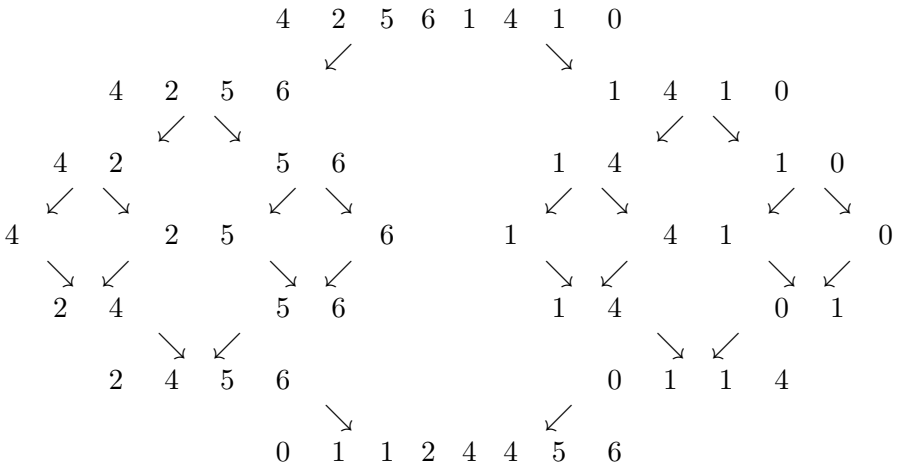
EA4 – Éléments d'algorithmique
TD n° 6 : permutations et tris
(Correction)

Exercice 3 : tri fusion

1. Appliquer à la main l'algorithme de tri fusion sur le tableau `[4,2,5,6,1,4,1,0]`.

▷



2. Écrire `fusionIterative(T1, T2)` une version itérative (de complexité linéaire) de la fonction de fusion de tableaux.

▷ *On parcourt les deux listes simulatnément de gauche à droite et on insère dans la liste résultat* `res` *le plus petit des deux éléments. Il ne faut pas oublier lorsqu'on est arrivé à la fin d'une liste, d'insérer les éléments non encore vus de l'autre liste.*

```
def fusionIterative(T1,T2):
  res = []
  i = 0
  j = 0
  while (i < len(T1) and j < len(T2)):
    if T1[i] <= T2[i]:
      res += [ T1[i] ]
      i += 1
    else:
      res += [ T2[i] ]
      j += 1
  for k in range(i, len(T1)):
    res += [ T1[k] ]
  for k in range(j, len(T2)):
    res += [ T2[k] ]
  return res
```

3. Modifier la fonction précédente pour compter le nombre d'inversions dans le tableau `T1+T2` (en supposant `T1` et `T2` triés).

▷ *Lors de la fusion de* `T1` *et* `T2`*, lorsqu'on compare* `T1[i]` *et* `T2[j]` *:*

- *si* `T1[i] <= T2[j]` *alors* `T1[i] <= T2[k]` *pour tout* $k >= j$ *car* `T2` *est trié. Donc les paires* `(i,k)` *ne sont pas des inversions pour tout* $k >= j$,
- *si* `T1[i] > T2[j]` *alors* `T1[k] > T2[j]` *pour tout* $k >= i$ *car* `T1` *est trié. Donc les paires* `(k,j)` *sont des inversions pour tout* $k >= i$.

```
def fusionIterative(T1,T2):
  res = []
  i = 0
  j = 0
  nbInv = 0 # NEW
  while (i < len(T1) and j < len(T2)):
    if T1[i] <= T2[j]:
      res += [ T1[i] ]
      i += 1
    else:
      res += [ T2[j] ]
      j += 1
      nbInv += len(T1) - i # NEW
  for k in range(i, len(T1)):
    res += [ T1[k] ]
  for k in range(j, len(T2)):
    res += [ T2[k] ]
  return res, nbInv
```

Exercice 4 : opérations ensemblistes

On représente ici des ensembles d'entiers par des tableaux *triés* et *sans doublon*. En vous inspirant de l'algorithme de fusion de deux listes triées, décrire des algorithmes permettant de calculer :

▷ *Pour chaque question, l'idée est de s'appuyer sur l'algorithme de fusion de deux listes triées, et de décider, pour chaque comparaison effectuée, si on procède ou non à un ajout dans la liste résultat ; dans le cas où les deux éléments sont identiques, les deux curseurs sont incrémentés quoi qu'il arrive pour préserver l'absence de doublon.*
Regardez comment se fait l'union et procédez dans le même esprit pour les autres questions.

1. l'*union* de deux ensembles : $E \cup F = \{x \mid x \in E \text{ ou } x \in F\}$,

▷ *Dans ce cas, on garde tous les éléments, sans doublon.*

```
def union(L1, L2) :
  res = []
  i = 0
  j = 0
  while (i < len(L1) and j < len(L2)):
    if L1[i] == L2[j] :
      res += [ L1[i] ]
      i += 1
      j += 1
    elif L1[i] < L2[j]:
      res += [ L1[i] ]
      i += 1
    else:
```

```
    res += [ L2[j] ]
    j += 1
for k in range(i, len(L1)):
  res += [ L1[k] ]
for k in range(j, len(L2)):
  res += [ L2[k] ]
return resel
```

2. l'*intersection* de deux ensembles : $E \cap F = \{x \mid x \in E \text{ et } x \in F\}$,

▷ *Dans ce cas, on garde seulement les éléments présents dans les deux listes : lorsque les deux éléments comparés sont distincts, le plus petit est éliminé.*

3. la *différence* de deux ensembles : $E \setminus F = \{x \mid x \in E \text{ et } x \notin F\}$,

▷ *Ce cas est dissymétrique, puisque seuls (certains) éléments de E sont conservés.*

4. la *différence symétrique* de deux ensembles (ensemble formé des éléments appartenant à l'un des deux ensembles, mais pas aux deux) : $E \Delta F = \{x \mid x \in E \text{ ou (exclusif) } x \in F\}$.

▷ *Ce cas est l'opposé de l'intersection : un élément n'est conservé que s'il est plus petit que celui auquel il est comparé.*