C7 Algorithmes de tri

1. Préambule

Pourquoi étudier des algorithmes de tri ?
Autant ne pas le cacher, ces algorithmes sont déjà implémentés (quelque soit le langage) dans des fonctions très performantes.

En Python, on utilise la fonction `sort()` :

🐍 **Script Python**

```
>>> tab = [4, 8, 1, 2, 6]
>>> tab.sort()
>>> tab
[1, 2, 4, 6, 8]
```

Le meilleur de nos futurs algorithmes de tri sera moins efficace que celui de cette fonction `sort()`...
Malgré cela, il est essentiel de se confronter à l'élaboration manuelle d'un algorithme de tri.
Le tri par insertion est le premier des deux algorithmes de tri que nous allons étudier (nous étudierons aussi le tri par sélection).
Ces deux algorithmes ont pour particularité de :

- ne pas nécessiter la création d'une nouvelle liste. Ils modifient la liste à trier sur place.
- ne pas faire intervenir de fonctions complexes.

2. Activités

2.1.

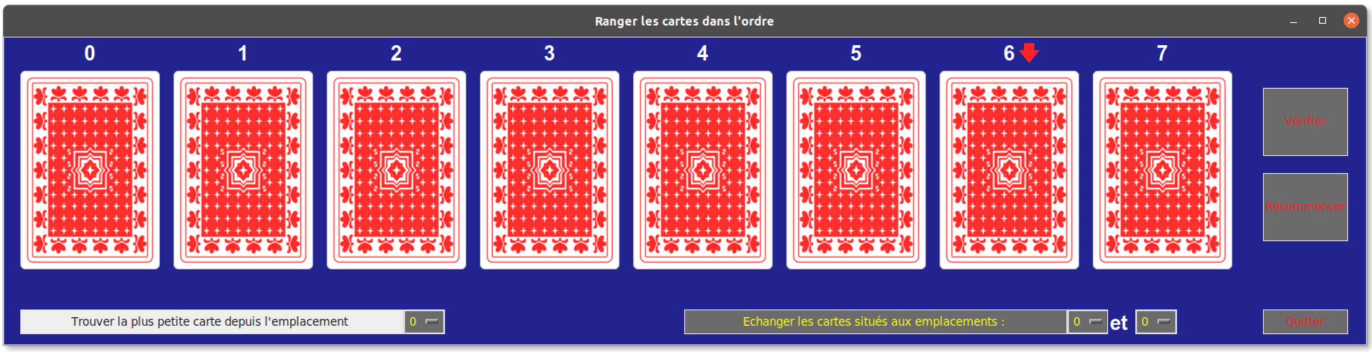
■
Activité 1 : *Tri par sélection*

1. Commencer par télécharger une application Python :
    - **Tri par sélection**
    - Copier ce fichier dans le répertoire de votre choix
    - Faire un clic droit sur le fichier compressé et choisir *Extraire ici*
    - Lancer le programme Python `activite1.py`.

2. Dans cette activité, on doit ranger des cartes par ordre croissant mais **sans les voir**, on dispose par contre de deux boutons :

    - Un bouton Trouver la plus petit carte depuis l'emplacement qui permet de savoir quelle carte est la plus petite à partir de l'emplacement qu'on sélectionne dans le menu déroulant à côté.
    - Un bouton Echanger les cartes situés aux emplacements qui permet d'échanger les cartes situés aux emplacements sélectionnés dans les menus déroulants.

    Voici une capture d'écran de l'application dans laquelle on vient de sélectionner la plus petite carte depuis l'emplacement 0, elle est alors indiquée par une flèche rouge au-dessus (emplacement 6) :

3. Proposer un algorithme permettant à un ordinateur de ranger une suite de nombres par ordre croissant.
4. Implémentation en python

    a. Ecrire une fonction `echange(liste,i,j)` qui échange les éléments d'indice `i` et `j` de la liste `liste` par exemple si `liste=[12,17,10,11,32]` alors après `echange(liste,0,2)` le contenu de `liste` sera `[10,17,12,11,32]`.

    b. Ecrire une fonction `min_depuis(liste,i)` qui renvoie le minimum de la liste `liste` à partir de l'indice `i` par exemple `min_depuis([10,17,12,11,32],2)` renvoie `11`.

    c. En utilisant ces deux fonctions, proposer une implémentation en Python de l'algorithme du tri par sélection.

