

```
# -*- coding: utf-8 -*-  
"""
```

```
Created on Sat Nov 23 14:31:54 2019
```

```
@author: dconduche  
"""
```

```
import numpy as np
```

```
def triInsertion(L):
```

```
    """Trie par insertion une liste L. Tri en place.
```

```
    L -- liste de nombres
```

```
    ne retourne rien : la liste L est modifiée par effet de bord.
```

```
    """
```

```
    for i in range(1, len(L)):
```

```
        k = i
```

```
        v = L[i]
```

```
        while (k > 0) and (v < L[k-1]):
```

```
            L[k] = L[k-1]
```

```
            k = k-1
```

```
        L[k] = v
```

```
    "Complexité du tri par insertion :  $O(n^2)$ "
```

```
def quicksort(L):
```

```
    if len(L) <= 1:
```

```
        return L
```

```
    else:
```

```
        pivot = L[0]
```

```
        plus_grand = []
```

```
        plus_petit = []
```

```
        for x in L[1:]:
```

```
            if x >= pivot:
```

```
                plus_grand.append(x)
```

```
            else:
```

```
                plus_petit.append(x)
```

```
        return quicksort(plus_petit) + [pivot] + quicksort(plus_grand)
```

```
    "Complexité du tri rapide :  $O(n \log(n))$ "
```

```
def quicksortAl(L):
```

```
    """Quicksort avec pivot aléatoire"""
```

```
    if len(L) <= 1:
```

```
        return L
```

```
    else:
```

```
        k = np.random.randint(len(L))
```

```
        pivot = L.pop(k)
```

```
        plus_grand = []
```

```
        plus_petit = []
```

```
        for x in L[1:]:
```

```
            if x >= pivot:
```

```
                plus_grand.append(x)
```

```
            else:
```

```
                plus_petit.append(x)
```

```
        return quicksortAl(plus_petit) + [pivot] + quicksortAl(plus_grand)
```

```

def fusion(L1, L2):
    """fusionne deux listes triées"""
    if L1 == []:
        return L2
    if L2 == []:
        return L1
    #Désormais, les deux listes sont non vides.
    if L1[0] < L2[0]:
        return [L1[0]] + fusion(L1[1:], L2)
    else:
        return [L2[0]] + fusion(L1, L2[1:])

def tri_fusion(L):
    if len(L) <= 1:
        return L
    m = len(L)//2
    return fusion(tri_fusion(L[:m]), tri_fusion(L[m:]))

"Complexité du tri fusion :  $O(n \log(n))$ "

```