ITC – TD

Algorithmes de tris – 1

On se propose au cours de quelques séances de TD de tester et approfondir notre étude des tris classiques vus en cours. Comme dans le cours, on utilisera des numpy array afin d'avoir un comportement naturel pour les tris en place récursifs.

Un fichier est fourni avec le TD à préparer : il contient des en-têtes de fonctions et procédures, à modifier. Ainsi vous avez une docstring à respecter pour chaque fonction/procédure. Lisez les commentaires pour savoir où compléter ; en particulier, vous pouvez retirer les « return » des procédures une fois celles-ci écrites.

I. Premiers tris

1 – Coder une procédure `tri_insertion` qui trie une liste en place selon l'algorithme par insertion.

2 – Coder une procédure `tri_fusion` qui trie une liste **en place** selon l'algorithme de partition-fusion.

II. Mise en place : tests et benchmarking

3 – Écrire une fonction `générer_tab_test` qui prend en paramètre des entiers n, $mini$ et $maxi > mini$, et qui renvoie une liste de taille n d'entiers compris dans l'intervalle $[mini; maxi[$. On rappelle que `randint(a, b)` renvoie un entier aléatoire dans l'intervalle $[a; b]$.

4 – Proposer un jeu de tests pour un algorithme de tri en place, et les ajouter dans la procédure `tester_tri` prenant en paramètre une procédure de tri, partiellement fournie.

5 – Tester la validité les deux premières procédures `tri_insertion` et `tri_fusion`.

6 – Écrire une fonction `perf_algo` qui prend en paramètres une procédure de tri `algo` à évaluer (par exemple `tri_insertion`), un entier `taille`, un entier `val_max` et un entier `n_tests` ; et qui effectue `n_tests` évaluations du temps de calcul de la procédure `algo` sur des tableaux de taille `taille` de nombres entiers générés aléatoirement entre 0 et `val_max`. La fonction `perf_algo` renvoie le temps moyen obtenu. On pourra mesurer le temps de calcul pour un tableau donné avec :

```
start = perf_counter()           # on démarre le chrono
algo(tab)                        # on exécute la procédure algo sur l'entrée tab
perf += perf_counter() - start   # on arrête le chrono
```

suite à ces instructions, la durée (en secondes) d'exécution de la ligne 2 est stockée dans `perf`.

7 – En utilisant la fonction précédente, écrire une fonction `mesure_complexité` qui prend en paramètre une procédure `algo`, une liste de tailles, une valeur maximale et un nombre de tests, et qui renvoie la liste des performances moyennes de l'algorithme pour chaque taille de tableau.

8 – Lancer les instructions à la fin du fichier python fourni, et commenter le résultat obtenu.

III. Si le temps : amélioration du tri par insertion

Lors du tri par insertion, on insère le $j-^{e}$ élément parmi les éléments $0 \dots j-1$ déjà triés. On peut tirer parti de cette propriété pour chercher la position à laquelle insérer l'élément j par dichotomie.

9 – Écrire la procédure `tri_insertion_dicho` et vérifier qu'elle passe les tests précédemment écrits.

10 – Ajouter cette procédure à la liste des bechmark. Commenter.