2.2.

■
Activité 2 : *Tri par insertion*

1. De même que dans l'activité précédente, commencer par télécharger une application Python :

    - **Tri par insertion**
    - Copier ce fichier dans le répertoire de votre choix
    - Faire un clic droit sur le fichier compressé et choisir *Extraire ici*
    - Lancer le programme Python `activite2.py`.

2. De même que dans l'activité précédente, il faut ranger les cartes dans l'ordre *sans les voir*, on dispose d'un unique bouton permettant d'échanger une carte dont on donne le numéro avec sa voisine *si elles ne sont pas dans le bon ordre*
3. Proposer un algorithme permettant de ranger une liste par ordre croissant en utilisant comme seul "*ingrédient*" l'échange de deux cartes dont on donne les emplacements.

   Aide

   Bien évidemment, des boucles et des tests seront aussi nécessaires

4. Proposer une implémentation en Python de cet algorithme

   Aide

   On pourra utiliser la fonction `echange` définie dans l'activité précédente.

5. Tester cette fonction

3. Cours

Vous pouvez télécharger une copie au format pdf du diaporama de synthèse de cours présenté en classe :

Diaporama de cours

Attention

Ce diaporama ne vous donne que quelques points de repères lors de vos révisions. Il devrait être complété par la relecture attentive de vos **propres** notes de cours et par une révision approfondie des exercices.

4. Cours : tri par insertion

4.1. Principe et algorithme

Considérons la liste `[5,4,9,7,2,1,8,0,6,3]`
Voici le fonctionnement de l'algorithme :

5	4	9	7	2	1	8	0	6	3

Vidéo Tri par insertion

Tri par insertion

Explications :

- On traite successivement toutes les valeurs à trier, en commençant par celle en deuxième position.
- Traitement : tant que la valeur à traiter est inférieure à celle située à sa gauche, on échange ces deux valeurs.

4.2. Codage de l'algorithme

Algorithme :

Pour toutes les valeurs, en commençant par la deuxième :

- Tant qu'on trouve à gauche une valeur supérieure et qu'on n'est pas revenu à la première valeur, on échange ces deux valeurs.

Tri par insertion (version simple)

❤

🐍 **Script Python**

```python
def tri_insertion1(liste):
    '''trie en place la liste lst donnée en paramètre'''
    for ind in range(len(liste)-1):                   #(1)
        pos = ind                                     #(2)
        while pos >= 0 and liste[pos+1] < liste[pos] :       #(3)
            liste[pos], liste[pos+1] = liste[pos+1], liste[pos]  #(4)
            pos = pos - 1                             #(5)
```

1. On commence à 0 et on finit à longueur -1.
2. On «duplique» la variable `ind` en une variable `pos`.
   On se positionne sur l'élément d'indice `pos`. On va faire «reculer» cet élément tant que c'est possible. On ne touche pas à `ind`.
3. Tant qu'on n'est pas revenu au début de la liste et qu'il y a une valeur plus grande à gauche.
4. On échange de place avec l'élément précédent.
5. Notre élément est maintenant à l'indice `pos - 1`.
   La boucle peut continuer.

Application :

🐍 **Script Python**

```python
>>> maliste = [7, 5, 2, 8, 1, 4]
>>> tri_insertion1(maliste)
>>> maliste
[1, 2, 4, 5, 7, 8]
```

A vous

Réaliser le tri par insertion de la liste suivante : `[27,10,12,8,11]`
Ecrire toutes les étapes.

A vous

Réaliser le tri par insertion de la liste suivante : `[9,6,1,4,8]`
Ecrire toutes les étapes.

