NSI 1ère - Algorithmique - Tris 3

QK

Tri par insertion

Exercice

1. Reprendre l'algorithme du tri par insertion avec l'exemple `[1, 3, 4, 2]`
   Décrire l'état du tableau :
   - avant le tri,
   - après chaque tour de la boucle principale,
   - après le tri.
2. Décrire l'algorithme avec les indices.
3. Écrire le code Python de l'algorithme.
4. Terminaison : prouver qu'à chaque tour de la boucle externe, la taille de la partie non triée diminue. Prouver que l'algorithme se termine.
5. Proposer un invariant de boucle pour le tri par insertion.
6. Complexité : le nombre total d'échanges est inférieur à n^2.

Le tri par insertion : correction de l'exercice.

On commence avec une liste déjà triée vide. On itère sur la liste et, à chaque tour on insère le premier élément non trié à sa place dans la liste triée

Propriétés :

- Tri *stable* : il ne change pas l'ordre de deux éléments "égaux"
- Tri en *place* : il n'utilise pas plus de mémoire
- Complexité : la complexité en temps du tri par insertion est quadratique ($O(n^2)$)

1. Exemples

Exemple 1

Triés	Non Triés	Élément le plus à gauche
()	(1, 3, 4, 2)	(1)
(1)	(3, 4, 2)	(3)
(1, 3)	(4, 2)	(4)
(1, 3, 4)	(2)	(4)
(1, 2, 3, 4)		

Exemple 2

Tri par insertion

2. Pseudo code

```
Tri Insertion(tableau t, entier n)
i = 0
Tant que i < n
    j = i
    Tant que j > 0 et t[j-1] > t[j]
        echanger t[j] et t[j-1]
        j = j - 1
    fin tant que
    i = i + 1
fin tant que
```

3. Programme python

```python
def tri_insertion(tableau: list) -> None:
    i = 0
    while i < len(tableau):
        j = i
        while j > 0 and tableau[j-1] > tableau[j]:
            echanger(tableau, j, j - 1)
            j = j - 1
        i = i + 1


def echanger(tableau: list, i: int, j: int) -> None:
    temp = tableau[i]
    tableau[i] = tableau[j]
    tableau[j] = temp
```

4. Terminaison du tri par insertion

Est-on certain que notre algorithme va se termine bien ?

1. À chaque tour de la boucle extérieure, la taille de la liste restante (les non triés) diminue.
2. À chaque tour de la boucle intérieure, `j` augmente. Elle s'arrête bien.

5. Correction : prouver que l'algorithme est correct

Comment s'assurer que l'algorithme fait ce qu'il annonce ?

- Est-on certain d'obtenir une tableau triée à la fin ?
- Est-on certain que l'algorithme s'arrête ?

Invariant de boucle

Un invariant de boucle est une propriété qui est vraie AVANT et APRÈS l'exécution d'un tour de la boucle.

Tri par insertion : invariant de boucle

Les `i` premiers éléments sont triés.

1. C'est vrai **dès le départ** car on commence avec `i=0` et un tableau vide est trié.
2. Cela **reste vrai** à chaque étape. En effet, chaque étape de la boucle déplace un nouvel élément à sa position.
3. Lorsque l'algorithme s'arrête, le tableau complet est trié.

6. Complexité

Nous allons compter le nombre d'échanges.

- La boucle extérieure comporte n tours (n désigne la taille du tableau)
- La boucle intérieure comporte au pire i tours (lorsque le tableau de départ est trié par ordre décroissant).

Il y a donc au pire $C_n = \sum_{i=0}^{n} i$ échanges.

D'après le chapitre suites arithmétiques de première $C_n = \dfrac{n(n-1)}{2}$.

En effet :

$$C_n = 0 + 1 + \cdots + (n-1) + n$$

Écrivons cette somme à l'envers :

$$C_n = n + (n-1) + \cdots + 1 + 0$$

Ajoutons en colonne :

$$2C_n = n + n + \cdots + n + n$$

$$2C_n = n \times (n+1)$$

$$C_n = \frac{n(n+1)}{2}$$

Le nombre d'échanges est, au pire, inférieur à un polynôme en n^2.

On dit que le tri par insertion est *quadratique*.

La compliqué en temps du tri par insertion est $O(n^2)$

Remarque ce calcul de complexité suppose que :

1. Tous les échanges prennent sensiblement le même temps,
2. Les autres opérations (affectation, décrément, incrément) prennent toutes le même temps.

C'est vrai en Python.