TP n°10 - Étude expérimentale de complexité Entrées/Sorties.

Tous les codes de ce TP devront être remis sur Moodle - TP10 - avant jeudi 08/12 23h59

Pour ce TP, sur les machines du lycée, il faudra travailler dans un répertoire `TP10` situé en dehors du dossier `Documents`.

Exercice 1 (Question rapidité - OCaml - 10 minutes max).

On réfléchira d'abord en pseudo-code sur papier.

Écrire en OCaml une fonction `merge : 'a list -> 'a list -> 'a list` qui reçoit deux listes **triées** et qui renvoie une liste triée formée des éléments des deux listes.

Par exemple :

- `merge [1;3;5] [2;4;6;8;10]` renvoie `[1;2;3;4;5;6;8;10]`
- `merge ['b';'b';'e';'k';'p'] ['a';'a';'g','z']` renvoie `['a';'a';'b';'b';'e';'g';'k';'p']`

On ne demande pas dans un premier temps l'implémentation d'une récursivité terminale.

Pour les plus rapides :

- tester votre fonction sur des tableaux d'entiers pseudo-aléatoires triés (cf TPs précédents).
- rendre cette fonction récursive terminale. On pourra utiliser la fonction `rev` du module `List` (équivalent de la fonction `miroir` que nous avions codée)

Cette fonction nous resservira lorsque nous verrons l'algorithme de tri fusion au second semestre.

Sous Linux, dans le terminal, il est possible de mesurer le temps d'exécution d'une commande en la faisant précéder de la commande `time`. Par exemple, la commande

```
time ./mon_prog 8
```

permet d'obtenir le temps d'exécution du programme `mon_prog` qui a été lancé avec 8 comme unique entrée.

Exercice 2 (Analyse du temps d'exécution du tri par insertion).

1. Copier le code C de l'algorithme de tri par insertion **iteratif** du TP n°9 dans votre dossier de travail. Nous allons faire fonctionner ce code sur de très grands tableaux, je vous invite donc à commenter les affichages de tableaux.
2. Exécuter l'algorithme de tri par insertion en mesurant le temps d'exécution sur des tableaux d'entiers aléatoires de tailles n, avec les valeurs de n (imposées) suivantes : 1000, 10000, 25000, 50000, 75000, 100000, 125000, 150000. Dans un fichier texte `mesures.txt`, sauvegarder les informations relevées pour chaque exécution : la taille de tableau donnée en entrée et le temps d'exécution **réel**.
3. Écrire un script Python `trace_complexite.py` qui trace la courbe reliant le temps de calcul obtenu en fonction de la taille n du tableau à trier.
4. Observer la courbe obtenue. Quelle conjecture pouvez-vous faire sur la complexité temporelle de l'algorithme ?
5. La méthodologie utilisée vous parait-elle satisfaisante ? Quels aspects de cette petite étude peuvent être améliorés ?

Un prochain travail permettra de définir une méthodologie plus rigoureuse et automatisée pour cette étude de complexité temporelle expérimentale.

Exercice 3 (Analyse d'un fichier en langage C).
Les fonctions suivantes seront codées en C dans un fichier `analyse_fichier_texte.c`
On fait l'hypothèse que tous les fichiers textes manipulés sont au format ASCII.

1. Écrire une fonction `compte_lignes` qui compte le nombre de lignes d'un fichier. Le chemin du fichier à analyser (chemin absolu ou relatif) sera donné sur la ligne de commande par l'utilisateur *Indication : on atteint la fin d'une ligne quand on lit un caractère de retour à la ligne.*
2. Tester votre code sur différents fichiers. Vous pouvez créer vous-mêmes quelques fichiers de tests assez courts sous `emacs` pour commencer. Une base de fichiers de tests est à disposition sur Moodle.
3. Réfléchir sur papier à une fonction `compte_mots` qui prend en entrée le nom d'un fichier texte, et renvoie en sortie le nombre de mots du fichier. *Indication : On considère que les mots sont séparés par un* ***ou plusieurs*** *espaces et/ou retours à la ligne. Les caractères de ponctuation collés aux mots sont comptés comme faisant partie d'un mot.*
4. Coder cette fonction dans votre fichier `analyse_fichier_texte.c`
5. Tester comme pour la première fonction, avec de petits fichiers que vous vous créerez puis de plus gros fichiers, par exemple ceux fournis sur Moodle. Vous pouvez comparer vos résultats avec ceux de la commande Linux `wc` (*word count*) dont vous pourrez aller lire le manuel [a].

a. Pour obtenir le manuel : `man wc`