4.3. Complexité de l'algorithme

🐍 **Script Python**

```python
def tri_insertion(liste):
    '''trie en place la liste lst donnée en paramètre'''
    for ind in range(1, len(liste)-1):                 #(1)
        pos = ind                                      #(2)
        while pos >= 0 and liste[pos+1] < liste[pos] :     #(3)
            liste[pos], liste[pos+1] = liste[pos+1], liste[pos]  #(4)
            pos = pos - 1                              #(5)
```

1. On commence à 1 et non pas à 0.
2. On «duplique» la variable `ind` en une variable `pos`.
   On se positionne sur l'élément d'indice `pos`. On va faire «reculer» cet élément tant que c'est possible. On ne touche pas à `ind`.
3. Tant qu'on n'est pas revenu au début de la liste et qu'il y a une valeur plus grande à gauche.
4. On échange de place avec l'élément précédent.
5. Notre élément est maintenant à l'indice `pos - 1`.
   La boucle peut continuer.

4.3.1. Démonstration

Dénombrons le nombre d'opérations dans le pire des cas, pour une liste de taille n.

- boucle `for` : elle s'exécute $n-1$ fois.
- boucle `while` : dans le pire des cas, elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$. Or

$$1+2+3+\dots+n-1=\dfrac{n \times (n-1)}{2}$$

Le terme de plus haut degré de l'expression $\dfrac{n \times (n-1)}{2}$ est de degré 2 : le nombre d'opérations effectuées est donc proportionnel au **carré** de la taille des données d'entrée.
Ceci démontre que le tri par insertion est de complexité **quadratique** noté $O(n^2)$.

Dans le cas (rare, mais il faut l'envisager) où la liste est déjà triée, on ne rentre jamais dans la boucle `while` : le nombre d'opérations est dans ce cas égal à $n-1$, ce qui caractérise une complexité linéaire.

4.4. Résumé de la complexité

- dans le meilleur des cas (liste déjà triée) : complexité **linéaire**

- dans le pire des cas (liste triée dans l'ordre décroissant) : complexité **quadratique**

4.5. Preuve de la terminaison de l'algorithme

Est-on sûr que notre algorithme va s'arrêter ?
Le programme est constitué d'une boucle `while` imbriquée dans une boucle `for`. Seule la boucle `while` peut provoquer une non-terminaison de l'algorithme. Observons donc ses conditions de sortie :

🐍 **Script Python**

```python
while  pos >= 0 and liste[pos+1] < liste[pos]  :
```

La condition `liste[pos+1] < liste[pos]` ne peut pas être rendue fausse avec certitude. Par contre, la condition `pos >= 0` sera fausse dès que la variable `pos` deviendra négative. Or la ligne `pos = pos - 1` nous assure que la variable `pos` diminuera à chaque tour de boucle. La condition `pos >= 0` deviendra alors forcément fausse au bout d'un certain temps.

Nous avonc donc prouvé la **terminaison** de l'algorithme.

Vocabulaire

On dit que la valeur `pos` est un **variant de boucle**.
C'est une notion théorique (ici illustrée de manière simple par la valeur `pos`) qui permet de prouver *la bonne sortie d'une boucle* et donc la terminaison d'un algorithme.

4.5.1. Pour aller plus loin : Preuve de la correction de l'algorithme

Les preuves de correction sont des preuves théoriques. La preuve ici s'appuie sur le concept mathématique de **récurrence**. Principe du raisonnement par récurrence : une propriété \(P(n)\) est vraie si :

- \(P(0)\) (par exemple) est vraie
- Pour tout entier naturel \(n\), si \(P(n)\) est vraie alors \(P(n+1)\) est vraie.

Ici, la propriété serait : « Quand \(k\) varie entre 0 et `longueur(liste) -1`, la sous-liste de longueur \(k\) est triée dans l'ordre croissant.»

Aide

On appelle cette propriété un **invariant de boucle**.
Invariant siginifie qu'elle reste vraie pour chaque boucle.

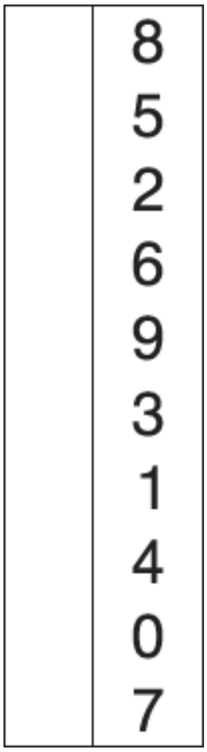
- quand \(k\) vaut 0, on place le minimum de la liste en l[0], la sous-liste l[0] est donc triée.

