

Fiche 5.0 : complexité des algorithmes

BUT Informatique
IUT de Vélizy

0 Tris

Dans cette partie, vous allez écrire et tester *deux tris célèbres et assez simples à mettre en oeuvre*. Puis vous mesurerez leurs performances. Enfin, vous les comparerez au tri natif de python.

Objectif : apprendre à mesurer et augmenter les performances de ses algorithmes.

Vous travaillerez dans le fichier `tri_sort.py`, dans lequel vous progresserez au fur et à mesure de la lecture de cet énoncé.

Génération de listes aléatoires

- 0. **Fonction de génération de liste aléatoire** Afin de nourrir vos algorithmes de tri, vous allez écrire une fonction `randlist` qui, pour un entier `n` donné, renvoie une liste de `n` entiers compris entre 0 et `n-1` et mélangés aléatoirement (faites appel à la fonction `shuffle()`).
Exemple de fonctionnement : `randlist(4)` pourrait renvoyer `[3, 1, 0, 2]`.
- 1. **Test de `randlist()`** Une fois écrite cette fonction, complétez `test_randlist()` et vérifiez que `randlist()` fonctionne correctement. **Vous devez écrire au moins 3 appels de test variés.**

Tri par sélection

- 2. **Découverte du tri par sélection** Le principe du tri par sélection est assez simple :
 - (a) Rechercher dans la partie de la liste qui n'est pas encore triée la position de la valeur minimale.
 - (b) Placer cette valeur en début de tableau en effectuant un échange des deux positions concernées.
 - (c) Recommencer à l'étape (a) jusqu'à ce que toute la liste soit triée.Pour vous aider à bien comprendre le principe de ce tri, rendez-vous sur Visualgo, sélectionnez dans la barre du haut *SEL* (comme *selection sort*, le nom anglais du tri par sélection), et lancez le tri en cliquant sur *Sort* en bas à gauche.
- 3. **À vous de sélectionner** Complétez la fonction `tri_sel()`.
- 4. **Le moment de vérité** Complétez la fonction `test_tri_sel()` qui va vous permettre de tester `tri_sel()` avec des listes aléatoires générées par `randlist()`. Effectuez également des tests avec des cas particuliers : **trouvez-en au moins trois.**

Tri à bulles

- 5. **Découverte des bulles** Rendez-vous à nouveau sur Visualgo afin de bien comprendre ce que fait le tri à bulles.
- 6. **À vous de buller** Codez le tri à bulles dans la fonction `tri_bul()`. Vous penserez à ajouter quelques lignes afin de vérifier s'il est possible d'interrompre l'algorithme quand on détecte que le tableau est trié avant la fin des boucles.
- 7. **Test des bulles** Complétez la fonction `test_tri_bul()` afin de tester votre tri. Cette fonction peut être une simple copie de la fonction `test_tri_sel()`.

1 Mesures de performances

1.1 Temps de calcul

- 8. **Chrono reverse** Dans le fichier `cpx_sort.py`, l'exercice 8 vous montre comment mesurer le temps requis par l'exécution de certaines instructions données. Appelez cette fonction `perf_reverse()` et observez son fonctionnement.
- 9. **Perf des tris** Vous allez écrire une fonction `perf_tris()` qui va mesurer et stocker les performances de vos deux tris pour un nombre variable d'entiers. Par exemple, vous pourriez lancer vos tris sur des tableaux de taille 100, 200, 300, ..., 2900, 3000. Vous stockerez les temps obtenus dans des listes, dans l'ordre d'obtention des résultats.

1.2 Graphiques

- 10. **Démo pyplot** Dans le fichier `cpx_sort.py`, l'exercice 10 vous montre comment réaliser un affichage graphique des données d'une liste. Appelez cette fonction `demo_pyplot()` et observez son fonctionnement.
- 11. **Plotez** Vous êtes maintenant un as de `pyplot`. Si si. Vous pouvez compléter votre fonction `perf_tris()` en y ajoutant l'affichage graphique des performances mesurées pour vos deux tris.
- 12. **Bravo** Bravo.

1.3 Comparatif

- 13. **Sa majesté Quick sort** Nous allons maintenant nous mesurer à un adversaire de taille : l'algorithme *quick sort* (*tri rapide* en français) implémenté dans la méthode `sort()` native des listes de python. Commencez par aller jeter un oeil sur le travail étrange que fait quick sort dans Visualgo.
- 14. **The race** Dans votre fonction `perf_tris()`, ajoutez un appel et une mesure des performances de quicksort. Appelez, mesurez, affichez, pleurez.

2 Mini-projet

Vous êtes en avance ? Nous vous invitons à donner un coup de main à vos camarades, ou à entamer le mini-projet qui suit. Son principe : coder soi-même un *tri quick sort*.
Surprise garantie si vous y parvenez.

2.1 Principe

Le principe général du *quick sort* est :

- pour un tableau donné, choisir un pivot quelconque, c'est-à-dire une des valeurs du tableau ;
- retirer temporairement ce pivot du tableau ;
- placer toutes les valeurs inférieures ou égales au pivot en début de tableau et toutes les valeurs supérieures à la fin, puis replacez le pivot à la place restante ;
- à cet instant, **le pivot se trouve à sa place définitive** ;
- relancez récursivement ces étapes sur les deux sous-tableaux restants.

Encore une fois, vous êtes invité.e.s à visionner le fonctionnement de ce tri sur Visualgo.

2.2 Pivot

Codez une fonction `pivot(l, deb, fin)` qui, pour le sous-tableau de `l` compris entre `deb` et `fin` inclus, choisit un pivot aléatoirement puis place les valeurs de `l` à gauche et à droite du pivot.

Attention : cette fonction est cruciale pour les performances du *quick sort*. Vous devez réussir à effectuer le classement en un seul passage, sans déplacer chaque valeur plus d'une fois.

2.3 Fonction principale

Bonne nouvelle, votre travail est quasiment terminé. En effet, il ne vous reste plus qu'à écrire une fonction récursive `monqs(l, deb, fin)` de quelques lignes.

2.4 Performances

Comme dans les sections précédentes, mettez en place une mesure de performance de votre tri ainsi qu'un affichage graphique. Vous devez obtenir des performances bien meilleures que celles de vos tris quadratiques.

Nous verrons en TD pourquoi, et à quelles conditions, cet algorithme est si performant.

2.5 Surprise

Comparez graphiquement votre tri au *quick sort* natif de python. **Caramba, que pasa ?**