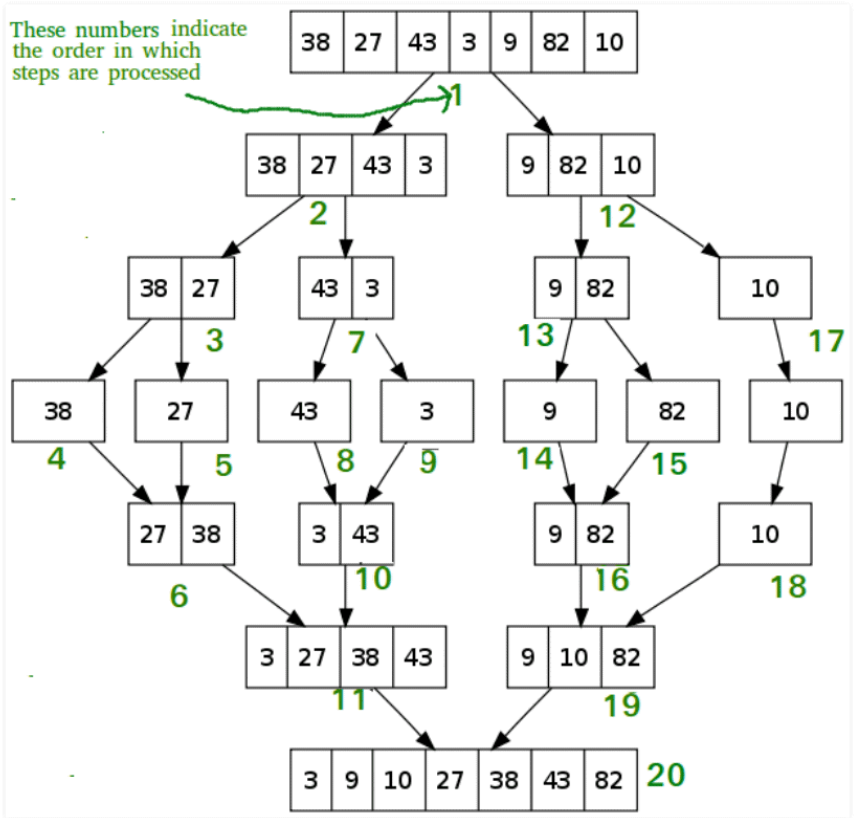
L'algorithme Tri fusion utilise la méthode « diviser pour régner » pour effectuer un tri. C'est l'un des algorithmes les plus performants en matière de tri pour des données quelconques que l'on veut classer.

I- Principe général :

Trier un jeu de carte :
https://www.youtube.com/watch?v=OW3Be5CvYgA

Le tri en danse :
https://www.youtube.com/watch?v=XaqR3G_NVoo

Prenons par exemple la liste suivant :



Le principe du tri est de **diviser** la liste de départ en 2, en répétant l'opération successivement sur chacune des sous listes.

Ensuite on procède à la **fusion** qui consiste à reconstituer une liste en respectant l'ordre de classement.

Le graphe représente la décomposition et la fusion.

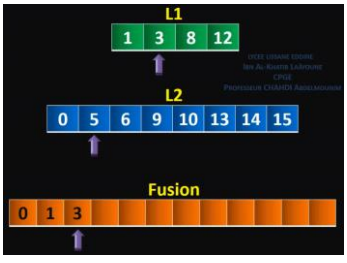
Les numéros en vert (sous les listes) indiquent l'ordre de traitement par le programme (déroulement de l'algorithme)

Pour mieux appréhender le principe, vous pouvez tester le tri fusion sur cette animation interactive :
http://lwh.free.fr/pages/algo/tri/tri_fusion.html

```
TriFusion(tableau,gauche,droite)
If droite > gauche
     1. Trouver le milieu :
            milieu = (gauche+droite)/2
     2. Appeler TriFusion pour moitié gauche:
            TriFusion(tableau,gauche, milieu)
     3. Appeler TriFusion pour moitié droite:
            TriFusion(tableau, milieu+1, droite)
     4. Fusionner les 2 moitiés :
            Fusion(tableau,gauche, milieu, droite)
```

L'algorithme de TriFusion est récursif.

