TD6. Algorithmes de tri

Bases de la Programmation Impérative – Chapitre 2 : iterations
version du 25 décembre 2023

Les objectifs de ce TD sont les suivants :

- (re) voir deux algorithmes classiques de tri ;
- s'interroger sur le coût de ces algorithmes en fonction des entrées ;

Exercice 1 : tri par insertion

On se propose d'implémenter un tri par insertion. Cet algorithme fonctionne en triant le tableau à partir de son début. Au départ rien n'est trié et à la fin tout est trié. Normal, c'est l'objectif et si ce n'était pas le cas, l'algorithme serait incorrect.

Quel est l'état intermédiaire du système ? En milieu de parcours tout le début du tableau est composé d'éléments ordonnés comme le montre la figure ci dessous. On parle **d'invariant** de boucle, c'est à dire d'une propriété vraie au début de chaque itération de la boucle.

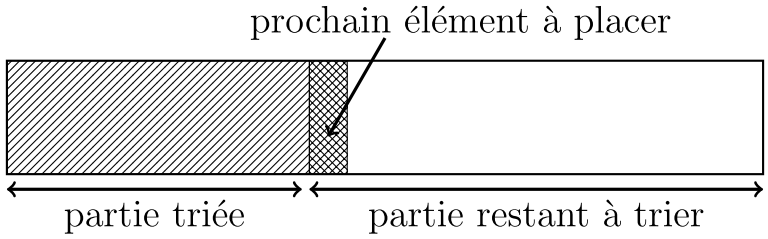

Fig. 1 : illustration tri par insertion

On prend le premier élément non ordonné et on l'insère à la bonne place dans la partie ordonnée du tableau. Cette insertion augmente la taille de la partie ordonnée de 1 et il suffit donc de la répéter pour obtenir un tableau complètement trié.

Question 1

Programmer un tri par insertion.

Question 2

Quelle est à la complexité en temps de l'algorithme ?

Exercice 2 : tri par sélection

Un second algorithme simple de tri est le tri par sélection. Comme dans le tri par insertion, le tableau est toujours divisé entre une partie triée et une partie non triée.

Comme l'illustre la figure ci-dessous, au lieu de prendre le premier élément non trié et de l'insérer au bon endroit on sélectionne parmi les éléments non triés celui qui prendra sa place définitive à la fin de la partie triée. Les éléments de la partie triée ne sont donc jamais déplacés.

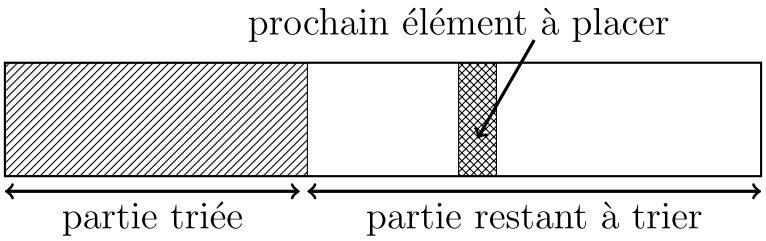

Fig. 2 : illustration tri par sélection

Question 1

Programmer un tri par sélection.

Question 2

Quelle est à la complexité en temps de l'algorithme ?

Exercice 3 : fusion de tableaux triés

Question 1

Écrire un algorithme qui fusionne deux tableaux triés en un seul également trié.

Question 2

Quelle est la complexité en temps de l'algorithme ?

Exercice 4 : un peu de stabilité (pour aller plus loin)

Un algorithme de tri est **stable** si deux éléments ayant la même valeur dans le tableau se retrouvent dans le même ordre dans le tableau après le tri. Il faut donc avoir une notion d'ordre en plus de celle définie pour trier le tableau pour parler de stabilité. En général dans le contexte des tableaux, la position initiale des éléments, c'est-à-dire les indices du tableau auxquels ils se trouvent, est utilisée pour définir cet ordre supplémentaire.

Question 1

L'algorithme de tri par insertion est-il stable ? Pourquoi ?

Question 2

L'algorithme de tri par sélection est-il stable ? Pourquoi ?

Exercice 5 : peut-on faire mieux ? (pour aller encore plus loin)

Question 1

Quel est l'algorithme de tri utilisé par la méthode `list.sort()` et la fonction `sorted()` de l'implémentation standard Python ? Comparez son coût aux algorithmes étudiés dans ce TD.