Travaux pratiques n°7

Insertions et tris

F. Cuvellier, J. Tanoh

M.P.S.I. 1, 2014–2015

1 Insertions

Exercice 1.

1. On se donne un nombre a et une liste L de n nombres rangés dans l'ordre croissant, dont certains peuvent être égaux. Proposer un algorithme d'insertion de a dans la liste 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 listes de mille nombres.

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

1. Écrire un algorithme pour former une liste constituée des $m+n$ nombres de ces deux listes, 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'une liste 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.

1. Écrire l'algorithme de tri par insertion d'une liste 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 une liste L de n nombres. On la 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 listes de nombres, que l'on crée si nécessaire. Si l'élément considéré est plus grand que le plus grand élément d'une des nouvelles listes, on le met au bout de la première d'entre elles. Sinon, on crée une nouvelle liste dont il est l'unique élément (à cet instant). Dans un second temps, lorsqu'on a épuisé la liste L, on fusionne les listes précédentes en une seule.

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'une liste L de nombres dont la longueur est une puissance de 2. Si elle est de longueur 1, elle est triée. Sinon, on la partage en deux sous-listes 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 listes de mille vingt-quatre nombres formées de façon aléatoire ($2^{10} = 1024$).