Tri par insertion

1. Introduction aux tris

Nous allons étudier plusieurs algorithmes pour trier des données structurées. Il pourra s'agir de nombres, de chaînes de caractères (l'ordre lexicographique sera alors utilisé), voire de plusieurs critères subordonnés (note, puis nom puis prénom puis date de naissance). Par convention, on convient que les tableaux sont triés *par ordre croissant*. Évidemment, ce choix est sans influence sur les algorithme ni sur leurs performances ; il suffit d'inverser la relation d'ordre. Les algorithmes se distingueront d'une part par l'espace qu'ils occuperont en mémoire, d'autre part par le nombre de comparaisons entre éléments qu'ils utiliseront. Cette opération, qui peut être coûteuse en temps selon la nature des données, sera celle utilisée pour évaluer la complexité en temps de ces algorithmes.

2. Principe du tri par insertion

L'algorithme de tri par insertion parcourt le tableau à trier du début à la fin. Au moment où on considère le i-ème élément, les éléments qui le précèdent sont déjà triés.

L'objectif d'une étape est d'insérer le i-ème élément à sa place parmi ceux qui précèdent. Il faut pour cela trouver où l'élément doit être inséré en le comparant aux autres, puis décaler les éléments afin de pouvoir effectuer l'insertion.

En pratique, ces deux actions sont effectuées en une passe, qui consiste à faire remonter l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

Plus formellement, on parcourt le tableau depuis la position i-1 jusqu'à rencontrer un élément plus petit que x, élément à l'origine en position i. Si x est plus petit que l'élément rencontré, on décale ce dernier d'une position et on continue la boucle.

Exercice 1. Utiliser cet algorithme pour trier à la main, pas à pas, le tableau T=[6, 3, 8, 4, 7]. Combien a-ton réalisé de comparaisons ?

3. Pseudo-code

Exercice 2. Écrire en pseudo-code l'algorithme du tri par insertion.

4. Terminaison et correction de l'algorithme

4.1. Terminaison de l'algorithme.

Exercice 3. Prouvez que cet algorithme se termine en un temps fini.

4.2. Correction de l'algorithme.

Exercice 4. 1. Montrer qu'un invariant de la boucle `for` est « P(i)= [T[0],...,T[i-1]] est triée »
2. Montrer la correction de l'algorithme.

5. Calculs de complexité

5.1. **Complexité en mémoire.** Le tri par insertion est dit **en place** : ceci signifie que la fonction travaille directement sur le tableau T et ne demande donc aucune occupation supplémentaire de place.

5.2. **Complexité en temps.**

Exercice 5. On a convenu de ne prendre en compte, dans les calculs de complexité, que les comparaisons. Les tableaux étant considérés comme des listes chaïnées, quelles autres opérations sont-elles effectuées par l'algorithme ?

5.3. **Le pire des cas.**

Exercice 6. En dénombrant le nombre de comparaisons entre éléments du tableau, donner la complexité dans le pire des cas (on fera un calcul exact).

5.4. **Complexité dans le meilleur des cas.** La complexité du tri par insertion peut être nettement meilleure que celle étudiée plus haut. Plus intéressant, lorsque le tableau est « presque trié », le tri par insertion a une complexité en temps qui est linéaire.

Exercice 7. 1. Quelle est la complexité dans le meilleur des cas ?
2. On dit que le tableau T de longueur n est presque trié si $T_i < T_{i+1}$ pour tout $0 \leqslant i \leqslant n-1$ sauf pour un nombre borné p de valeurs de i. Montrer alors que le tri par insertion a une complexité en $\mathcal{O}(n)$.