Algorithmes de tri

Tri par sélection

Imaginons un tableau de n nombres dans un ordre quelconque. Comment faire pour le trier de façon systématique ? Attention, on ne veut pas créer un nouveau tableau, cela pourrait prendre trop de place en mémoire, il faut juste modifier le tableau existant. On appelle cela un **tri sur place**.

Une approche toute simple est la suivante :

- On commence par parcourir tous les éléments pour trouver le plus petit, puis on l'échange avec celui qui était position 0.
- On a bien mis le plus petit au début du tableau, il faut maintenant trier le reste. On parcourt tous les éléments à partir de la position 1 pour trouver le plus petit, puis on l'échange avec celui qui était en position 1.
- On a bien mis les 2 plus petits au début du tableau, en ordre croissant, il faut maintenant trier le reste. On parcourt tous les éléments à partir de la position 2 pour trouver le plus petit, puis on l'échange avec celui qui était en position 2.
- On a bien mis les 3 plus petits au début du tableau, en ordre croissant, il faut maintenant trier le reste. On parcourt tous les éléments à partir de la position 3 pour trouver le plus petit, puis on l'échange avec celui qui était en position 3.
- ...
- On a bien mis les i plus petits au début du tableau, en ordre croissant, il faut maintenant trier le reste. On parcourt tous les éléments après le i-ieme, pour trouver le plus petit, puis on l'échange avec celui qui était en 3ieme position.
- On a bien mis les n-1 plus petits au début du tableau, en ordre croissant, il ne reste qu'un seul élément qui est le plus grand, il est trié.

Le début du tableau étant déjà trié (jusqu'à i exclus), on parcourt le reste pour trouver le plus petit élément à rajouter en fin de la partie triée (en i).

```
def tri_selection(T):
    n = len(T)
    for i in range(n):   # on suppose T trié jusqu'à i exclus
        i_min = i
        for j in range(i+1, n):
            if T[j] < T[i_min]:
                i_min = j
        T[i], T[i_min] = T[i_min], T[i]
    return T
```

Quelques remarques :

- On peut bien sûr écrire `for i in range(n-1)` puisque à la fin de la boucle le dernier élément (position `n-1`) est de toute fçon le plus grand, inutile de continuer à trier.

- On peut aussi écrire `for j in range(i, n)`, ce qui ne change rien puisque la condition `if T[j] < T[i_min]` est fausse.
- Python permet l'échange des deux valeurs en une seule instruction : `T[i], T[i_min] = T[i_min], T[i]`. Dans d'autres langages où ce n'est pas possible il faut passer par une variable temporaire : `temp = T[i]; T[i] = T[i_min]; T[i_min] = temp;`

Au niveau de la **terminaison** de l'algorithme, les boucles `for` se terminent toujours donc on sait que l'algorithme se terminera, inutile de chercher un **variant de boucle**.

La **correction** est un peu plus compliquée. Ici, l'**invariant de boucle** est le suivant : « Tous les éléments jusqu'à `i` (exclus) sont triés en ordre croissant et inférieurs à tous les éléments de `i` à la fin ». On va faire un raisonnement par récurence pour vérifier que cet invariant est correct :

- Il est clairement vérifié au départ puisqu'il n'y a aucun élément avant l'indice `0`.
- Supposons qu'il est vérifié jusqu'à une valeur de `i` : Tous les éléments jusqu'à `i` (exclus) sont triés en ordre croissant et inférieurs à tous les éléments de `i` à la fin. Le prochain passage dans la boucle `for i in range(n)` consiste à chercher la plus petite valeur entre tous les élements de `i` jusquà la fin et de la mettre en position `i`. Cette nouvelle valeur mise en `i` est bien supérieure à tous les élements qui la précèdent et inférieure à tous ceux qui la suivent. On peut toujours dire que « tous les éléments jusqu'à `i+1` (exclus) sont triés en ordre croissant et inférieurs à tous les éléments de `i+1` à la fin », l'invariant est donc toujours vérifié.

Une fois que la boucle `for i in range(n)` se termine, l'invariant de boucle nous assure que tous les élements jusqu'à `n` (exclus) sont triés en ordre croissant. On a donc bien **prouvé la correction** de l'algorithme.

Etudions maintenant le **coût, ou complexité temporelle** de l'algorithme pour un tableau de grande taille n. La boucle `for i in range(n)` se répète donc n fois. Et à chaque répetition, la boucle `for j in range(i+1, n)` va faire $n-1$ comparaisons `if T[j] < T[i_min]` et quelques opérations, puis $n-2$ comparaisons, puis $n-3$ comparaisons, etc., jusqu'à 0. Au total, on va faire : $(n-1)+(n-2)+(n-3)+\ldots+2+1$ comparaisons et quelques opérations supplémentaires. Ceux qui connaissent la formule savent que cette somme est égale à $\frac{n\times(n-1)}{2}$, mais seul l'ordre de grandeur nous intéresse ici et on vois bien que c'est de l'ordre de $n\times n$, autrement dit que le **coût est quadratique en** $O(n^2)$.

Cours

Algorithme : À étape on cherche le plus petit élément parmi les élements restants et on l'ajoute au tableau trié.

Terminaison : Les boucles `for` se terminent.

Correction/Invariant de boucle : Les éléments jusqu'à `i` (exclus) sont triés en ordre croissant et inférieurs à tous les éléments de `i` à la fin.

Coût (ou complexité) : Quadratique, en *O(n)*.