```python
# JL - 03/17
## Une bonne fois pour toute

def echange(T, i, j):
    "T[i] <-> T[j]"
    T[i], T[j] = T[j], T[i]

# Complexité : O(1)

################################
## Exercice 1 : Tri selection ##
################################

def indice_min(T, i):
    "renvoie l'indice du minimum de T[i:]"
    imin = i
    for j in range(i + 1, len(T)):
        # imin contient l'indice du minimum de T[i:j]
        if T[j] < T[imin]:
            imin = j
    return imin

# Complexité : O(n - i) où n = len(T)

def tri_selection(T):
    "effectue le tri selection de T sur place"
    n = len(T)
    for i in range(n - 1):
        # Les i premiers éléments de T sont triés
        imin = indice_min(T, i)
        echange(T, imin, i)

# Les invariants se prouvent sans problème par récurrence.
# La complexité est O(n^2), et même, en comptant les comparaisons,
# exactement sum(n - i, i = 0..(n - 1)) = n (n - 1) / 2 comparaisons.

# Une version en une seule fonction :

def tri_selection(T):
    "effectue le tri selection de T sur place"
    n = len(T)
    for i in range(n - 1):
        # Les i premiers éléments de T sont triés
        imin = i
        for j in range(i + 1, len(T)):
            # imin contient l'indice du minimum de T[i:j]
            if T[j] < T[imin]:
                imin = j
        T[imin], T[i] = T[i], T[imin]

# Une version récursive :

def indice_min(T, i):
    "renvoie l'indice du minimum de T[i:]"
    if i == len(T) - 1:
        return i
    else:
        imin = indice_min(T, i + 1)
        if T[i] < T[imin]:
            return i
        else:
            return imin
```

```python
def tri_selection(T, i=0):
    "effectue le tri selection de T[i:] sur place"
    if i < len(T):
        imin = indice_min(T, i)
        echange(T, imin, i)
        tri_selection(T, i + 1)

########################################
## Exercice 2 : Tri insertion récursif ##
########################################

def insere(T, i):
    """insere x = T[i] parmi les i
premiers éléments supposés triés"""
    if i > 0 and T[i - 1] > T[i]:
        echange(T, i - 1, i)
        insere(T, i - 1)

def tri_insertion(T, i=0):
    """trie le tableau T[i:] sur place."""
    if i < len(T) - 1:
        insere(T, i)
        tri_insertion(T, i + 1)

################################################
## Exercice 3 : Tri par insertion dichotomique ##
################################################

def position_dicho(T, x, debut, fin):
    """renvoie l'indice de la position de x
dans T[debut:fin] supposé trié, par dichotomie."""
    if T[fin - 1] <= x:
        return fin
    elif T[debut] > x:
        return debut
    else:
        milieu = (debut + fin) // 2
        if T[milieu] == x:
            return milieu
        elif T[milieu] > x:
            return position_dicho(T, x, debut, milieu)
        else:
            return position_dicho(T, x, milieu, fin)

# Complexité : O(ln(fin - debut))

def insere_dicho(T, i):
    """insere x = T[i] parmi les i
premiers éléments supposés triés"""
    x = T[i]
    position = position_dicho(T, x, 0, i)
    for j in range(i - 1, position - 1, -1):
        T[j + 1] = T[j]
    T[position] = x

# Complexité : O(ln(i) + i)

def tri_insertion_dicho(T, i=1):
    """trie le tableau T[i:] sur place."""
    if i < len(T):
        insere_dicho(T, i)
        tri_insertion_dicho(T, i + 1)
```

```python
# Complexité : O(n×ln(n) + n²) = O(n²)

################################
## Exercice 4 : Tri linéaire ##
################################

def tri_lineaire(T, N):
    "trie le tableau T d'entiers entre 0 et N - 1"
    compteurs = [0 for i in range(N)]
    # On compte le nombre d'occurences de chaque entier
    for element in T:
        compteurs[element] += 1

    # Construction du tableau trié
    new_T = []
    for i in range(N):
        for _ in range(compteurs[i]):
            new_T.append(i)

    return new_T

# Complexités linéaire et spatiale : O(n + N).

################################
## Exercice 5 : Buble sort ##
################################


def parcours(T):
    """parcours T en échangeant deux éléments successifs
mal ordonnée et renvoie un booléen indiquant si un échange
a été fait"""
    echange_effectue = False
    for i in range(len(T) - 1):
        if T[i + 1] < T[i]:
            echange(T, i, i + 1)
            echange_effectue = True
    return echange_effectue

def bubble_sort(T):
    """Trie T sur place"""
    while parcours(T):
        pass

