```python
#Correction_info5

#EX1

def fusion(L1:list,L2:list)->list:
    """réalise la fusion entre deux listes triées par ordre ccroissant"""
    n1,n2=len(L1),len(L2)
    i1,i2=0,0 #on définit des compteurs qui permettront de parcourir les deux listes.
    L=[]
    while i1<n1 and i2<n2:
        if L1[i1]<L2[i2]: #on compare les premiers éléments des listes
            L.append(L1[i1])
            i1=i1+1
        else:
            L.append(L2[i2])
            i2=i2+1
    if i1==n1 and i2!=n2: #la liste L1 a été épuisée, on ajoute la fin de L2
        L=L+L2[i2:]
    if i2==n2 and i1!=n1: #la liste L2 a été épuisée, on ajoute la fin de L1
        L=L+L1[i1:]
    return L

def trifusion(L:list)->list:
    """renvoie une liste triée par tri fusion"""
    if len(L)==1 or len(L)==0:
        return L
    else:
        m=len(L)//2
        L1=L[0:m]
        L2=L[m:len(L)]
        return fusion(trifusion(L1),trifusion(L2))

#A chaque étape, on appelle le programme sur la moitié des éléments de la liste... à
la fin, on est donc ramené à la condition initiale.

#EX2

def tribulles(L:list):
    for k in range(0,len(L)): #on va faire n passages pour replacer tous les éléments
        for i in range(0,len(L)-k-1): #puis, à chaque passage, on remonte la bulle si
nécessaire, par des échanges successifs.
            if L[i+1]<L[i]:
                L[i],L[i+1]=L[i+1],L[i]
            else:
                pass
    return L

#EX3

def selectmin(L:list):
    m,ind=L[0],0  #on va comparer le min aux autres éléments
    for i in range(0,len(L)):
        if L[i]<=m:
            m,ind=L[i],i
        else:
            pass
    return m,ind

def triselection(L:list):
    L2=[] #on va compléter cette nouvelle liste au fur et à mesure
    for i in range(0,len(L)):
        m,ind=selectmin(L)
        L2.append(m) #on stocke la valeur min
        del(L[ind]) #puis, on l'extrait de la liste initiale
    return L2

#EX4
```



```python
def insertion(L:list,x:float):
    #on va chercher la place de x dans cette liste triée
    i=0
    while i<len(L) and x>L[i]: #attention avec la boucle while, il faut éviter de
sortir de la liste : ce sont des effets de bord qu'il faut gérer convenablement
        i=i+1
    return L[:i]+[x]+L[i:]

def triinsertion(L:list):
    L2=[L[0]] #on initialise notre liste triée
    for i in range(1,len(L)):
        L2=insertion(L2,L[i]) #puis on insère dans L2 les éléments à chaque étape
    return L2

#EX5

def tririapide(L:list):
    if len(L)==0 or len(L)==1:
        return L
    else:
        m=len(L)//2
        L1,L2=[],[]
        for k in range(0,len(L)):
            if k!=m and L[k]<=L[m]:
                L1.append(L[k])
            elif k!=m and L[k]>L[m]:
                L2.append(L[k])
        return tririapide(L1)+[L[m]]+tririapide(L2)

#En fait, à chaque étape, on peut considérer qu'on a placé L[m], et ainsi on est
ramené à des sous-problèmes de taille n-1, puis n-2... ce qui permet d'accrocher la
condition d'arrêt à un moment. Le programme peut alors remonter la pile d'appels
récursifs pour renvoyer la liste triée.
```