Exercice 7. Tri par sélection en place

Question 1 –

```
# Renvoie l'indice de l'élément minimal de L[i_min:]
# Hypothèse: L[i_min:] est non vide
def mini_ter(L, i_min):
    res = i_min
    for i in range(i_min+1, len(L)):
        if L[i] < L[i_min]:
            i_min = i
    return i_min
```

```
def tri_selection_EP(L):
    for i in range(len(L)):
        i_min = mini_ter(L, i)
        tmp = L[i]
        L[i] = L[i_min]
        L[i_min] = tmp
```

Question 2 – C'est bien un tri en place, puisque la quantité de mémoire utilisée est indépendante de la taille de la liste en entrée.

Le tri n'est pas stable. Par exemple pour la liste `[3.0, 3, 0]`, on obtient `[0, 3, 3.0]`.

Exercice 8. Tri fusion

Question 1 –

```
# Hypothèse: T1 et T2 sont triées
def fusion(T1, T2):
    T = []
    i1 = 0; i2 = 0
    while i1 < len(T1) and i2 < len(T2):
        if T1[i1] <= T2[i2]:
            T.append(T1[i1])
            i1 += 1
        else:
            T.append(T2[i2])
            i2 += 1
    return T + T1[i1:] + T2[i2:]  # T1[i1:] ou T2[i2:]  est vide
```

```
def tri_fusion(L):
    if len(L) <= 1:
        return L[:]
    else:
        m = (len(L)+1)//2
        L1 = L[:m]; L2 = L[m:]
        T1 = tri_fusion(L1); T2 = tri_fusion(L2)
        return fusion(T1, T2)
```

Question 2 – Ce n'est pas un tri en place car une nouvelle liste est créée.

C'est un tri stable. Le point important est que dans la boucle `while` de la fonction `fusion`, si `T1[i1] == T2[i2]` alors on ajoute `T1[i1]` avant `T2[i2]` dans `T`.

Exercice 9. Sélection en temps linéaire (facultatif)

Question 1 –

```
def get_L_med(L):
    L_med = []
    for i in range(0, len(L), 5):
        T = sorted(L[i: i+5])
        L_med.append(T[(len(T)-1)//2])
    return L_med
```

```
def separer_bis(L, pivot):
    L1 = []; L2 = []
    for e in L:
        if e < pivot:
            L1.append(e)
        elif e > pivot:
            L2.append(e)
    return L1, L2
```

```
# Hypothèse: les éléments de L sont distincts deux à deux
# Hypothèse: L est non vide
# Hypothèse: 0 <= i < len(L)
def selection(L, i):
    if len(L) <= 5:
        T = sorted(L)
        return T[i]
    L_med = get_L_med(L)
    med_med = selection(L_med, (len(L_med)-1)//2)
    L1, L2 = separer_bis(L, med_med)
    if len(L1) == i:
        return med_med
    elif len(L1) < i:
        return selection(L2, i-len(L1)-1)
    else:
        return selection(L1, i)
```