TP 1 : Algorithmes de tri

1 Python : introduction

Python est un langage de programmation très populaire, notamment grâce à sa syntaxe épurée et la richesse de ses librairies (calcul scientifique, développement web). Le cours de programmation de ce semestre s'appuiera sur ce langage.

1.1 Environnement de programmation

Nous utiliserons le logiciel `idle` (un peu minimal) ou `geany` (un peu plus pratique) pour écrire du code python (l'utilisation d'un autre logiciel, si on le connaît bien, est permise également).

Au démarrage d'`idle`, une invite de commande `>>>` apparaît. Elle permet de tester du code python de façon intéractive. Par exemple, on peut effectuer des opérations arithmétiques, ou des comparaisons :

```
>>> 1+3
4
>>> 1 < 3
True
>>> 2 <= 1
False
```

Dans le cas de `geany`, il n'y a par contre pas d'invite de commande. Il faut de plus configurer dans `Edit->Preferences->Editor->Indent` et choisir `spaces` au lieu de `tabs`, pour éviter des soucis d'indentation.

Pour les deux éditeurs, pour écrire des programmes plus conséquents et les sauvegarder, on crée un fichier (menu `File->New`), avec extension `.py` par convention, sur lequel on écrira du code python, que l'on peut ensuite exécuter (`<F5>`). Penser à sauvegarder souvent (`Control-s`).

1.2 Syntaxe

La syntaxe de python est assez simple, comme nous allons le voir à travers des exemples ci-dessous. Notons cependant l'utilisation de l'indentation pour délimiter les blocs (pas d'accolades ou de mots-clés comme `begin/end`), donc il faut être soigneux sur ce point.

```
# Ceci est un commentaire (introduit par '#')

# Affichage
print("Hello, world!") # Afficher la chaîne "Hello, world!"
print(3) # Afficher l'entier 3

# Tableaux (appelés aussi "listes" en python)
t = [1, 2, 3]  # tableaux avec les trois entiers 1, 2 et 3
t[0] = 10      # mettre 10 en position 0 ==> t == [10, 2, 3]
t[0], t[1] = t[1], t[0]
               # échanger deux valeurs ==> t == [2, 10, 3]
t.append(4)    # t == [2, 10, 3, 4]
t2 = range(3)  # t2 == [0,1,2]
t3 = range(-1) # t3 == [] (tableau vide)

# Conditionnelle
# (ne pas oublier les ":" après la condition et le else, et
# remarquer l'indentation)
if x > 0:
  print("strictement positif")
else:
  print("négatif")

# boucle while (tant que)
while i < j:
  # faire quelque chose
  # (le bloc doit être indenté)

# Définir une fonction
def fib(n):
  a, b = 1, 1
  for i in range(n - 2):
    a, b = b, a + b
  return b
```

```
# Appel de fonction
x = fib(5) # x == 5

# importer le module "random"
import random
# entier aléatoire entre 1 et 10 (inclus)
n = random.randint(1,10)

# "Lists comprehensions". Exemple:
# liste des carrés des nombres de 0 à 9
t = [x ** 2 for x in range(10)]
```

2 Tris

Dans la suite, l'énoncé demande de programmer plusieurs algorithmes de tri sur des tableaux : une procédure de tri appliquée à un tableau de nombres `t`, doit modifier le tableau de façon à conserver ses éléments (et leurs nombres d'occurrence) et les classer par ordre croissants.

$$\texttt{t}[0] \leq \texttt{t}[1] \ldots \leq \texttt{t}[\texttt{len}(\texttt{t}) - 1]$$

Les deux premiers algorithmes que nous étudieront sont dits *en place* car ils n'utilisent pas de tableaux auxiliaires dans leurs calculs, limitant ainsi l'usage mémoire.

2.1 Tri insertion

Écrire une fonction `insert_sort(t)`, qui trie en place un tableau en utilisant l'algorithme du tri insertion. Cet algorithme procède ainsi : pour $k = 1, \ldots, len(t) - 1$, on insère le k-ième élément du tableau à sa place dans les k premières cases (les $k - 1$ premières étant déjà triées).

2.2 Tri rapide

Écrire une fonction `quick_sort(t)`, qui trie en place un tableau en utilisant l'algorithme du tri rapide. L'algorithme procède ainsi :

- Si le tableau a plus d'un élément, choisir un pivot (un élément du tableau, choisi au hasard de préférence).
- Mettre tous les éléments plus petits que le pivot à gauche de celui-ci, et ceux plus grands à droite de celui-ci (indication : utiliser deux indices i, j, l'un croissant, l'autre décroissant).
- Faire un appel récursif sur les deux sous-tableaux ainsi créés.

On pourra utiliser la fonction `random.randint(i,j)`, qui renvoie un entier aléatoire entre `i` et `j` (inclus), pour choisir le pivot.

2.3 Tri fusion (bonus)

Écrire une fonction `merge_sort(t)` qui trie un tableau (pas en place) en utilisant l'algorithme du tri fusion. Cet algorithme procède de la façon suivante :

- Si le tableau a plus d'un élément strictement, le partager en deux sous-tableaux de taille égale (à ± 1 près).
- Appliquer récursivement l'algorithme aux deux sous-tableaux.
- Fusionner les deux sous-tableaux triés (indication : utiliser deux indices croissants dans chaque sous-tableau).

Remarque. Pour le faire efficacement, il faut allouer préalablement un tableau supplémentaire de même taille, sur lequel on peut effectuer des copies auxiliaires.

3 Affichage

Instrumenter les fonctions des deux premiers tris, de sorte à renvoyer la liste des transpositions successives effectuées lors du tri, afin de pouvoir afficher progressivement les étapes des algos de tris.

Télécharger le fichier `AnimateSort.py` à l'adresse `http://perso.eleves.ens-rennes.fr/~yfern356/AnimateSort.py`, et le mettre dans le même dossier que le fichier du TP.

```
# Import du module d'affichage :
import AnimateSort

# Pour l'utiliser :
# t : tableau à trier
# l : liste des transpositions à effectuer
# (ex: [(1,2),(2,3)] représente l'échange de 1 et 2,
# suivi de l'échange de 2 et 3)
AnimateSort.Animation(t,[l]).go()
# Comparer visuellement avec la liste d'origine:
AnimateSort.Animation(t,[[],l]).go()
```