Il fait appel à la fonction Fusion qui elle est itérative (nous le verrons plus loin)



Principe fusion de deux listes triées :
https://www.youtube.com/watch?v=A4uGnjEQsN0

II – Algorithme de fusion

```
1   #
2   # Fusionner deux listes
3   #
4   L1 = [1,3,5,7,float('inf')]
5   L2 = [2,4,5,6,9,float('inf')]
6
7   i=0
8   j=0
9   long_totale = len(L1)+len(L2)-2
10  L3 = [0 for i in range(long_totale)]
11
12  for k in range(long_totale):
13      if L1[i] <= L2[j]:
14          L3[k] = L1[i]
15          i = i+1
16      else:
17          L3[k] = L2[j]
18          j = j+1
19
20  print (L3)
```

On veut fusionner deux listes L1 et L2 : on y a rajouté un nombre très grand (infini) à la fin pour les besoins de l'algorithme.

On créer une liste L3 remplie de 0 qui contiendra la liste triée.

Exo (compléter ci-dessous) :
Donner les valeurs de :
L1[i], L2[j] ,L3[k] , i et j pour les 4 premières étapes du programme.
Indiquer à la fin de chaque itération le contenu de la liste L3

Indiquer directement sur le programme les valeurs, préciser par Vrai ou Faux le test du if et barrer la partie du code non exécutée:

k=0

```
for k in range(long_totale):
    if L1[i] <= L2[j]:
        L3[k] = L1[i]
        i = i+1
    else:
        L3[k] = L2[j]
        j = j+1
```

L3 =

k=2

```
for k in range(long_totale):
    if L1[i] <= L2[j]:
        L3[k] = L1[i]
        i = i+1
    else:
        L3[k] = L2[j]
        j = j+1
```

L3 =

k=1

```
for k in range(long_totale):
    if L1[i] <= L2[j]:
        L3[k] = L1[i]
        i = i+1
    else:
        L3[k] = L2[j]
        j = j+1
```

L3 =

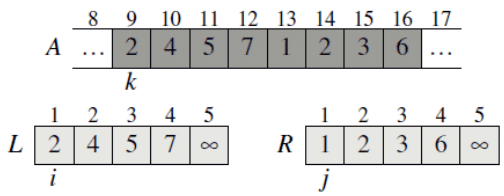
k=3

```
for k in range(long_totale):
    if L1[i] <= L2[j]:
        L3[k] = L1[i]
        i = i+1
    else:
        L3[k] = L2[j]
        j = j+1
```

L3 =

III- Voici l'algorithme général du tri fusion :

```
FUSION(A, p, q, r)
1  n1 ← q − p + 1
2  n2 ← r − q
3  créer tableaux L[1 .. n1 + 1] et R[1 .. n2 + 1]
4  pour i ← 1 à n1
5      faire L[i] ← A[p + i − 1]
6  pour j ← 1 à n2
7      faire R[j] ← A[q + j]
8  L[n1 + 1] ← ∞
9  R[n2 + 1] ← ∞
10 i ← 1
11 j ← 1
12 pour k ← p à r
13     faire si L[i] ⩽ R[j]
14               alors A[k] ← L[i]
15                    i ← i + 1
16               sinon A[k] ← R[j]
17                    j ← j + 1
```



Pour transcrire cet algo en python, il faut faire attention à deux choses : les tableaux (ou liste) en python commencent avec l'indice 0. De plus lorsqu'on fait for i in range(0,n) en python , la boucle s'arrête à n-1 , donc il faut là encore adapter…

Signifie « partie entière »

```
TRI-FUSION(A, p, r)
1  si p < r
2     alors q ← ⌊(p + r)/2⌋
3           TRI-FUSION(A, p, q)
4           TRI-FUSION(A, q + 1, r)
5           FUSION(A, p, q, r)
```

```python
def tri_fusion(A,p,r):
    if p<r:
        q = (p+r)//2          # diviser
        tri_fusion(A,p,q)     # régner
        tri_fusion(A,q+1,r)   # régner
        fusion(A,p,q,r)       # combiner (fusionner)
```

