α – tri

En préambule on se propose d'étudier un tri basé sur un découpage de listes en « séquences croissantes maximales » d'éléments consécutifs (appelées **scm**). Ces séquences sont croissantes au sens large. Il consiste à effectuer une succession de fusion de **scm** consécutives jusqu'à n'avoir plus qu'une seule **scm**. Fusionner deux **scm** consécutives consiste à réordonner leurs éléments pour ne former qu'une seule **scm** comme pour le tri fusion.

L'algorithme α–tri se déroule en deux temps. On commence par partitionner la liste en **scm** consécutives, en identifiant leurs indices de début et de fin dans la liste. Dans un second temps, on effectue les fusions.

Les fonctions nécessaires au tri sont données ci-dessous.

▪ **Python**

```
def scm(s):
    r = []
    d, f = 0, 0
    for i in range(len(s)-1):
        if s[i] <= s[i+1]:
            f += 1
        else:
            r.append((d,f))
            d = f = f+1
    r.append((d,f))
    return r

def tri(s):
    r = scm(s)
    n = len(r)
    for i in range(n-1):
        fusionnerEnPlaceRec(s, r[0],r[1])
        r = scm(s)
```

```
def fusionner(s, r1, r2):
    d1, f1 = r1
    d2, f2 = r2
    if d1 > f1 or d2 > f2:
        return s
    else:
        assert f1+1 == d2
        if s[d1] <= s[d2]:
            return fusionner(s, (d1+1, f1), r2)
        else:
            for i in range(f2-d2+1):
                t = s.pop(f2)
                s.insert(d1,t)
            f1 = d1 + (f2-d2+1) - 1
            d2 = f1 + 1
            return fusionner(s, (d1,f1), (d2,f2))
```

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]`.

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

On donne la liste `s=[3,4,8,11,1,5,2,7,9,0,10,0]` et `t=[(0, 3), (4, 5), (6, 8), (9, 10), (11, 11)]`.

Question 3 *Donner le résultat de l'appel suivant :* `fusionner(s,0,1)`.

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.*

Détection de collisions

On considère un ensemble de n particules en mouvement dans un espace à deux dimensions, délimité par un rectangle de dimensions (non nulles) `largeur x hauteur`. L'objectif est de faire évoluer le système jusqu'à ce que deux particules entrent en collisions.

On considère que le temps est discret. La simulation commence à $t = 0$ et à chaque étape, on calcule la configuration au temps $t+1$ en fonction de la configuration au temps t.

À tout instant t donné, chaque particule est définie par un quadruplet (x, y, v_x, v_y) où (x, y) sont ses coordonnées réelles représentées par les nombres flottants et où (v_x, v_y) est son vecteur vitesse lui aussi constitué de deux nombres flottants.

Dans tout le sujet, on suppose que la norme de la vitesse de toute particule est majorée par une constante v_{max}. Pour calculer les paramètres au temps $t+1$ d'une particule qui, au temps t est en position (x, y) avec un vecteur vitesse (v_x, v_y), on procède successivement aux traitements suivants

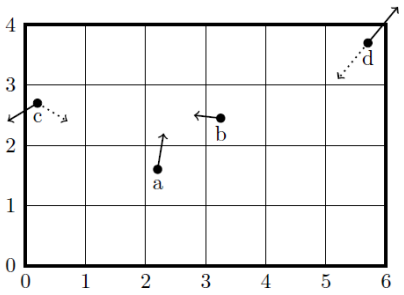
1. si $x + v_x$ atteint ou dépasse une paroi verticale, v_x est changé en $-v_x$ pour simuler le rebond ;
2. si $y + v_y$ atteint ou dépasse une paroi verticale, v_y est changé en $-v_y$ pour simuler le rebond ;
3. (x, y) est changé en $(x + v_x, y + v_y)$.

Les points (1) et (2) simulent de façon simplifiée les rebonds sur les parois : on considère que la particule rebondit à l'endroit où elle est au temps t, ce qui nous permet d'éviter de calculer le véritable point de collision avec la paroi. Il y a rebond lorsqu'une particule arrive exactement sur la paroi ou qu'elle la dépasse. Il est possible qu'une particule rebondisse sur une paroi verticale et horizontale pendant une même mise à jour, ce qui correspond au rebond sur un coin.

R Important : au départ aucune particule n'est sur la paroi. On suppose de plus que $v_{max} < \frac{1}{2}\min(\texttt{largeur}, \texttt{hauteur})$, ce qui garantit que les particules restent toujours strictement à l'intérieur des parois.

▪ **Exemple**

Dans l'exemple ci-dessus, le rectangle est de dimension `largeur x hauteur = 6 x 4`. Les particules **a** et **b** se déplacent sans rebondir au temps $t+1$. La particule **c** est sujette au point (1) comme $x+v_x \leq 0$, elle rebondi sur la paroi, ce que l'on simule en changeant v_x en $-v_x$ avant d'effectuer le déplacement (le nouveau vecteur vitesse est en pointillés). La particule **d** est sujette aux deux points (1) et (2), puisque $x + v_x \geq$ `largeur` et $y + v_y \geq$ `hauteur`, on change v_x en $-v_x$ et v_y en $-v_y$ avant de déplacer cette particule.



▪

Une particule est stockée sous la forme d'un tuple de la forme `particule = (x,y,vx,vy)`. Les particules sont stockées sous forme de listes. Un ensemble de particules est représenté par un triplet `(largeur, hauteur, listeParticules)` tel que `largeur x hauteur` sont les dimensions du rectangle et `listeParticules` est la liste des particules considérées.

▪ **Exemple** Pour la figure ci-dessus, `listeParticules` est de la forme : `(6, 4, [(2.2, 1.6, 0.1, 0.6), (3.25, 2.45, 0.45, 0.05), (0.2, 2.7, -0.5, -0.3), (5.7, 3.7, 0.5, 0.6)])`. ▪

Non triée

Dans un premier temps, il n'y a aucune contrainte sur l'ordre des particules dans la liste.

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

Question 6 *Écrire une fonction* `maj(paricules()` *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.*

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`.

Question 8

Question 9

Question 10

Question 11

Question 12