Tris et complexité

L'objectif de ce TP est d'implémenter les trois algorithmes de tri au programme (tri par insertion, tri rapide et tri par fusion), puis de mesurer leurs temps d'exécution dans différents cas, ce qui permettra d'illustrer différentes nuances de complexité en temps, l'étude théorique étant faite en cours.

Nous utiliserons dans ce TP des implémentations récursives. Aussi est-t-il prudent de placer en début de script les commandes suivantes, qui éviteront les appels infinis et ceux dépassant les limites prédéfinies en matière de pile de récursivité :

```
import sys
sys.setrecursionlimit(3200)
```

On testera chacun des différents algorithmes implémentés sur les listes U = [16, 4, 32, 8, 2, 64, 1, 128] et V = $[\sin(0), \sin(1), \ldots, \sin(500)]$.

Question 1. Définir ces deux listes. Importer la bibliothèque `copy`.

1 Tri par insertion

L'algorithme de tri par insertion a été vu en cours.

Question 2. Coder une fonction `triinsertion(L)` qui effectue le tri par insertion d'une liste L de nombres. Afin de ne pas modifier la liste en argument, faire travailler la fonction sur une liste locale `M = copy.deepcopy(L)`. On fera la même chose pour les deux autres algorithmes (tri rapide et tri fusion).

Question 3. Tester cette fonction avec [16, 4, 32, 8, 2, 64, 1, 128], avec $[\sin(0), \sin(1), \ldots, \sin(500)]$.

2 Tri rapide, ou *quicksort*

La méthode consiste à placer un élément du tableau (appelé pivot) à sa place définitive, en permutant tous les éléments de telle sorte que tous ceux qui sont inférieurs au pivot soient à sa gauche et que tous ceux qui sont supérieurs au pivot soient à sa droite.

Cette opération s'appelle le partitionnement. Pour chacun des sous-tableaux, on définit un nouveau pivot et on répète l'opération de partitionnement. Ce processus est répété **récursivement**, jusqu'à ce que l'ensemble des éléments soit trié. Nous étudierons en détail les fonctions récursives, c'est-à-dire les fonctions qui s'appellent elles-mêmes. Elles sont en effet au cœur du programme de deuxième année.

Concrètement, le principe de ce tri est le suivant. Pour trier une liste L :
- Si L est vide ou admet un seul élément elle est triée (c'est notre critère d'arrêt).
- sinon :
  - On choisit un élément $e \in$ L quelconque, que l'on retire de L
  - On construit deux sous-listes : Li formée des éléments inférieurs à e et Ls formée des éléments plus grands.
  - Le résultat est la concaténation de Li, récursivement triée, de [e] et de Ls, elle aussi récursivement triée.

Question 4. Coder une fonction `trirapide(L)` qui effectue le tri rapide d'une liste L d'entiers. On choisira pour e l'élément situé en position 0. Il pourra être utile d'utiliser la syntaxe suivante, qui permet de construire des listes par compréhension *sous condition* en Python :
`[e(t) for t in T if condition(t)]`.

On rappelle que la méthode `L.remove(e)` permet de retirer la première occurrence de l'élément `e` de la liste `L`.

Question 5. Tester cette fonction avec [16, 4, 32, 8, 2, 64, 1, 128] et avec $[\sin(0), \sin(1), \ldots, \sin(500)]$.

3 Temps d'exécution

Importer la bibliothèque time : `import time as tm`. Dans les fonctions suivantes, on utilisera la commande `tm.perf_counter()` pour accéder à l'instant de l'horloge interne exprimé en s. Attention : on ne s'intéresse pas à la date, fût-elle précise, mais au temps écoulé pendant l'exécution de la fonction.

Copier / coller les fonctions obtenues dans les deux premières questions en supprimant le deepcopy, c'est-à-dire en travaillant en place. *La copie prend en effet du temps de manière ici inutile, vu que l'on souhaite comparer les vitesses d'exécution sans être intéressé par le fait de garder la liste initiale.*

Question 6. Écrire une fonction `tpsinsertion(L)` qui renvoie le temps mis par l'algorithme de tri par insertion pour effectuer ce tri pour la liste L.

Question 7. Écrire une fonction `tpsrapide(L)` qui renvoie le temps mis par l'algorithme de tri rapide pour effectuer ce tri pour la liste L.

Question 8. Aurait-on pu écrire une seule fonction à la place des deux fonctions précédentes ?

L'objectif dans les quatre parties suivantes sera de comparer les efficacités temporelles de ces deux algorithmes de tri dans des cas aléatoires : pour cela, nous allons tout d'abord coder une fonction qui permet de générer des tableaux aléatoires d'entiers (dans un intervalle entier donnée) de longueur donnée, puis tester nos algorithmes dans différents cas, avant de récupérer ces données pour les analyser et les afficher. On parle de *complexité empirique* pour une étude statistique faite sur des tableaux aléatoires par opposition à la *complexité* tout court, produit d'une étude théorique.

4 Création de tableaux aléatoires

Donnons quelques commandes de la bibliothèque `random` :

```
import random as rand
rand.randint(a,b) # renvoie un entier aléatoire dans l'intervalle [[a,b]]
rand.random() # renvoie un flottant aléatoire entre 0 et 1
rand.uniform(a,b) # renvoie un flottant aléatoire entre min(a,b) et max(a,b).
rand.sample(L,n) # renvoie n termes pris au hasard sans remise dans une liste L ou un itérateur.
```

Question 9. Écrire une fonction `randomtab(N)` qui renvoie une liste aléatoire de N flottants compris entre 0 et 1.

