Le tri par insertion

1 Tri par insertion

1.1 Principe

Trier les deux premiers, insérer le troisième parmi les deux premiers déjà triés,
insérer le quatrième parmi les trois premiers déjà triés...

1.2 Complexité

Notons $C(n)$ le nombre de comparaisons à effectuer ; si la liste est déjà triée ?

Réponse

$C(n) = n - 1$

Dans le pire des cas ?

Réponse

$C(n) = \frac{n(n-1)}{2}$

En moyenne ?

Réponse

$$C(n) = \frac{n(n-1)}{4}$$

Chaque élément est comparé en moyenne à la moitié des précédents.

1.3 Comment tester expérimentalement le caractère quadratique ?

On peut comparer le temps d'éxecution pour n, puis $2n$, $4n$...
On peut aussi examiner $\frac{T(n)}{n^2}$ pour différentes valeurs de n.

1.4 Amélioration

On pourrait effectuer une recherche dichotomique pour trouver l'emplacement où on insère.
Le nombre de comparaisons est alors dominé par $(n.\ln n)$.
Mais la complexité de l'algorithme reste quadratique.

2 Programmes

```python
import random as rd
from time import clock

def tri_insertion1(L):
    for indice in range(1, len(L)):
        a = L[indice]
        j = indice - 1
        while j >= 0 and L[j] > a:
            L[j+1] = L[j]
            j -= 1
        L[j+1] = a


# deux variantes avec 'slicing'
# légèrement plus rapides

def tri_insertion2(L):
    for indice in range(1, len(L)):
        a = L[indice]
        j = indice - 1
        while j >= 0 and L[j] > a:
            j -= 1
        L[j+2 : indice+1] = L[j+1 : indice]
        L[j+1] = a

def tri_insertion3(L):
    for indice in range(1, len(L)):
        a = L[indice]
        j = indice - 1
        while j >= 0 and L[j] > a:
            j -= 1
        L[indice : indice + 1]= []
        L[j+1 : j+1] = [a]


n = 10

L = [rd.randint(0, 2*n) for k in range(n)]
print(L)
t1 = clock()
tri_insertion1(L)
t2 = clock()
print(L)
print('temps1 : ', t2-t1)

print('\n')

L = [rd.randint(0, 2*n) for k in range(n)]
print(L)
t1 = clock()
tri_insertion2(L)
t2 = clock()
print(L)
print('temps2 : ', t2-t1)
```