ITC – TD

Algorithmes de tris – 2 : quicksort & co

On se propose au cours de quelques séances de TD de tester et approfondir notre étude des tris classiques vus en cours.

Repartez de votre fichier de travail du TD précédent, avec les sections I et II complétées : on supposera ici que les fonctions précédemment définies sont disponibles.

I. Le tri rapide

1 – Écrire une fonction `partition` qui partitionne un tableau en vue d'un tri rapide. On prendra comme pivot la première ou la dernière case du tableau.

2 – Écrire la procédure `tri_rapide` et la tester.

3 – Ajouter cette fonction à la liste des benchmarks et commenter le résultat obtenu. À partir de quelle taille de tableau est-il plus performant que le tri par insertion ?

II. Première modification

Afin d'éviter le pire cas (tableau trié), on peut choisir aléatoirement le pivot.

4 – Écrire une fonction `partition_aléa` qui partitionne un tableau en vue d'un tri rapide, en choisissant aléatoirement le pivot, puis écrire la procédure `tri_rapide_aléa` correspondante et la tester.

5 – Ajouter cette fonction à la liste des benchmarks et commenter le résultat obtenu.

III. Combinaison de tris

Puisque le tri par insertion est plus efficace pour les petits tableaux ou des tableaux presque triés, on se propose de l'utiliser en combinaison du tri rapide afin d'éviter quelques appels récursifs.

6 – Écrire une procédure `tri_combiné` qui adapte le tri rapide en prenant pour cas de base : « si la taille du tableau est < 10, trier par insertion ». On pourra écrire une procédure `tri_insertion_bornes` pour trier par insertion un sous-tableau.

7 – La variante de Sedgewick propose de prendre le tri rapide avec pour cas de base « si la taille du tableau est < 10, ne rien faire » puis de faire un tri par insertion une fois les appels récursifs terminés (et donc le tableau presque trié). Écrire une procédure `tri_sedgewick` qui applique cette stratégie.

8 – Tester puis bechmarker ces tris en comparaison à `tri_rapide` ; commenter.