EPU GB4-BIMB **TD d'Algorithmique pour la biologie n.1** janvier 2024

Premiers algorithmes de Tris

Exercice 1 : (**tri par insertion**)

Le tri par insertion permet de trier un tableau ou une liste en plaçant successivement chacun des éléments à la bonne place. Dans la version vue en cours, le tri est effectué sur place, on est donc amené à déplacer certains éléments.

1. Programmez la version non récursive, vue en cours, qui effectue le tri sur place. Testez votre fonction sur plusieurs exemples.
2. Implémentez une fonction récursive qui insère un élément dans une liste supposée triée.
3. En utilisant la fonction précédente, implémentez une version récursive du tri par insertion.
4. Quelle est la version la plus simple à comprendre ?
5. Est-ce que ces différentes versions ont même complexité ?

Exercice 2 : (**tri par fusion, A RENDRE**)

L'algorithme de tri par fusion est basée sur l'approche diviser pour régner. Il est donc fondé sur la récursivité. Nous allons donner une implémentation en Python de cet algorithme.

1. Ecrire d'abord la fonction `fusionner` qui prend en entrée 2 listes triées, et renvoie la liste triée contenant tous les éléments des deux listes.
2. Ecrire la fonction `triFusion`

Exercice 3 : (**A RENDRE**)

Dans cet exercice, nous allons tester la complexité temporelle des tris par insertion et par fusion.

1. En utilisant le module `random` (instruction `import random` en début de fichier, puis `random.randint(min,max)`) générez une liste d'entiers d'au moins 1000 valeurs.
2. Faites tourner les 2 algorithmes sur cette liste.
3. Observe-t-on un temps de réponse différent pour les 2 algorithmes ? A partir de quelle taille de liste, cette différence est bien visible ? Pour comparer les temps d'exécution, on utilisera le module `time` : instruction `import time` en début de fichier, puis `time.time()`.
4. Génerez des listes d'entiers aléatoires de tailles allant de 10 à 2000 par pas de 10, triez-les par insertion et par fusion, puis tracez les 2 courbes des temps d'excecution de ces 2 algorithmes en fonction des longueurs des listes.

Remarque : On pourra utiliser `sys.setrecursionlimit(n)` du module `sys` qui permet d'augmenter la taille de la pile de récursion.