```
##############################################
##TRI RAPIDE#####################################

def partition(a):
    '''partitionne le tableau a en deux,le premier tableau renvoyé contient les éléments inférieurs
            ou égaux au pivot (sans le pivot), le deuxième tableau contient les éléments strictement supérieurs'''

    pivot=a[0]  #on choisit toujours le premier élément de la liste comme pivot
    n=len(a)
    b=[]   #b contiendra les éléments plus petits que le pivot
    c=[]   #c contiendra les éléments plus grands que le pivot
    for x in a[1:n]:
        if x<=pivot:
            b.append(x)
        else:
            c.append(x)

    return b,pivot,c

def partitionv2(a):
    '''partitionne le tableau a en deux,la première partie étant inférieure à un élément du tableau choisi comme pivot,
    le deuxième partie étant constituée d'éléments supérieurs.
    Le résultat renvoyé est ce tableau partitionné et la position du pivot'''
    pivot=a[0]                    # on choisit le premier élément de la liste comme pivot
    m=0
    for i in range(1,len(a)):     # on parcourt un par un tous les éléments de la liste
        if a[i]<pivot:
            a[i],a[m+1]=a[m+1],a[i]
            m=m+1
    a[0],a[m]=a[m],a[0]           # on place le pivot au bon endroit
    return a,m



def tri_rapide(t):

    if len(t)==1 or len(t)==0:  # si la liste est vide ou contient un seul élément, elle est déjà triée
        return(t)

    b,pivot,c=partition(t)             # on partionne la liste autour du pivot
    t=tri_rapide(b)+[pivot]+tri_rapide(c) # on fait un appel récursif
                                          #en appliquant la fonction aux deux
                                          sous-tableaux
    return t

##############################################
##TRI FUSION#####################################


def fusion(a,b):
    '''réalise la fusion de deux tableaux a et b déjà triés'''

    t=[]
    i=0;j=0

    while i<len(a) and j<len(b):  #la boucle tourne tant qu'au moins une des listes n'est pas parcourue entièrement
        if a[i]<b[j]:
            t.append(a[i])
            i=i+1
        else:
            t.append(b[j])
            j=j+1

    if i==len(a):       # s'il s'agit de la liste a qui est épuisée, on complète t avec les éléments restants de b
        t=t+b[j:len(b)]
```

```
65      if j==len(b):       # s'il s'agit de la liste b qui est épuisée, on complète t
        avec les éléments restants de a
66          t=t+a[i:len(a)]
67
68      return t
69
70
71  def tri_fusion(t):
72
73      if len(t)==0 or len(t)==1:  # si la liste est vide ou contient un seul élément,
74                                      #elle est déjà triée
75          return t
76
77      n=len(t)                # on repère la position moitié (ou presque) dans la liste
78      t1=tri_fusion(t[0:n//2  ])       # appel récursif pour trier la première moitié
        du tableau
79
80      t2=tri_fusion(t[n//2  :len(t)])  # appel récursif pour trier la deuxième moitié
        du tableau
81
82      return fusion(t1,t2)      # on retourne la fusion des deux tableaux triés
83
84
85  ##############################################
86  ##TRI PAR INSERTION#############################
87
88  def tri_insertion(a):
89      '''retourne un tableau avec les mêmes éléments triés par ordre croissant'''
90
91      n=len(a)
92
93      for i in range(1,n):  #on parcourt un par un tous les éléments de la liste
94          x=a[i]              #x est l'élément à insérer dans la partie de liste déjà
            triée
95          j=i                   #on va parcourir la partie de liste déjà triée ,
96                                      #de la droite vers la gauche
97          while j>0 and x<a[j-1]: #tant qu'on est pas au début de la liste
98                                      #et que les elts rencontrés sont plus grand que x
99              a[j]=a[j-1]         #on décale vers la droite les éléments
100             j=j-1
101         a[j]=x   #on insère x à la bonne place
102     return a
103
104  ##############################################
105 ##TRI A BULLES#############################
106  def tri_bulle(t):
107     n = len(t)
108     for i in range(n-1):
109         for j in range(n-i-1):
110             if t[j] > t[j+1]:
111                 t[j], t[j+1] = t[j+1], t[j]
112     return t
113
114 ##############################################
115 ##TRI PAR SELECTION#############################
116
117  def indice_min(t,g,d):
118     '''renvoie l'indice du plus petit élément de la partie de t comprise entre g
        inclus et d inclus'''
119     i_min=g
120     min=t[g]
121     for i in range(g+1,d+1):
122         if t[i]<min:
123             min=t[i]
124             i_min=i
125     return i_min
126
127 def tri_selection_iteratif(t):
128     n=len(t)
129     for k in range(n-1):
130         j=indice_min(t,k,n-1)  #position du minimum dans le tableau à partir de
            l'indice k
```

```
131             t[k],t[j]=t[j],t[k]
132         return t
133
134     def tri_selection_recursif(t):
135         n=len(t)
136         if n==0 or n==1:
137             return(t)   #si le tableau est vide ou de taille 1 alors il est déjà trié
138         j=indice_min(t,0,n-1) #position du minimum du tableau
139         t[0],t[j]=t[j],t[0]  #on le met en première position
140         return [t[0]]+tri_selection_recursif(t[1:n]) #le premier élément est bien placé
            et on trie la partie restante
141
142     ##########  COMPLEXITE  ###################
143     #la complexité de la fonction indice_min est linéaire puisque l'on parcourt une fois
        le tableau
144     #la complexité du tri par sélection version itérative est donc quadratique puisque
        l'on parcourt une fois le tableau,
145         #en appelant à chaque fois la fonction indice_min
146
147     #la complexité du tri par sélection version récursive vérifie: C(n)=5+a+bn+C(n-1) ,
        où a+bn est la complexité dela fonction indice_min  .
148        #On procédant par itérations successives, on conjecture facilement que
           C(n)=O(n^2), complexité quadratique.
```