DM 1 : Étude de quelques algorithmes de tri

CPES « Sciences des données, arts et cultures » : Introduction à l'algorithmique

Antoine GAUQUIER — antoine.gauquier@ens.fr
Pierre SENELLART — pierre@senellart.com

Mis en ligne le 03 octobre 2023, à rendre avant le **22 octobre 2023 23h59**

Les algorithmes de tri sont des algorithmes qui permettent d'organiser une collection d'éléments selon une *relation d'ordre* bien définie. Ces éléments peuvent prendre la forme de n'importe quels objets, à partir du moment où l'on définit une façon d'ordonner ces éléments. On pense naturellement aux nombres que l'on peut ranger par ordre croissant ou décroissant, mais on peut également penser à l'ordre *lexicographique* (c.-à-d., l'ordre du dictionnaire) sur les chaînes de caractères par exemple.

Il existe un grand nombre de façons différentes de réaliser de tels tris, via des algorithmes plus ou moins efficaces. Le but de ce premier devoir maison est d'étudier quelques-uns de ces algorithmes, et de les comparer, tant sur le plan théorique qu'empirique. Cette étude vous servira par ailleurs pour la suite du cours, où une séance sera dédiée à l'étude des algorithmes de tri plus efficaces, mais aussi plus complexes.

Les trois algorithmes que vous allez étudier sont les suivants : le *tri à bulles*, le *tri par insertion* et le *tri par sélection*.

Informations pratiques

Vous avez jusqu'au **22 octobre 2023 23h59** pour rendre ce devoir maison. Le rendu doit se faire sur le Moodle[1] du cours, sous la forme d'une archive au format `.zip`. Tout travail rendu après cette date se verra pénalisé par un malus au pro-rata du retard. Dans le cas où vous ne parveniez pas à accéder à Moodle, vous pouvez rendre votre travail par mail, en suivant la même date limite de rendu (la date et l'heure de l'envoi du mail faisant foi). Cette archive devra contenir :

— Un fichier au format PDF, contenant les réponses à toutes les questions qui ne nécessitent pas de rédaction de code : calculs de complexités algorithmiques, notations asymptotiques, justifications de choix… La forme du PDF est laissée libre. Vous pouvez, de façon non exhaustive : utiliser un logiciel de traitement de texte (*OpenOffice Writer*, *Word*…), utiliser le système de composition de documents LaTeX ou encore directement scanner des pages que vous auriez rédigées à la main. **Nous attirons toutefois votre attention sur le fait que le PDF fourni doit être lisible, et vous devez être en mesure d'utiliser**

1. `https://moodle.monlycee.net/course/view.php?id=8303`

les symboles mathématiques nécéssaires à une rédaction correcte. Prenez donc (notamment) en compte le fait que l'intégration d'équations ou symboles mathématique est fastidieuse avec Word, ou que les scans de documents doivent être de bonne qualité. La solution optimale reste l'utilisation de LaTeX, mais sa prise en main est assez longue.

— **Trois** fichiers contenant du code Python (un pour chaque algorithme de tri), **organisés en fonctions**. Lorsque le sujet indique un nom de fonction spécifique, il vous est demandé de **respecter ces conventions de nommage**, afin de faciliter la correction. Il est recommandé de rendre un *Notebook* Python (un fichier en `.ipynb` si vous utiliser *Jupyter Notebook*), en particulier pour répondre aux questions qui demandent de faire appel à de l'affichage, même si ce n'est pas obligatoire. Dans ce sens, il ne vous est pas imposé d'utiliser la fonction `__main__`.

