**Activité : Complexité en temps d'algorithmes de tri**

**Objectif :** A l'aide de la procédure "mesure\_temps\_tri", mesurer le temps d'exécution de différents algorithmes de tri, et l'ordre de grandeurs du nombre d'opérations unitaires effectué par ces algorithme afin d'en déterminer expérimentalement la complexité en temps.

**I. Algorithmes de tri à tester**

Les algorithmes dont on cherche à estimer la complexité en temps sont ceux établis précédemment au sein des fonctions "tri\_selection", "tri\_bulle", "tri\_rapide" et "tri\_sort".

- "tri\_selection" effectue un tri croissant d'une liste d'entiers en utilisant un algorithme de tri par sélection;

- "tri\_bulle" effectue un tri croissant d'une liste d'entiers en utilisant un algorithme de tri par sélection;

- "tri\_rapide" effectue un tri croissant d'une liste d'entiers en utilisant un algorithme de tri par pivot non récursif;

- "tri\_sort" effectue un tri croissant d'une liste d'entiers en utilisant l'algorithme de tri natif de python.

**II. Mesure du temps d'exécution de chaque fonction de tri**

Les fonctions "mesure-temps" et "mesure\_complexite" font partie du module "complexite\_temps.py" : il faut donc importer ce module au sein de votre module comportant les fonctions de tri. Ces deux fonctions nécessitent l'utilisation de listes, d'entiers triés ou non triés, enregistrées dans des fichiers de type csv : Le professeur peut fournir ces fichiers, ou ils peuvent être générés à l'aide des fonctions "generateurListeCsv" ou "genereToutesLesListes" contenues aussi dans le module "complexite\_temps.py".

**II.1.** A partir de l'aide de la fonction "mesure\_temps", des listes d'entiers non triés sous forme de fichiers de type csv, et des différentes fonctions de tri à tester, proposer un programme en python permettant de mesurer le temps d'exécution des quatre algorithmes de tri en fonction du nombre n d'entiers à trier.

**II.2.** Relever dans le code de la fonction "mesure\_temps" les lignes de code servant à mesurer le temps d'exécution de la fonction passée en paramètre.

**II.3.** Réaliser et exécuter ce programme pour mesurer le temps d'exécution des quatre algorithmes de tri en fonction du nombre n d'entiers à trier.

**II.4.** Rassembler ces données sur un même fichier de tableur, et tracer pour chaque fonction sur le même graphe t=f(n), avec t le temps d'exécution de la fonction et n le nombre d'entiers dans la liste à trier.

Adopter des échelles permettant d'obtenir une bonne lisibilité des graphes.

**II.5.** Conclure en comparant le temps d'exécution de chaque algorithme et observer l'influence de la taille des listes à trier sur ce temps.

**III. Estimation de la complexité en temps de chaque algorithme de tri**

Pour estimer la complexité des algorithmes de tri, on va compter le nombre de tours de boucle effectué dans chaque algorithme pour traiter les listes.

Ceci nous donnera un ordre de grandeur du nombre d'opérations unitaires effectuées, et donc nous permettra de déterminer expérimentalement la classe de complexité de l'algorithme étudié.

**III.1.** A partir de l'aide de la fonction "mesure\_complexite", des listes d'entiers non triés sous forme de fichiers de type csv, et des différentes fonctions de tri à tester, réaliser le programme en python permettant de compter le nombre de tours de boucle effectué par les trois algorithmes de tri (sélection, bulle et rapide) en fonction du nombre n d'entiers à trier.

**III.2.** Rassembler ces données sur un même fichier de tableur, et tracer pour chaque fonction sur le même graphe ntb=f(n), avec ntb le nombre de tours de boucle effectué et n le nombre d'entiers dans la liste à trier.

Adopter des échelles permettant d'obtenir une bonne lisibilité des graphes.

**III.3.** Ajouter sur le précédent graphe les courbes suivantes :

- ntb = n.log(n) (avec log le logarithme en base 2)

- ntb = n

- ntb = n²

Ces trois courbes donnent l'allure des classes de complexité en temps O( n.log(n) ), O( n ) et O( n² ).

**III.4.** Par comparaison des courbes empiriques (question précédente III.3.) avec les courbes correspondantes aux algorithmes testés (question III.2.), en déduire la classe de complexité en temps des algorithmes de tri par sélection, bulle et rapide.

Aide : Si deux des courbes ntb=f(n) ont la même allure, on peut alors dire qu'elles correspondent à la même classe de complexité.

**III.5.** Conclure quant à la classe de complexité observée expérimentalement, et empirique de chaque algorithme de tri étudié.

**IV. Estimation de la complexité en temps de l'algorithme de tri natif de python (fonction sort())**

N'ayant pas accès au code de la fonction sort() de python, on ne peut pas compter les tours de boucle qu'elle effectue. En revanche, il est possible de l'estimer.

Proposer, et réaliser, un protocole expérimental permettant de tracer comme précédemment la courbe ntb=f(n) pour la fonction "tri\_sort" encapsulant la fonction native sort() de python. (aide : on peut se baser sur la mesure du temps d'exécution)

En déduire la classe de complexité de la fonction native sort() de python.

**V. Etude des pires cas pour chaque fonction de tri**

A l'aide des fonctions "mesure\_temps" ou "mesure\_complexite", des listes d'entiers triés (sous ordre croissant et décroissant) sous forme de fichiers de type csv, et des différentes fonctions de tri à tester, proposer un programme en python permettant de mesurer le temps d'exécution des quatre algorithmes de tri en fonction du nombre n d'entiers à trier.

Réaliser et exécuter ce programme pour mesurer le temps d'exécution des quatre algorithmes de tri en fonction du nombre n d'entiers à trier pour tous les cas.

En déduire quel est le pire cas pour chaque algorithme de tri.