Algorithmes de tris

S6 - Algorithmique (1)

Dans cette partie du cours, nous allons étudier deux algorithmes de tris : le tri par insertion et le tri par sélection.

Étant donné un tableau de nombres, l'objectif est d'écrire une fonction qui renvoie un tableau contenant les mêmes nombres mais dans l'ordre croissant.

1. Tri par insertion

Le principe

> **!** Principe de l'algorithme
>
> En commençant par le deuxième élément du tableau :
>
> - On compare l'élément courant avec l'élément précédent.
> - Si l'élément courant est plus petit, on échange les deux éléments.
> - On continue à comparer et échanger l'élément courant avec les éléments précédents jusqu'à ce que l'élément courant soit plus grand que l'élément précédent.

Animation du tri par insertion :

Programmation

```
def tri_insertion(tableau: list) -> list:
    """Tri en place par insertion le tableau passé en paramètre."""
    for i in range(1, len(tableau)):                                    ①
        j = i                                                           ②
        while j > 0 and tableau[j] < tableau[j-1]:                      ③
            tableau[j], tableau[j-1] = tableau[j-1], tableau[j]         ④
            j -= 1                                                      ⑤
    return tableau
```

① On commence à l'indice 1 qui correspond au deuxième élément du tableau.
② On stocke l'indice courant dans une variable `j` pour pouvoir le modifier.
③ Tant que l'indice courant est supérieur à 0 et que l'élément courant est plus petit que l'élément précédent, on échange les deux éléments.
④ On échange les deux éléments.
⑤ L'élément courant est maintenant l'élément précédent, on décrémente donc l'indice courant.

Test de l'algorithme :

```
tri_insertion([5, 2, 4, 6, 1, 3])
```

```
[1, 2, 3, 4, 5, 6]
```

Preuve de terminaison

Montrons que l'algorithme termine.

D'une part, il est certain que la boucle `for`, boucle bornée par nature, se termine. D'autre part, la boucle `while` se termine aussi. La variable `j` peut être est un **variant de boucle**. À chaque itération, sa valeur de diminue de 1 : elle finit donc toujours par atteindre 0.

La terminaison de l'algorithme est donc prouvée.

Preuve de correction

Montrons que l'algorithme trie bien le tableau.

Pour cela, considérons la propriété suivante : à chaque itération, le sous-tableau composé des `i` premiers éléments est trié. Montrons que cette propriété est un **invariant de boucle**.

- **Initialisation** : au début de l'algorithme, le sous-tableau composé uniquement du premier élément est trié.
- **Conservation** : supposons que le le sous-tableau composé des `i` premiers éléments est trié : $[e_0, e_1, ..., e_{i-1}]$ avec $e_0 \leqslant e_1 \leqslant ... \leqslant e_{i-1}$. L'algorithme considère alors l'élément e_i et le compare avec les éléments précédents. Si e_i est plus petit que e_{i-1}, on échange les deux éléments. On continue alors à comparer e_i avec les éléments précédents jusqu'à ce que e_i soit plus grand que l'élément précédent. Le sous-tableau composé des `i+1` premiers éléments est alors trié.
- **Termination** : à la fin de l'algorithme `i` a la valeur `n-1` ce qui correspond à l'indice du dernier élément du tableau. Le sous-tableau composé des `n` premiers éléments est donc trié. Or, `n` est le nombre d'éléments du tableau, donc le tableau entier est trié.

La correction de l'algorithme est donc prouvée.

Complexité

2. Tri par sélection