Tri par insertion

1 Description

Le tri par insertion considère un par un chaque élément de la liste et l'insère à la bonne place parmi les éléments déjà triés. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés. C'est un tri de complexité $O(n^2)$.

Pour trouver la place où insérer un élément parmi les précédents, il faut le comparer à ces derniers, et les décaler afin de libérer une place où effectuer l'insertion. Le décalage occupe la place laissée libre par l'élément considéré. En pratique, ces deux actions s'effectuent en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

2 Exemple

Voici les étapes de l'exécution du tri par insertion sur la liste `[6, 5, 3, 1, 8, 7, 2, 4]`.
La liste est représentée au début et à la fin de chaque itération.

i = 1 :	6	5	3	1	8	7	2	4	→	5	6	3	1	8	7	2	4
i = 2 :	5	6	3	1	8	7	2	4	→	3	5	6	1	8	7	2	4
i = 3 :	3	5	6	1	8	7	2	4	→	1	3	5	6	8	7	2	4
i = 4 :	1	3	5	6	8	7	2	4	→	1	3	5	6	8	7	2	4
i = 5 :	1	3	5	6	8	7	2	4	→	1	3	5	6	7	8	2	4
i = 6 :	1	3	5	6	7	8	2	4	→	1	2	3	5	6	7	8	4
i = 7 :	1	2	3	5	6	7	8	4	→	1	2	3	4	5	6	7	8

Élément déjà trié Élément à trier

Détaillons l'itération 6 de l'exemple. La valeur à trier est le 2 et on compare les valeurs précédemment triées de la liste (en gris).

Tant que valeur comparée est plus grande que 2, on la décale (copie) à l'indice +1.
Si valeur est plus petite que 2 ou que l'on est arrivé au début de la liste, le 2 est à sa place.

j = 6	1	3	5	6	7	8		4
j = 5	1	3	5	6	7		8	4
j = 4	1	3	5	6		7	8	4
j = 3	1	3	5		6	7	8	4
j = 2	1	3		5	6	7	8	4
j = 1	1	3	3	5	6	7	8	4
	1	2	3	5	6	7	8	4

8 est plus grand que 2, on le décale.
7 est plus grand que 2, on le décale.
6 est plus grand que 2, on le décale.
5 est plus grand que 2, on le décale.
3 est plus grand que 2, on le décale.
1 n'est pas plus grand que 2, on insère 2.

Élément déjà trié Élément à trier Cellule vide

3 Pseudo-code

```
procédure tri_insertion(liste l)

    pour i de 1 à taille(l) - 1

        # mémoriser l[i] dans x
        x ← l[i]

        # décaler les éléments l[0]..l[i-1] qui sont plus grands que x,
        # en partant de l[i-1]
        j ← i
        tant que j > 0 et l[j-1] > x
            l[j] ← l[j-1]
            j ← j - 1

        # placer x dans le "trou" laissé par le décalage
        l[j] ← x
```

Exercice 1 : Trier la liste suivante en utilisant le tri par insertion.

i = 1 :	8	4	7	5	2	3	1	6	⟶	4	8	7	5	2	3	1	6
i = 2 :	4	8	7	5	2	3	1	6	⟶	4	7	8	5	2	3	1	6
i = 3 :	4	7	8	5	2	3	1	6	⟶	4	5	7	8	2	3	1	6
i = 4 :	4	5	7	8	2	3	1	6	⟶	2	4	5	7	8	3	1	6
i = 5 :	2	4	5	7	8	3	1	6	⟶	2	3	4	5	7	8	1	6
i = 6 :	2	3	4	5	7	8	1	6	⟶	1	2	3	4	5	7	8	6
i = 7 :	1	2	3	4	5	7	8	6	⟶	1	2	3	4	5	6	7	8

Exercice 2 : Coder l'algorithme du tri par insertion en Python. Aider vous du pseudo-code si besoin. Coder le test pour la liste de l'exercice 1.

```
1   def tri_insertion(l):
2       """
3       Effectue un tri par insertion sur la liste l
4       >>> tri_insertion([8, 4, 7, 5, 2, 3, 1, 6])
5       [1, 2, 3, 4, 5, 6, 7, 8]
6       """
7
8       for i in range(1, len(l)):
9           # mémoriser T[i] dans x
10          x = l[i]
11          # décaler les éléments T[0]..T[i-1] qui sont plus grands que
12          # x, en partant de T[i-1]
13          j = i
14          while j > 0 and l[j-1] > x:
15              l[j] = l[j-1]
16              j = j - 1
17          # placer x dans le "trou" laissé par le décalage
18          l[j] = x
19      return l
```