5 Premiers tests

Question 10. Écrire une fonction `testrandominsertion(ntab,N)` qui renvoie une liste des `ntab` temps d'exécution de l'algorithme de tri par insertion pour `ntab` tableaux aléatoires de `N` éléments.

Question 11. Reprendre la question précédente avec le tri rapide.

Question 12. Écrire une fonction `tempsinsertion(ntab,N)` qui renvoie sous la forme d'un tuple le temps d'exécution moyen et le temps d'exécution le plus lent mis pour trier par insertion `ntab` tableaux de `N` éléments.

Question 13. Écrire une fonction `tempsrapide(ntab,N)` qui renvoie sous la forme d'un tuple le temps d'exécution moyen et le temps d'exécution le plus lent mis pour trier `ntab` tableaux de `N` éléments par le tri rapide.

Question 14. Tracer sur une même figure le temps moyen du tri rapide et du tri par insertion pour 100 tableaux à 20, 50, 100, 200, 500, 1000, 2000 et 3200 éléments. Garder les données en mémoire.

Question 15. Reprendre la question précédente (sur une autre figure) pour le temps le plus lent de ces deux tris.

6 Régression linéaire

Pour avoir une idée plus précise de ce qui se passe quantitativement dans les cas précédents, nous allons mettre en place une régression.

La régression linéaire consiste à trouver pour une liste `X=[`x_0`,...,`x_{n-1}`]` et une liste `Y=[`x_0`,...,`x_{n-1}`]` de valeurs réelles, la « meilleure » droite $y = ax + b$ approchant le nuage des points $(x_i, y_i)_{0\leq i<n}$. On convient de choisir cette meilleure droite de manière optimale au sens des moindres carrés en ceci qu'elle minimise la somme des écarts au carré entre les y_i et les $ax_i + b$.

On note $\overline{x}$ et $\overline{y}$ les moyennes respectives des listes X et Y.

La covariance de X et de Y est définie par $\mathrm{Cov}(X,Y) = \dfrac{1}{n}\displaystyle\sum_{i=0}^{n-1}(x_i - \overline{x})(y_i - \overline{y})$.

La variance de X est définie par $\mathrm{V}(X) = \mathrm{Cov}(X,X)$.

Les coefficients a et b sont donnés par les formules suivantes :

$$a = \frac{\mathrm{Cov}(X,Y)}{\mathrm{V}(X)} \text{ et } b = \overline{y} - a\overline{x}$$

Le coefficient de corrélation linéaire, compris entre -1 et 1, mesure si la régression linéaire est de bonne qualité ou non. Un coefficient proche de 1 en valeur absolue dénote un bon ajustement ; plus ce coefficient est faible en valeur absolue et moins la régression linéaire est adaptée. Il est donné par la formule :

$$\rho = \frac{\mathrm{Cov}(X,Y)}{\sqrt{\mathrm{V}(X)\,\mathrm{V}(Y)}}.$$

Question 16. Écrire une fonction `regressionlineaire(X,Y)` qui prend en argument deux listes X et Y et qui renvoie les coefficients a, b et ρ calculés par les formules ci-dessus.

7 Détermination expérimentale de la complexité

Les allures des courbes de temps de calcul en fonction du temps évoquent possiblement une complexité polynomiale. On aurait alors $T(n) = \Theta(n^\alpha)$ où n est la longueur des listes à trier et T le temps d'exécution.

Question 17. Expliquer pourquoi il suffit de faire une régression linéaire avec les listes `X = [ln(n)]` et `Y = [ln(T(n))]`. Créer ces listes dans les quatre cas étudiés précédemment.

Question 18. Dans chacun des quatre cas, effectuer la régression linéaire et conclure quant à la complexité empirique des cas étudiés.

8 Listes presque triées

On considère la fonction suivante :

```
def presquetriee(N):
      i = 0
      L = []
      for j in range(N):
            i += rand.randint(-1,9)
            L.append(i)
      return L
```

Question 19. Implémenter cette fonction et la tester avec $N = 30$. Expliquer son nom.

Question 20. Reprendre les questions 14 et 15 (graphe temps moyen et temps lent) pour des tableaux presque triés. Que constate-t-on ?

9 Tri fusion

Le tri fusion est un autre tri récursif, qui est optimal pour les listes chaînées. Il repose sur une stratégie de type "diviser pour régner". Le principe est le suivant :

- On observe que pour rassembler deux tableaux T1 et T2 déjà triés, on peut les interclasser de manière suivante : on compare les plus petits éléments de chacun d'eux, on place le plus petit des deux dans un nouveau tableau T et on poursuit cette opération jusqu'à épuisement de l'un des deux tableaux. On complète alors T en ajoutant à la fin de celui-ci les éléments du tableau non vide.
- Le tri fusion est alors défini de la manière suivante :
  - Si le tableau T a au plus un élément il est déjà trié
  - Si le tableau T a deux éléments ou plus, on partage T en deux sous-tableaux de même taille lorsque possible, on appelle récursivement la fonction sur chacun des sous-tableaux et on interclasse les copies triées.

Question 21. Écrire une fonction qui opère le fusion de deux tableaux triés par ordre croissant selon la procédure décrite ci-dessus.

Question 22. Coder une fonction `trifusion(L)` qui effectue le tri fusion d'une liste L de nombres.

Question 23. Tester cette fonction avec [16, 4, 32, 8, 2, 64, 1, 128] et avec $[\sin(0), \sin(1), \ldots, \sin(500)]$.

Question 24. Reprendre les questions 12, 14 , 15 et 18 pour le tri fusion.