1 Écriture du pseudo-code des algorithmes

Question 1. Vous trouverez ci-dessous la description en langue naturelle des trois algorithmes de tri mentionnés plus tôt en introduction. À partir de celles-ci, proposez trois pseudo-codes correspondants. Voici les descriptions en question :

— **Tri par insertion.** Le tri par insertion est l'algorithme de tri le plus « naturel ». Il est généralement employé lorsque l'on souhaite trier sa main dans un jeu de cartes. Son principe peut être décrit à partir de cet exemple : dans notre main droite, on positionne les cartes non-triées en éventail. Au départ, la main gauche ne contient aucune carte. Lors de la première itération, on vient prendre la carte la plus à gauche de notre main droite, et on la place dans la main gauche, vide. Lors de la deuxième itération, on prend la nouvelle carte la plus à gauche dans notre main droite, puis on vient la placer dans la main gauche, en s'assurant de la ranger de telle sorte à ce que la main gauche reste toujours triée par ordre croissant. On répète ainsi l'opération jusqu'à prendre la dernière carte de la main droite pour la placer dans la main gauche. La main gauche contiendra alors toutes les cartes triées dans l'ordre croissant.

— **Tri par sélection.** Le tri par sélection peut se décrire au travers d'un autre exemple. Prenons le cas d'un groupe d'individus en file indienne, placés aléatoirement, qui doit se ranger du plus petit au plus grand. Le principe de l'algorithme est le suivant : à la première itération, tous les individus de la file donnent leur taille. Une fois que c'est fait, l'individu dont la taille est la plus petite lève la main, et il échange sa place avec l'individu en tête de la file. Puis vient la seconde itération, où tous les individus, excepté celui en tête de file qui a déjà été trié, donnent à nouveau leur taille. Parmi ceux-ci, le nouveau plus petit lève sa main, et vient échanger sa place avec l'individu qui se trouve derrière le dernier individu trié, en l'occurrence ici derrière l'individu en tête de file. On continue ainsi jusqu'à ce qu'il n'y ait plus que deux individus non triés, qui donnent chacun leur taille, et échangent leur place si le plus petit individu occupe la dernière place de la file.

— **Tri à bulles.** Le tri à bulles, aussi appelé tri par propagation, tient son nom du fait qu'il fait progressivement remonter les plus grands éléments en haut de la liste, comme une bulle remonte à la surface de l'eau. Son principe est simple : on va faire des passes de gauche à droite dans la liste, et comparer les éléments deux à deux (de proche en proche),

en les permutant si ils ne sont pas dans le bon ordre. Ainsi, lors de la première passe, on aura fait remonter le plus grand élément dans la dernière position de la liste, à sa place définitive. À la deuxième itération, on vient répéter la même opération, sauf que l'on ne considère plus que les éléments qui ne sont pas à leur place définitive, à savoir tous les éléments de la liste excepté celui à la dernière position. On continue ainsi jusqu'à ce qu'il ne reste plus que deux éléments qui ne sont pas triés (qui sont aussi les deux éléments les plus petits), et on les intervertit si ils ne sont pas à la bonne place.

2 Analyse théorique des algorithmes

Question 2. En partant des pseudo-codes que vous avez écrits dans la question précédente, déterminez, pour chacun d'entre eux, les fonctions $f_p(n)$ et $f_m(n)$, correspondant respectivement au nombre d'opérations élémentaires dans le pire cas et dans le cas moyen. On ne cherche pas ici une notation en O, Ω ou Θ mais un comptage des opérations élémentaires identifiées dans chaque pseudo-code. Justifiez votre réponse.

Question 3. Déterminez ensuite, à partir de ces fonctions, les notations asymptotiques associées en O, Ω et Θ. Pour y parvenir, nous vous demandons de repartir de la définition mathématique de ces notations, et de donner des valeurs de paramètres pour lesquelles ces notations sont correctes.

Question 4. On considère avoir trouvé un pseudo-code au nom inconnu, proposant un algorithme de tri qui, après analyse, a les complexités temporelles asymptotiques suivantes : $O\left(n^2\right)$ dans le pire cas, et $O\left(n \times \log(n)\right)$ dans le cas moyen. Dans chacune des deux situations, cet algorithme est-il meilleur que les trois algorithmes étudiés ? Pire ? Justifiez votre réponse.

Question 5. On dit qu'un algorithme de tri est *en place* quand le tri peut s'effectuer sans recopie de l'ensemble de la liste, mais en manipulant directement ces éléments au sein de la liste. D'après vous, le ou lesquels de ces algorithmes peuvent être qualifiés de *tri en place* ? Justifiez votre réponse.