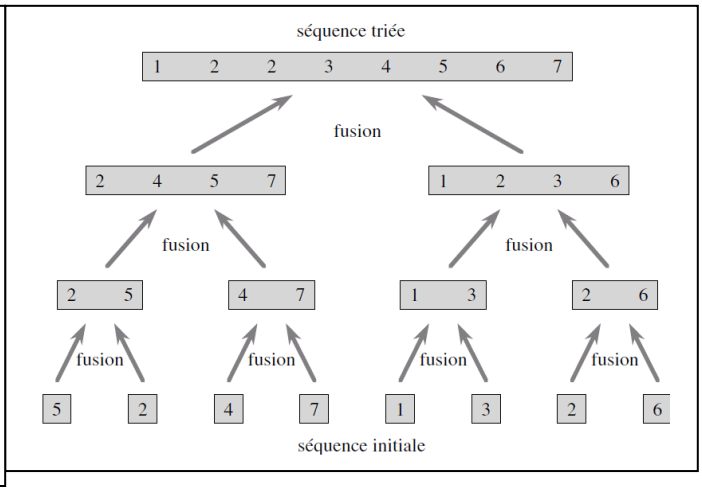
```python
def fusion(A,p,q,r):
    n1 = q-p+1
    n2 = r-q
    L = [0 for i in range(0,n1+1)]
    R = [0 for i in range(0,n2+1)]

    for i in range(0,n1):  # pour i = 1 à n1
        L[i] = A[p+i]

    for j in range(0,n2):   # pour j = 1 à n2
        R[j] = A[q+j+1]

    L[n1] = float('inf')
    R[n2] = float('inf')

    # ici démarre la fusion des listes
    # L et R sont triés, il faut les fusionner
    i = 0
    j = 0

    for k in range(p,r+1): # pour k = p à r
        if L[i]<=R[j]:
            A[k] = L[i]
            i = i+1
        else:
            A[k] = R[j]
            j = j+1
```

On peut représenter l'infini en python

La compréhension dans le détail de ce programme n'est pas demandée. Vous devez comprendre la philosophie générale du tri-fusion (page 1) ainsi que le principe de fusion vu en page 2.

IV- Complexité du programme

La complexité est ici en O(n * $\log_2$(n)). Pour la déterminer, il faut connaître le nombre de données parcouru pour chaque niveau.
Lors de la phase de division (en descente), il n'y pas vraiment de « données parcourues », on se contente simplement de calculer des milieux.
Par contre lors de la phase de remontée, on parcourt à chaque niveau toute la liste (mais par beaucoup de petits bouts au début, puis par blocs de plus en plus grands mais moins nombreux)
Le nombre de données parcourues est donc de n * nombre de niveaux = n*$\log_2$(n) . On avait déjà vu lors de l'étude de l'algorithme de recherche dichotomique que le nombre de niveaux d'une structure de largeur n qui se divise par 2 à chaque fois , est $\log_2$(n)



V- Travail à faire :

Taper le programme Tri-Fusion en python de la page 3.
Tester et valider son fonctionnement en testant avec la liste A ci-dessous. On passe à la fonction l'indice de départ de la liste et l'indice de fin de liste (correspond donc à len(A)-1).

```
A=[1,8,4,10,0,7,55,1,3,3,0,0,25,10,12,33]
tri_fusion(A,0,len(A)-1)
print (A)
```

Résultat :

```
[0, 0, 0, 1, 1, 3, 3, 4, 7, 8, 10, 10, 12, 25, 33, 55]
```

On souhaite comparer la vitesse d'exécution du tri-fusion avec une méthode de tri que vous avez déjà vu, par exemple le tri par sélection ou encore le tri par insertion.
Pour mesurer le temps d'exécution d'une fonction, vous devez importer la bibliothèque « time » : (import time) et utiliser ensuite le code comme suit :

```
start = time.time()
tri_fusion(l,0,len(l)-1)
duree = time.time() - start
```

Il suffit ensuite d'afficher la valeur de duree.

En testant sur de petites listes, il est fort probable que vous obteniez 0,0 pour duree car l'exécution est trop rapide. Il faut donc générer de très grandes listes (longueur 100 , 1000, 2000 …) pour pouvoir faire des mesures.

→ **Faire une fonction** genintlist(n) qui prend en paramètre un entier n et qui renvoie une liste de longueur n constituée de nombres aléatoires compris entre 0 et n . Vous devez importer la bibliothèque « random » et utiliser l'instruction random.randint(0,n) qui génère un nombre aléatoire entre 0 et n.

→ **Faire ensuite un programme** qui va injecter cette liste aléatoire aux fonctions tri_fusion , tri_insertion , et tri_selection et afficher les temps pour chacun. Attention, enlever les « print » éventuels dans les codes de vos programmes de tri sinon les temps sont faussés !

→ **Faire des essais** sur plusieurs longueurs de listes et faire un tableau comparatif de vos résultats. Une représentation graphique de vos résultats sera très apprécié (excel, matplot de python ou a la main au pis)

Le travail sera à rendre pour le :