqué plus haut) à l'instant t et renvoie un ensemble contenant des particules à l'instant $t+1$, sans s'occuper des collisions éventuelles.*

Correction

```
def maj(particules):
    largeur,hauteur,donnees=particules
    for i in range(len(donnees)):
        x, y, vx, vy = donnees[i]
        if x+vx<=0 or x+vx>=largeur: # Détection d'un bord latéral
            vx *= -1
        if y+vy<=0 or y+vy>=hauteur: # Détection d'un bord haut bas
            vy *= -1
        donnees[i]=(x+vx, y+vy, vx, vy)
    return ((largeur,hauteur,donnees))
```

Question 7 *À l'aide de la fonction précédente, écrire une fonction* `majOuCollision(particules)` *qui prend en paramètre un ensemble de particules à l'instant t et renvoie un ensemble contenant les particules à l'instant $t+1$, s'il n'y a pas eu de collision à l'instant $t+1$. S'il y a eu une collision la fonction renvoie* `None`.

Correction

```
def majOuCollision(particules):
    nParticules = maj(particules)
    largeur, hauteur, listeParticules = nParticules
    collision = False
    for i in range(len(listeParticules)-1):
        for j in range(i+1, len(listeParticules)):
            if detecterCollisionEntreParticules(listeParticules[i], listeParticules[j]):
                collision = True
    if collision:
        return None
    else:
        return nParticules
```

Question 8 *Écrire une fonction* `attendreCollision(particules, tMax)` *qui prend un ensemble de particules et un temps* `tMax` *en paramètres et renvoie le temps où a eu lieu la première collision entre deux particules. La prise en compte des positions des particules se fait à chaque intervalle de temps* Δt*=1. S'il n'y a pas de collision avant le temps* `tMax`*, la fonction renvoie* `None`*. Quelle est sa complexité, en fonction du nombre* n *de particules et de* `tMax` *? La réponse devra être justifiée.*

Correction

```
def attendreCollision(particules, tMax):
    # O(n * n * tMax)
    t = 0
    while t < tMax and particules != None:
        t += 1
        particules = majOuCollision(particules)
    if t == tMax:
        return None
    else:
        return t
```


Listes triées

Question 9 *Pour que deux particules* ***a*** *et* ***b*** *aient une chance d'entrer en collision à un instant* $t+1$ *donné, à quelle distance, au maximum, devaient-elles se trouver à l'instant* t *? On exprimera le résultat en fonction du* `rayon` *des particules et de leur vitesse maximale* `vMax`*.*

Correction

`dMax=2*(vMax+rayon)`

Question 10 *Écrire la fonction* `majOuCollisionX(particules)`*. Elle prend en paramètre un ensemble de particules dont la liste des particules est triée par abscisses croissantes. Elle renvoie un ensemble contenant les particules à l'instant* $t+1$*, sauf si une collision survient entre deux particules, auquel cas la fonction renvoie* `None`*. Cette fonction devra exploiter le fait que la liste des particules est triée pour limiter le nombre d'appels à la fonction* `detecterCollisionEntreParticules`*.*

Correction

```
def majOuCollisionX(particules):
    nParticules = maj(particules)
    largeur, hauteur, listeParticules = particules
    largeur, hauteur, nlisteParticules = nParticules
    collision = False
    for i in range(len(listeParticules)-1):
        j = i + 1
        while (listeParticules[j][0]-listeParticules[i][0]<=2*(vMax + rayon))
         and (j<len(listeParticules)-1) and not collision:
            if detecterCollisionEntreParticules(nlisteParticules[i], nlisteParticules[j]):
                collision = True
            j += 1
    if collision:
        return None
    else:
        return nParticules
```