Tri par insertion

I Description

Le tri par insertion considère chaque élément du tableau 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.

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.

Le tri par insertion est un tri stable (conservant l'ordre d'apparition des éléments égaux) et un tri en place (il n'utilise pas de tableau auxiliaire).
L'algorithme a la particularité d'être online, c'est-à-dire qu'il peut recevoir la liste à trier élément par élément sans perdre en efficacité.

II Exemple

Voici les étapes de l'exécution du tri par insertion sur le tableau [6, 5, 3, 1, 8, 7, 2, 4].
Le tableau est représenté 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

Exercice : Trier le tableau suivant.

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

III Pseudo-code

```
procédure tri_insertion(tableau T)

    pour i de 1 à taille(T) - 1

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

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

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

Détaillons cette itération de l'exemple :

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

Nous avons `i` égale à 6, et nous mémorisons la valeur à trier `x` = 2. On pose ensuite `j` = `i` = 6.
On compare les valeurs précédemment triés du tableau (en gris) avec `x`.

Tant que valeur à l'indice `j-1` est plus grande que `x`, la valeur à l'indice `j-1` est copiée à l'indice `j` puis on décrémente `j`.
Si valeur à l'indice `j-1` est plus petite que `x` ou que `j` = 0 (début du tableau), on place `x` à l'indice `j`.

j = 6	1	3	5	6	7	8	2	4	8 est plus grand que 2, on le décale.
j = 5	1	3	5	6	7	8	8	4	7 est plus grand que 2, on le décale.
j = 4	1	3	5	6	7	7	8	4	6 est plus grand que 2, on le décale.
j = 3	1	3	5	6	6	7	8	4	5 est plus grand que 2, on le décale.
j = 2	1	3	5	5	6	7	8	4	3 est plus grand que 2, on le décale.
j = 1	1	3	3	5	6	7	8	4	1 n'est pas plus grand que 2, on insère 2.
	1	2	3	5	6	7	8	4	

Élément déjà trié Élément à trier Cellule vide

Exercice : Compléter le code python suivant.

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