# TD informatique du chapitre 15 : Algorithmes de tri 2

**Exercice 1 : Tri fusion**

Le tri fusion est basé sur la stratégie "diviser pour régner" : on divise le problème en plusieurs sous-problèmes que l'on va résoudre.

Le principe est le suivant :

- on partage la liste en deux (à une unité près),
- on trie les deux sous-liste obtenues par un argument récursif,
- on fusionne les deux listes triées.

La question est donc maintenant de fusionner les deux listes triées de façon à obtenir une liste qui soit toujours triée. Pour cela, on utilise le principe suivant :

- on compare les plus petits éléments des deux listes que l'on appellera $l1$ et $l2$, on sélectionne le plus petit des deux,
- on ne considère plus le terme sélectionné et on réitère le processus.

Par exemple, pour $l1 = [1,5,7]$ et $l2 = [2,5]$ :

- on compare 1 dans $l1$ et 2 dans $l2$ et on garde 1 : [1],
- on ne considère plus 1 dans $l1$ : $l1 = [\cancel{1},5,7]$ et $l2 = [2,5]$,
- on compare 5 dans $l1$ et 2 dans $l2$ et on garde 2 : [1,2],
- on ne considère plus 2 dans $l2$ : $l1 = [\cancel{1},5,7]$ et $l2 = [\cancel{2},5]$,
- on compare 5 dans $l1$ et 5 dans $l2$ et on garde 5 : [1,2,5],
- on ne considère plus 5 dans $l1$ (choix arbitraire) : $l1 = [\cancel{1},\cancel{5},7]$ et $l2 = [\cancel{2},5]$,
- on compare 7 dans $l1$ et 5 dans $l2$ et on garde 5 : [1,2,5,5],
- on ne considère plus 5 dans $l2$ : $l1 = [\cancel{1},\cancel{5},7]$ et $l2 = [\cancel{2},\cancel{5}]$,
- il n'y a plus d'élément à comparer dans l2, on rajoute les éléments restant dans l1 : [1,2,5,5,7] qui est bien la liste triée.

1. Ecrire une fonction `fusion` qui prend comme argument deux listes triées `l1` et `l2` et qui renvoie la liste triée fusionnée en utilisant le principe précédent.
2. Ecrire une fonction récursive qui effectue un tri fusion.
3. Calculer la complexité du tri fusion (on pourra commencer par le cas où $n$ est une puissance de 2).

**Exercice 2 : Tri rapide**

Comme pour le tri fusion, on utilise la stratégie "diviser pour régner". La liste sera découpée en deux parties : les termes plus petits et les termes plus grand qu'une valeur choisie appelée pivot. Le pivot que l'on considèrera sera le premier terme de la liste.

1. Ecrire une fonction `partition` qui prend comme arguments une liste `l` et deux entiers `i` et `j` et qui :
   - modifie la liste `l` de façon à ce que, entre les indices $i$ et $j$ inclus on ait d'abord les éléments inférieurs à $l[i]$ puis $l[i]$ puis les éléments supérieurs à $l[i]$,
   - renvoie la position de $l[i]$.

   Pour cela, on utilisera les étapes suivantes :
   - on parcourt les éléments de la liste à partir de la gauche avec un indice $g$ (qui commence à i+1) et à partir de la droite avec un indice $d$ (qui commence à j),
   - on compare les éléments d'indices $g$ et $d$ au pivot et on s'arrête lorsque la comparaison n'est pas dans le bon sens,
   - on échange alors les éléments d'indices $g$ et $d$,
   - on réitère ce procédé jusqu'à ce que $g$ et $d$ se croisent,
   - on positionne le pivot en échangeant les termes de position $d$ et $i$.

   Par exemple :
   ```
   >>> l=[1,5,3,8,2,4,7,0,3]
   >>> partition(l,1,6)
       4
   >>> l
       [1, 2, 3, 4, 5, 8, 7, 0, 3]
   ```
2. Ecrire une fonction récursive `TriRapide_Aux` qui prend comme arguments une liste `l` et deux entiers `g` et `d` et qui trie les termes d'indices compris entre $g$ et $d$ en utilisant la partition précédente.
3. Ecrire une fonction `TriRapide` qui prend comme arguments une liste `l` et qui renvoie la liste triée en utilisant `TriRapide_Aux`.

On peut montrer que la complexité du tri rapide est $O(n^2)$ dans le pire des cas. Cependant sa complexité en moyenne est $O(n \ln n)$ d'où le qualificatif de rapide. Contrairement au tri fusion, il est en place et est donc plus optimal au niveau de la mémoire utilisée.