Travaux pratiques n°6

Insertions et tris

F. Cuvellier, J. Tanoh

M.P.S.I. 1, 2016–2017

1 Recherche et insertions

Exercice 1. On se donne un nombre a et un tableau L de n nombres rangés dans l'ordre croissant, dont certains peuvent être égaux. On voudrait insérer a dans le tableau L de sorte que le nouveau tableau soit encore ordonné.

1. Proposer un algorithme de recherche de l'endroit où insérer a dans le tableau L suivant l'ordre croissant. La complexité en temps sera de l'ordre de n dans le cas le pire. Estimer la complexité en mémoire.
2. Proposer maintenant un algorithme dont la complexité en temps soit de l'ordre de $\lg n$ (logarithme de base 2). Estimer la complexité en mémoire.
3. Implémenter ces algorithmes en Python et les comparer sur des tableaux de mille nombres. Pour l'insertion, on se servira de la méthode `insert()` applicable à un tableau.

Exercice 2. On se donne deux tableaux de nombres rangés dans l'ordre croissant, l'une de m nombres, l'autre de n nombres.

1. Écrire un algorithme pour former un tableau constitué des $m + n$ nombres de ces deux tableaux, rangés dans l'ordre croissant. Sa complexité en temps devra être de l'ordre de $m + n$.
2. L'implémenter en Python.

2 Tris

On se propose d'écrire des algorithmes de tri d'un tableau L de n nombres, dont certains peuvent être égaux.

Exercice 3. On considère d'abord l'algorithme naïf, qui consiste à rechercher le plus petit élément, à le ranger en premier, puis le deuxième élément le plus petit, à le ranger en deuxième position, etc.

1. Écrire cet algorithme. Estimer sa complexité en temps et sa complexité en mémoire.

2. L'implémenter en Python.

Exercice 4. Le tri par insertion d'un tableau L, formé des éléments $L[0]$, $L[1]$, ..., $L[n-1]$, consiste à rechercher où insérer un élément donné puis à pousser les suivants pour l'insérer. Par exemple, dans le cas où

$$L[0] \leqslant L[1] \leqslant \ldots \leqslant L[k-1],$$

où k désigne un indice tel que $1 \leqslant k \leqslant n-1$, pour insérer $L[k]$, on l'avance dans le tableau en faisant reculer successivement $L[k-1]$, $L[k-2]$, etc., de sorte que

$$L[0] \leqslant L[1] \leqslant L[j-1] \leqslant L[k] \leqslant L[h] \leqslant \ldots \leqslant L[k-1].$$

1. Écrire l'algorithme de tri par insertion d'un tableau L de longueur n. (Sa complexité en temps est de l'ordre de n^2 et sa complexité en mémoire de l'ordre de n.)

2. L'implémenter en Python.

Exercice 5. On se donne un tableau L de n nombres. On le trie suivant le procédé suivant. Dans un premier temps, on prend ses éléments l'un après l'autre, en cherchant à les ranger suivant l'ordre croissant dans des tableaux de nombres, que l'on crée si nécessaire. Si l'élément considéré est plus grand que le plus grand élément d'un des nouveaux tableaux, on le met au bout du premier d'entre eux. Sinon, on crée un nouveau tableau dont il est l'unique élément (à cet instant). Dans un second temps, lorsqu'on a épuisé le tableau L, on fusionne les tableaux précédents en un seul.

1. Écrire l'algorithme correspondant. Estimer sa complexité en temps et sa complexité en mémoire.

2. L'implémenter en Python.

Exercice 6. On part d'un tableau L de nombres dont la longueur est une puissance de 2. S'il est de longueur 1, il est trié. Sinon, on le partage en deux sous-tableaux de longueur moitié, que l'on trie récursivement, puis on les fusionne.

1. Écrire l'algorithme correspondant. (On ne cherchera à estimer ni sa complexité en temps, ni sa complexité en mémoire.)

2. L'implémenter en Python.

Exercice 7. Essayer les fonctions écrites en Python sur des tableaux de mille vingt-quatre nombres formées de façon aléatoire ($2^{10} = 1024$).