# EA4 – Éléments d'algorithmique
# TD n° 5 : permutations – tri par fusion

### Exercice 1 : permutations (décomposition, produit, inverse)

On considère les permutations suivantes :

$$\sigma = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 5 & 9 & 7 & 2 & 1 & 3 & 10 & 8 & 4 & 6 \end{pmatrix} \text{ et } \tau = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 1 & 5 & 8 & 4 & 2 & 7 & 6 & 3 & 9 & 10 \end{pmatrix}.$$

1. Écrire $\sigma$ et $\tau$ en notation cyclique (c'est-à-dire comme produits de cycles disjoints).
2. Calculer $\sigma\tau$ et $\tau\sigma$.
3. Calculer $\sigma^{-1}$, $\tau^{-1}$ et $(\sigma\tau)^{-1}$.
4. On appelle *transposition* une permutation ayant un unique cycle de longueur 2 (et donc $n-2$ points fixes). Décomposer $\sigma$ en produit de transpositions de deux façons différentes.

### Exercice 2 : permutations (encodage)

On peut modéliser les permutations par des tableaux d'entiers : un tableau `T` de longueur $n$ représente une permutation si et seulement s'il contient tous les entiers compris entre 1 et $n$ (nécessairement exactement une fois chacun).

1. Écrire une fonction `estPerm` qui prend en paramètre un tableau d'entiers `T` et qui renvoie `True` si `T` est une permutation et `False` sinon.
2. Écrire une fonction `inversePerm` qui prend en paramètre un tableau d'entiers `T` représentant une permutation et qui renvoie l'inverse de cette permutation. Si `T` n'est pas une permutation, la fonction doit renvoyer `None`. Vérifier ou modifier la fonction, de manière à s'assurer que la vérification se fait en temps linéaire.
3. Écrire une fonction `composePerm` qui prend en paramètre deux tableaux d'entiers `T1` et `T2` représentant deux permutations de même taille, et qui renvoie la composée de ces deux permutations. Si les deux permutations ne sont pas de même taille ou si l'un des deux tableaux n'est pas une permutation, la fonction doit renvoyer `None`.

### 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]`. Compter précisément le nombre de comparaisons effectuées.
2. Écrire une version itérative `fusionIterative(T1, T2)` (de complexité linéaire) de la fonction de fusion de tableaux.
3. On rappelle qu'une *inversion* de `T` est un couple d'éléments `T[i] < T[j]` mal ordonnés, c'est-à-dire dont les positions vérifient $i > j$.

   Modifier la fonction précédente en une fonction `nbInversionsEntre(T1, T2)` qui compte les inversions dans le tableau `T1+T2` (en supposant `T1` et `T2` triés).
4. En déduire un programme `nbInversions(T)` qui calcule le nombre d'inversions d'un tableau `T` en temps $\Theta(n \log n)$.