Pour chacune des fonctions écrites ci-dessous, on pourra utiliser `Python Tutor` *pour une meilleure visualisation.*

Exercice 1

1. On a le tableau suivant :

p	0	1	2	3
pmin	3	2	2	4
L	[2,4,3,7,5]	[2,3,4,7,5]	[2,3,4,7,5]	[2,3,4,5,7]

2. Pendant toute la fonction la liste est découpé en une partie triée (à gauche) et une partie à trier. A chaque itération on cherche le plus petit élément de la partie à trier et on le place en dernière position de la partie triée.
3. Ce tri s'appelle le tri par sélection. Sa différence avec celui vu en cours est que l'on recherche ici le minimum de la partie non triée, alors que dans le cours, on cherchait le maximum de la partie non triée.
4. D'autres tris existent comme, par exemple, le tri par insertion.

Exercice 2 La fonction suivant convient :

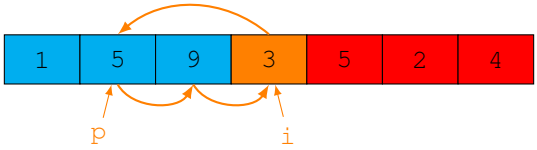
```
def TriSelectionDecroissant(L) :
    n = len(L)
    for p in range(n-1) :
        pmax = p
        for j in range(p,n) :
            if L[j] > L[pmax]:
                pmax = j
        L[pmax],L[p] = L[p],L[pmax]
    return L
```

Exercice 3

1. On a la fonction suivante :

```
def TriInsertion(L):
    n = len(L)
    for i in range(1,n):
        temp = L[i]
        p = 0
        while L[p] < temp:
            p = p + 1
        for j in range(i-1,p-1,-1):
            L[j+1] = L[j]
        L[p] = temp
    return L
```

2. Le compteur `i` correspond à l'index du premier élément de la liste non triée et la variable `p` correspond à l'index de la position où il faut insérer l'élément `L[i]`. On a ainsi le schéma suivant :



3. Notons n la longueur de la liste. La boucle `while` effectue `p` comparaisons et la boucle sur `j` effectue `i-p+1` comparaisons. La fonction effectue donc

$$\sum_{i=2}^{n}(i+1) \text{ comparaisons dans le pire des cas.}$$

On effectue donc un ordre de grandeur de $O(n^2)$ dans le pire cas, ce qui est comparable avec ce que l'on a vu dans le cours.

Exercice 4 Il suffit de changer l'instruction `L[p] < temp` par l'instruction `L[p] > temp`.

Exercice 5 (Tri par dénombrement) La fonction suivante répond à la question :

```
def TriDenombrement(L):
  N = [0 for i in range(K)]
  for i in range(n):
    N[L[i]] = N[L[i]] + 1
  k = 0
  for i in range(K):
    for j in range(N[i])
      L[k + j] = i
    k = k + N[i]
  return L
```

Vérification de la complexité :

- La première boucle est exécutée n fois.
- La double boucle est exécutée $N[1] + ... + N[K]$ fois et on a $N[i] \leqslant n$ pour tout i. Elle est donc exécutée au plus Kn fois.

La complexité de l'algorithme est donc bien linéaire.