TP4 : Tris et Diviser pour régner

Dans ce TP, nous nous intéressons à des problèmes relatifs aux tris, afin d'illustrer la méthode *Diviser pour régner*.

Exercice 1 (Quelques tris quadratiques)**.**
Pour commencer, nous commençons par implémenter des tris différents, ayant tous une complexité quadratique dans le pire cas.

1. *Tri bulle.* Écrire une fonction `triBulle(t)` qui prend en entrée un tableau `t`, et qui le trie en place en utilisant le principe suivant. On parcourt les cases de `t`, et on échange les éléments en positions i et $i+1$ si `t[i]>t[i+1]`. On répète cette opération jusqu'à ce qu'aucune inversion ne soit possible.

   Prouver (par récurrence) qu'après i répétitions de l'opération, les i derniers éléments de `t` sont à leur position correcte. En déduire la complexité du tri bulle.
2. *Tri par sélection.* Écrire une fonction `triSelection(t)` qui prend en entrée un tableau `t`, et qui le trie en place en utilisant le principe suivant. Pour chaque $i \geq 0$, on cherche le minimum du tableau `t[i:]`, et on le place en position i.
3. *Tri rapide.* Écrire une fonction récursive `triRapide(t)` qui prend en entrée un tableau `t`, et qui le trie en place en utilisant le principe suivant. Si `t` est de taille au plus 1, alors il est trié. Sinon, on choisit $x_0 \leftarrow$ `t[0]` comme *pivot*, et on réorganise (en temps linéaire) `t` de sorte qu'il commence par ses éléments inférieurs à x_0 (cela forme le sous-tableau `t0`), et termine par ses éléments supérieurs à x_0 (cela forme le sous-tableau `t1`). On applique ensuite récursivement le tri rapide à `t0` et à `t1`.

   Donner un exemple de tableau `t` sur lequel le tri rapide a une complexité quadratique.

Exercice 2 (Le tri fusion)**.**
Écrire une fonction *triFusion(l)* qui prend en entrée un objet `l` de la classe `Liste` définie lors du TP2, et retourne la liste triée contenant les éléments de `l`, en utilisant le tri fusion. On pourra réutiliser la fonction de fusion de deux listes chaînées triées implémentée à la fin du TP2.

Exercice 3 (Compter les inversions dans une permutation)**.**
On considère une permutation P de n éléments, représentée dans un tableau. On cherche à évaluer le nombre d'inversions au sein de P, c'est à dire le nombre de paires $(i, j) \in [n]^2$ telles que $i < j$ mais j apparaît avant i dans P.

1. Écrire une fonction `nombreInversions(P)` de complexité quadratique qui compte le nombre d'inversions dans la permutation représentée par le tableau `P`.
2. On cherche maintenant à améliorer l'algorithme précédent en utilisant le paradigme Diviser pour Régner. Pour cela, on coupe la permutation P en deux sous-permutations P_0 et P_1. On compte alors les inversions (i, j) qui peuvent être de deux types : soit i et j apparaissent dans la même sous-permutation, soit i apparaît dans une sous-permutation et j apparaît dans l'autre (inversion mixte).
   (a) Écrire une fonction `inversionsMixtes(P0,P1)` de complexité linéaire qui prend en entrée deux permutations `P0` et `P1` **triées**, et compte le nombre d'inversions mixtes entre `P0` et `P1`.

(b) Écrire une fonction `nombreInversionsRec(t)` qui utilise le paradigme *Diviser pour Régner* afin de compter le nombre d'inversions de n'importe quelle permutation `P` de n élements en temps $O(n \log n)$.