# Invariant de sortie du k-e parcours : les k derniers éléments
# de T sont à leur place définitive.
# Complexité : O(n²).

# Vu l'invariant, on peut améliorer légèrement, en parcourant une
# case de moins du tableau à chaque étape.

def bubble_sort_best(T):
    echange_effectue = True
    j = len(T) - 1
    while echange_effectue:
        echange_effectue = False
        for i in range(j):
            if T[i + 1] < T[i]:
                echange(T, i, i + 1)
                echange_effectue = True
        j -= 1
```

```
################################
## Exercice 6 : Cocktail sort ##
################################

def parcours(T, debut, fin):
    """parcours T[debut:fin] en échangeant deux éléments successifs
mal ordonnée et renvoie un booléen indiquant si un échange
a été fait"""
    echange_effectue = False
    for i in range(fin):
        if T[i + 1] < T[i]:
            echange(T, i, i + 1)
            echange_effectue = True
    return echange_effectue

def parcours_inverse(T, debut, fin):
    echange_effectue = False
    for i in range(fin, debut, -1):
        if T[i] < T[i - 1]:
            echange(T, i, i - 1)
            echange_effectue = True
    return echange_effectue

def cocktail_sort(T):
    i = 0
    debut = 0
    fin = len(T) - 1
    encore = True
    while encore:
        if i % 2 == 0:
            encore = parcours(T, debut, fin)
            fin -= 1
        else:
            encore = parcours_inverse(T, debut, fin)
            debut += 1
        i += 1



#########################################################
## Exercice 7 : Tri fusion avec tableau auxiliaire ##
#########################################################



def fusion(T, debut, milieu, fin, aux):
    """remplit aux[debut:fin] par la fusion de T[debut:milieu]
et T[milieu:fin] supposés triés puis remplace T[debut:fin]."""
    i1 = debut # parcours de T[debut:milieu]
    i2 = milieu # parcours de T[milieu:fin]
    for i in range(debut, fin): # parcours de aux
        if i2 == fin or (i1 < milieu and T[i1] < T[i2]):
            aux[i] = T[i1]
            i1 += 1
        else:
            aux[i] = T[i2]
            i2 += 1
    T[debut:fin] = aux[debut:fin]
```

```python
def tri_fusion(T):
    n = len(T)
    aux = [0 for _ in range(n)]
    def tri_aux(T, debut, fin, aux):
        """trie sur place T[debut:fin]"""
        if debut < fin - 1:
            milieu = (debut + fin) // 2
            tri_aux(T, debut, milieu, aux)
            tri_aux(T, milieu, fin, aux)
            fusion(T, debut, milieu, fin, aux)
    tri_aux(T, 0, n, aux)

########################################
## Exercice 8 : Tri rapide + insertion ##
########################################

# Version non en place

from random import randint

def partition(L, i):
    """renvoie un triplet (L1, L2, k) tel que L1 contient les éléments
de L < x, L2 ceux > x et k le nombre d'apparitions de x."""
    L1 = []
    L2 = []
    k = 0
    pivot = L[i]
    for elem in L:
        if elem < pivot:
            L1.append(elem)
        elif elem > pivot:
            L2.append(elem)
        else:
            k += 1
    return L1, L2, k

def tri_rapide(L):
    n = len(L)
    if n <= 5:
        Lc = L.copy()
        tri_insertion(Lc)
        return Lc
    else:
        i = randint(0, n - 1)
        pivot = L[i]
        L1, L2, k = partition(L, i)
        return tri_rapide(L1) + [pivot for _ in range(k)] + tri_rapide(L2)



# Version en place



def tri_insertion2(T, i, j):
    """trie le tableau T[i:j] sur place."""
    if i < j - 1:
        insere(T, i)
        tri_insertion2(T, i + 1, j)
```

```python
def partition2(T, debut, fin, ipivot):
    pivot = T[ipivot]
    echange(T, ipivot, debut) # pivot placé au début
    finpp = debut # les éléments rencontrés de début à finpp sont <= pivot
    for i in range(debut + 1, fin):
        if T[i] <= pivot:
            finpp += 1
            echange(T, finpp, i)
    if finpp > debut:
        echange(T, finpp, debut) # on remet le pivot à se place, en finpp
    return finpp

def tri_rapide2(T):
    def tri_aux(T, debut, fin):
        "tri en place de T[debut:fin]"
        if debut <= fin - 5:
            tri_insertion2(T, debut, fin)
        else:
            ipivot = randint(debut, fin - 1)
            ipivot = partition(T, debut, fin, ipivot)
            tri_aux(T, debut, ipivot)
            tri_aux(T, ipivot + 1, fin)
    tri_aux(T, 0, len(T))
```