Informatique en CPGE (2017-201)
Corrigé TD 4 : algorithmes de tri

Exercice 1 : tri par sélection

1. Exemple de programme :

```
def tri_selection(liste):
    for i in range(len(liste)-1):
        k = i # indice du minimum
        for j in range(i+1,len(liste)):
            # on cherche s'il y a un élément plus petit
            if liste[j] < liste[k]:
                k = j # et on stoche son indice
                # puis on échange les éléments
        liste[i],liste[k]=liste[k],liste[i]
    return liste
```

2. Programme du tri par insertion ; on ajoute une instruction permettant l'affichage de la liste à chaque étape.

```
def tri_insertion(liste):
    for i in range(len(liste)-1):
        k = i+1 # indice de la cle
        cle=liste[k]
        while cle<liste[k-1] and k>0:
            liste[k] = liste[k-1]
            k = k-1
        liste[k]=cle
        print("etape",i+1," :\t",liste)
    return liste
```

Si l'on interrompt l'exécution de l'algorithme du tri par sélection après k étapes, on obtient un tableau qui contient les k premiers éléments du tableau final calculé par l'algorithme.

Avec le tri par insertion, on trie d'abord le premier élément du tableau initial, puis les deux premiers, puis les trois premiers, etc. Si l'on interrompt l'exécution de l'algorithme du tri par insertion après k étapes, on obtient un tableau qui contient un tableau ordonné des k premiers éléments du tableau initial.

liste : [8, 3, 4, 6, 9, 1, 2, 5, 7]
test après k étapes avec le tri selection :
étape 1 : [1, 3, 4, 6, 9, 8, 2, 5, 7]
étape 2 : [1, 2, 4, 6, 9, 8, 3, 5, 7]
étape 3 : [1, 2, 3, 6, 9, 8, 4, 5, 7]
étape 4 : [1, 2, 3, 4, 9, 8, 6, 5, 7]
étape 5 : [1, 2, 3, 4, 5, 8, 6, 9, 7]
étape 6 : [1, 2, 3, 4, 5, 6, 8, 9, 7]
étape 7 : [1, 2, 3, 4, 5, 6, 7, 9, 8]
étape 8 : [1, 2, 3, 4, 5, 6, 7, 8, 9]

liste : [8, 3, 4, 6, 9, 1, 2, 5, 7]

test après k étapes avec le tri insertion :
étape 1 : [3, 8, 4, 6, 9, 1, 2, 5, 7]
étape 2 : [3, 4, 8, 6, 9, 1, 2, 5, 7]
étape 3 : [3, 4, 6, 8, 9, 1, 2, 5, 7]
étape 4 : [3, 4, 6, 8, 9, 1, 2, 5, 7]
étape 5 : [1, 3, 4, 6, 8, 9, 2, 5, 7]
étape 6 : [1, 2, 3, 4, 6, 8, 9, 5, 7]
étape 7 : [1, 2, 3, 4, 5, 6, 8, 9, 7]
étape 8 : [1, 2, 3, 4, 5, 6, 7, 8, 9]

3. Le tri par sélection est très simple mais peu efficace. Dans le pire des cas comme dans le meilleur des cas (et donc en moyenne) sa complexité est en $\mathcal{O}(n^2)$.

Exercice 2 : tri à bulles

1. Tableau à trier : [18, 10, 15, 11, 5]
   étape 1 : [10, 18, 15, 11, 5]
   étape 2 : [10, 15, 18, 11, 5]
   étape 3 : [10, 15, 11, 18, 5]
   étape 4 : [10, 15, 11, 5, 18]
   étape 5 : [10, 11, 15, 5, 18]
   étape 6 : [10, 11, 5, 15, 18]
   étape 7 : [10, 5, 11, 15, 18]
   étape 8 : [5, 10, 11, 15, 18]
2. Sur un tableau ordonné à n éléments, on ne parcourt la liste qu'une seule fois, donc la complexité est en $\mathcal{O}(n)$. C'est le meilleur des cas.
3. On considère le tableau : $[n, n-1, \ldots, 3, 2, 1]$.
   (a) Le nombre de permutations nécessaires pour amener n à la position correcte est $n-1$.
   (b) Pour amener $n-1$ à la bonne place il faut $n-2$ permutations et pour amener $n-2$ à la bonne place il en faut $n-3$.
   (c) Le nombre total de permutations nécessaires pour trier un tableau de taille n rangé initialement en ordre décroissant est donc $(n-1)+(n-2)+\ldots+1 = \dfrac{n(n-1)}{2}$.
4. Programme du tri à bulles ; on ajoute une instruction permettant l'affichage de la liste à chaque étape et un compteur qui permet d'arrêter le tri si on n'a pas trouvé d'éléments consécutifs rangés dans le désordre lors d'un parcours de la liste.

```
def tri_bulles(liste):
    for j in range(len(liste)-1,0,-1):
        cpt=0
        for i in range(j):
            if liste[i]>liste[i+1]:
                liste[i],liste[i+1]=liste[i+1],liste[i]
                cpt+=1
            print("etape",i+1," :\t",liste)
        if cpt==0:
            return liste
    return liste
```