## EXERCICE 2 (10 points)

La fonction `tri_insertion` suivante prend en argument un tableau `tab` (type `list`) et trie ce tableau en utilisant la méthode du tri par insertion. Compléter cette fonction pour qu'elle réponde à la spécification demandée.

On rappelle le principe du tri par insertion : on considère les éléments à trier un par un, le premier élément constituant, à lui tout seul, un tableau trié de longueur 1. On range ensuite le second élément pour constituer un tableau trié de longueur 2, puis on range le troisième élément pour avoir un tableau trié de longueur 3 et ainsi de suite…

A chaque étape, le premier élément du sous-tableau non trié est placé dans le sous-tableau des éléments déjà triés de sorte que ce sous-tableau demeure trié.

Le principe du tri par insertion est donc d'insérer à la n-ième itération, le n-ième élément à la bonne place.

```
def tri_insertion(tab):
    '''Trie le tableau tab par ordre croissant
    en appliquant l'algorithme de tri par insertion'''
    n = len(tab)
    for i in range(1, n):
        valeur_insertion = ...
        # la variable j sert à déterminer
        # où placer la valeur à ranger
        j = ...
        # tant qu'on n'a pas trouvé la place de l'élément à
        # insérer on décale les valeurs du tableau vers la droite
        while j > ... and valeur_insertion < tab[...]:
            tab[j] = tab[j-1]
            j = ...
        tab[j] = ...
```

Exemple :

```
>>> tab = [98, 12, 104, 23, 131, 9]
>>> tri_insertion(tab)
>>> tab
[9, 12, 23, 98, 104, 131]
```

## EXERCICE 2 (10 points)

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt le sous-tableau des éléments non rangés et on place le plus petit élément en première position de ce sous-tableau.

Exemple avec le tableau : `t = [41, 55, 21, 18, 12, 6, 25]`

- Étape 1 : on parcourt tous les éléments du tableau, on permute le plus petit élément avec le premier.

  Le tableau devient `t = [`**`6`**`, 55, 21, 18, 12, 41, 25]`

- Étape 2 : on parcourt tous les éléments **sauf le premier**, on permute le plus petit élément trouvé avec le second.

  Le tableau devient : `t = [`**`6, 12`**`, 21, 18, 55, 41, 25]`

Et ainsi de suite.

Le programme ci-dessous implémente cet algorithme.

```
def echange(tab, i, j):
    '''Echange les éléments d'indice i et j dans le tableau tab.'''
    temp = ...
    tab[i] = ...
    tab[j] = ...

def tri_selection(tab):
    '''Trie le tableau tab dans l'ordre croissant
    par la méthode du tri par sélection.'''
    N = len(tab)
    for k in range(...):
        imin = ...
        for i in range(..., N):
            if tab[i] < ...:
                imin = i
        echange(tab, ..., ...)
```

Compléter ce code de façon à obtenir :

```
>>> tab = [41, 55, 21, 18, 12, 6, 25]
>>> tri_selection(tab)
>>> tab
[6, 12, 18, 21, 25, 41, 55]
```

## EXERCICE 2 (10 points)

La fonction `tri_bulles` prend en paramètre un tableau `tab` d'entiers (type `list`) et le modifie pour le trier par ordre croissant.

Le tri à bulles est un tri en place qui commence par placer le plus grand élément en dernière position en parcourant le tableau de gauche à droite et en échangeant au passage les éléments voisins mal ordonnés (si la valeur de l'élément d'indice `i` a une valeur strictement supérieure à celle de l'indice `i + 1`, ils sont échangés). Le tri place ensuite en avant-dernière position le plus grand élément du tableau privé de son dernier élément en procédant encore à des échanges d'éléments voisins. Ce principe est répété jusqu'à placer le minimum en première position.

Exemple : pour trier le tableau `[7, 9, 4, 3]` :

- première étape : 7 et 9 ne sont pas échangés, puis 9 et 4 sont échangés, puis 9 et 3 sont échangés, le tableau est alors `[7, 4, 3, 9]`
- deuxième étape : 7 et 4 sont échangés, puis 7 et 3 sont échangés, le tableau est alors `[4, 3, 7, 9]`
- troisième étape : 4 et 3 sont échangés, le tableau est alors `[3, 4, 7, 9]`

Compléter le code Python ci-dessous qui implémente la fonction tri_bulles.

```
def echange(tab, i, j):
    '''Echange les éléments d'indice i et j dans le tableau tab.'''
    temp = ...
    tab[i] = ...
    tab[j] = ...

def tri_bulles(tab):
    '''Trie le tableau tab dans l'ordre croissant
    par la méthode du tri à bulles.'''
    n = len(tab)
    for i in range(...):
        for j in range(...):
            if ... > ...:
                echange(tab, j, ...)
```

Exemples :

```
>>> tab = []
>>> tri_bulles(tab)
>>> tab
[]
>>> tab2 = [9, 3, 7, 2, 3, 1, 6]
>>> tri_bulles(tab2)
>>> tab2
[1, 2, 3, 3, 6, 7, 9]
>>> tab3 = [9, 7, 4, 3]
>>> tri_bulles(tab3)
>>> tab3
[3, 4, 7, 9]
```

## Exercice 2 (4 points)

*Cet exercice porte sur la programmation et les algorithmes de tri.*

Au service des urgences d'un hôpital, le triage consiste à classifier ou à déterminer le degré de priorité des patients. Il implique une réévaluation périodique et systématique de ce degré pour les patients en attente.

Dans le système informatique, chaque patient obtient un identifiant à son arrivée en salle d'attente ainsi qu'une priorité dépendant de la gravité potentielle de ses symptômes.
Le patient ayant la priorité 1 est le premier qui doit être pris en charge et deux patients ne peuvent pas avoir la même priorité.

Pour modéliser la salle d'attente en Python, chaque patient est représenté par un tuple composé de son identifiant d'arrivée et de sa priorité. Ainsi, la variable `attente` implémentée ci-dessous représente une salle d'attente de trois patients où, à cet instant, le patient identifié 47 sera le premier à être pris en charge, puis le patient 45 et finalement le patient 49.

```
attente = [(45,2),(47,1),(49,3)]
```

**1.** Écrire l'instruction qui permet d'insérer dans la liste `attente`, définie ci-dessus, un nouveau patient identifié 50 avec une priorité de 4.

**2.** Pour optimiser le traitement informatisé de prise en charge des patients, on veut que la salle d'attente soit ordonnée par priorité. On utilise alors la fonction `tri(attente)` donnée ci-dessous qui renvoie la salle d'attente triée dans l'ordre croissant des priorités.

```
def tri(attente) :
    for i in range(len(attente)) :
        pos = i
        mini = attente[i][1]
        for j in range(i, len(attente)) :
            if attente[j][1] < mini :
                pos = j
                mini = attente[j][1]
        temp = attente[i]
        attente[i]= attente[pos]
        attente[pos] = temp
```

**a.** Quel algorithme de tri est ici implémenté ?

- ✓ le tri fusion
- ✓ le tri par insertion
- ✓ le tri par sélection
- ✓ le tri rapide

**b.** Quelle est la complexité en temps des tris par insertion et par sélection ?

- ✓ constante : $O(1)$
- ✓ logarithmique : $O(\log n)$
- ✓ linéaire : $O(n)$
- ✓ quadratique : $O(n^2)$