Dans tout le cours, L désigne une liste non vide de réels distincts, de longueur $n \geqslant 1$. EXEMPLE : $L = [4, 3, 7, 9, 8, 5]$

Tri par insertion

Principe : • On laisse le 1er élément de la liste à sa place.
• On prend le 2e élément et on l'insère en bonne position dans la liste $L[:1]$.
• On prend le 3e élément et on l'insère en bonne position dans la liste $L[:2]$
• et ainsi de suite jusqu'au bout de la liste.

```
def insertion(L,i) :
    cle=L[i]
    p=i
    while p>0 and cle<L[p-1] :
        L[p]=L[p-1]
        p=p-1
    L[p]=cle

def triInsertion(L) :
    n=len(L)
    for i in range(1,n) :
        insertion(L,i)
    return L
```

Tri par sélection

Principe : • On recherche le minimum de la liste L, on le place en première position par un échange .
• On recherche le minimum de la liste restante (privée du premier élément), on le place en deuxième position par un échange.
• Et ainsi de suite...

```
def tri_selection(L) :
    n = len(L)
    for i in range(0,n) :
        min = i
        for j in range(i+1,n) :
            if L[j] < L[min] :
                min = j
        if min!=i :
            L[i],L[min]=L[min],L[i]
    return L
```

Meilleur des cas : $L = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
Pire des cas : $L = [10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0]$

Tri fusion (*"merge sort"*)

Principe : • On découpe la liste à trier en deux listes de longueurs à peu près identiques (si $len(L) = n$, choisissons la première liste de longueur $n//2$).
• On trie récursivement ces deux listes.
• On fusionne les deux listes triées en une liste triée.

```
def fusion(L1, L2) :

    L = []
    n1, n2 = len(L1), len(L2)
    i, j = 0, 0
    while i < n1 and j < n2 :
        if L1[i] < L2[j] :
            L.append(L1[i])
            i = i + 1
        else :
            L.append(L2[j])
            j = j + 1
    while i < n1 :
        L.append(L1[i])
        i = i + 1
    while j < n2 :
        L.append(L2[j])
        j = j + 1
    return L

def tri_fusion(L) :
    n = len(L)
    if n == 1 :
        return L
    else :
        L1 = tri_fusion(L[0 :n//2])
        L2 = tri_fusion(L[n//2 :])
        return fusion(L1, L2)
```

Toutes les listes ont même ordre de complexité. Pas de meilleur ou de pire cas.

Tri rapide (*"quicksort"*)

Principe : • On sélectionne arbitrairement un élément clé de la liste L à trier (le dernier par exemple).
• On crée deux tableaux L1 et L2 qui contiennent respectivement les éléments de L inférieurs et supérieurs à clé.
• Par un appel récursif, on trie les tableaux L1 et L2.
• On fusionne alors L1, [clé] et L2.

```
def tri_rapide(L) :
    if len(L) <= 1 :
        return L
    L1, L2 = [], []
    cle = L.pop()
    for x in L :
        if x < cle :
            L1.append(x)
        else :
            L2.append(x)
    return tri_rapide(L1)+[cle]+tri_rapide(L2)
```

Meilleur des cas : $L = [5, 0, 1, 4, 3, 2, 8, 7, 6, 10, 9]$
Pire des cas : $L = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 0]$

Complexité des 4 algorithmes

Tri	Complexité en moyenne	Complexité pire des cas	Complexité meilleur des cas
Insertion	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n)$
Sélection	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$
Fusion	$\Theta(n \ln n)$	$\Theta(n \ln n)$	$\Theta(n \ln n)$
Rapide	$\Theta(n \ln n)$	$\Theta(n^2)$	$\Theta(n \ln n)$

Pour animations, voir :
http ://lwh.free.fr/pages/algo/tri/tri_fusion.html