3 Implémentation des algorithmes en Python

Question 6. Implémentez les pseudo-codes écrits dans le premier exercice en Python. Les trois fonctions doivent porter les noms `tri_insertion`, `tri_selection` et `tri_bulles`, et prendre un **unique argument**, qui est une liste Python de nombres (pouvant être entiers, à virgule flottante, positifs, négatifs), non-triée. Chaque fonction doit **retourner une liste Python**, contenant les éléments de la liste d'entrée, cette fois-ci triée.

Question 7. Écrivez une fonction `afficher_liste` prenant en argument une liste Python de nombres (non-triée), ainsi qu'une chaîne de caractère, représentant le nom de la méthode de tri qui sera utilisée. Cette fonction ne doit rien retourner, mais doit afficher la liste avant son tri, puis appliquer à cette liste l'algorithme de tri sélectionné dont le nom a été passé en paramètre, puis enfin afficher la liste triée par cet algorithme. Essayez ensuite cette fonction pour différentes listes de nombres, et ce en utilisant au moins une fois chaque algorithme.

Question 8. Dans cette dernière question, l'objectif est de pouvoir visualier les complexités temporelles des algorithmes, en utilisant une méthode de *profiling*. Pour ce faire, on pourra utiliser la même bibliothèque que dans le TP3, à savoir `time`. L'objectif est d'afficher plusieurs courbes, représentant les temps d'exécution des différents algorithmes implémentés. Plus précisément, vous devez afficher :

— Une courbe représentant l'évolution du temps **mesuré** de l'exécution de la fonction `tri_insertion`, en fonction de la taille de la liste d'entrée n.
— Une courbe représentant la complexité asymptotique théorique dans le pire cas (notation O) pour l'algorithme du tri par insertion, en fonction de la taille d'entrée n.
— Une courbe représentant l'évolution du temps **mesuré** de l'exécution de la fonction `tri_selection`, en fonction de la taille de la liste d'entrée n.
— Une courbe représentant la complexité asymptotique théorique dans le pire cas (notation O) pour l'algorithme du tri par sélection, en fonction de la taille d'entrée n.
— Une courbe représentant l'évolution du temps **mesuré** de l'exécution de la fonction `tri_bulles`, en fonction de la taille de la liste d'entrée n.
— Une courbe représentant la complexité asymptotique théorique dans le pire cas (notation O) pour l'algorithme du tri à bulles, en fonction de la taille d'entrée n.

On réfléchira au choix de données d'entrée appropriées pour tracer ces courbes.

Quelques indications pour vous aider : il est fortement conseillé d'utiliser un *Notebook* Python pour réaliser cette question. Il vous est aussi recommandé d'utiliser la bibliothèque `matplotlib`[2] qui permet de représenter graphiquement des courbes. En particulier, vous pouvez utiliser `matplotlib.pyplot.plot(x, y, label)` qui, à partir de deux listes x et y passées en paramètres, affichera sur une grille 2D chaque couple (x_k, y_k), simulant ainsi l'affichage d'une courbe. Le paramètre `label` permet de donner une étiquette à la courbe. Cela facilitera son identification lorsque vous afficherez plusieurs courbes simultanément, ce que vous pouvez faire en appelant la fonction autant de fois que vous avez de courbes à afficher. Vous pouvez utiliser ce petit tutoriel[3] afin d'avoir un exemple de code permettant de réaliser un affichage avec `matplotlib`, utilisant la fonction présentée ci-dessus. Vous pouvez également vous inspirer de la correction du TP3, qui fait appel à cette même fonction, sur un problème différent. Il est recommandé de partir d'un de ces deux scripts, puis de le modifier afin de réaliser l'affichage demandé dans la question.

Une fois l'affichage réalisé, commentez les résultats observés.

2. `https://matplotlib.org/`
3. `https://matplotlib.org/stable/gallery/lines_bars_and_markers/simple_plot.html#sphx-glr-gallery-lines-bars-and-markers-simple-plot-py`