- si la sous-liste de \(k\) éléments est triée, l'algorithme rajoute en dernière position de la liste le minimum de la sous-liste restante, dont tous les éléments sont supérieurs au maximum de la sous-liste de \(k\) éléments. La sous-liste de \(k+1\) éléments est donc aussi triée.

5. Cours : tri par sélection

5.1. Animation

Considérons la liste `[8,5,2,6,9,3,1,4,8,7]`
Voici le fonctionnement de l'algorithme :



Vidéo Tri par sélection

Tri par sélection

5.2. Principe

description de l'algorithme

Le travail se fait essentiellement sur les **indices**.

- du premier élément jusqu'à l'avant-dernier :
  - on considère que cet élément est l'élément minimum, on stocke donc son indice dans une variable *indice du minimum*.
  - on parcourt les éléments suivants, et si on repère un élémént plus petit que notre mininum on met à jour notre *indice du minimum*.
  - une fois le parcours fini, on échange l'élément de travail avec l'élément minimum qui a été trouvé.

5.3. Implémentation de l'algorithme

Tri par sélection

❤

🐍 **Script Python**

```python
def tri_selection(lst) :
    for k in range(len(lst)-1):
        indice_min = k
        for i in range(k+1, len(lst)) :
            if lst[i] < lst[indice_min]:
                indice_min = i
        lst[k], lst[indice_min] = lst[indice_min], lst[k]
```

Vérification :

🐍 **Script Python**

```python
>>> ma_liste = [7, 5, 2, 8, 1, 4]
>>> tri_selection(ma_liste)
>>> ma_liste
[1, 2, 4, 5, 7, 8]
```

5.4. Complexité de l'algorithme

5.4.1. Calcul du nombre d'opérations

Dénombrons le nombre d'opérations, pour une liste de taille n.

- boucle `for` : elle s'exécute $n-1$ fois.
- deuxième boucle `for` imbriquée : elle exécute d'abord 1 opération, puis 2, puis 3... jusqu'à $n-1$.

Or $1+2+3+\dots+n-1=\dfrac{n \times (n-1)}{2}$

Ceci est bien un polynôme du second degré, ce qui confirme que la complexité de ce tri est quadratique.

6. QCM

1. On applique l'algorithme du tri par sélection à la liste `[9,11,7,16]`, après la première étape, le contenu de la liste sera :

Réponses Correction

a) `[11,9,7,16]`

b) `[7,11,9,16]`

c) `[16,11,7,9]`

d) Aucune des propositions ci-dessus

a) ~~[11,9,7,16]~~

b) `[7,11,9,16]`

c) ~~[16,11,7,9]~~

d) ~~Aucune des propositions ci-dessus~~

2. On applique l'algorithme du tri par insertion à la liste `[9,11,7,16]`, quel sera le contenu de la liste après le premier échange ?

Réponses Correction

a) `[11,9,7,16]`

b) `[9,11,16,7]`

c) `[9,7,11,16]`

d) Aucune des propositions ci-dessus

a) ~~`[11,9,7,16]`~~

b) ~~`[9,11,16,7]`~~

c) `[9,7,11,16]`

d) ~~Aucune des propositions ci-dessus~~

3. L'algorithme du tri par insertion a une complexité :

Réponses Correction

a) logarithmique

b) linéaire

c) quadratique

d) exponentielle

a) ~~logarithmique~~

b) ~~linéaire~~

c) quadratique

d) ~~exponentielle~~

4. Un programme de tri par insertion prend environ 1 seconde pour trier une liste de \(10\,000\) éléments, combien de temps prendra-t-il environ pour trier une liste de \(100\,000\) éléments ?

Réponses **Correction**

a) 1 seconde

b) 10 secondes

c) 100 secondes

d) 1000 secondes

a) ~~1 seconde~~

b) ~~10 secondes~~

c) 100 secondes

d) ~~1000 secondes~~

5. Quelles sont les deux lignes manquantes dans la fonction ci-dessus qui renvoie le minimum d'une liste non vide :

```
1 def min_liste(liste):
2     elt_min = ....
3     for elt in liste:
4         if elt<elt_min:
5             ......
6     return elt_min
```

Réponses **Correction**

a) La ligne 2 est
`elt_min=liste[1]` et
la ligne 5 est
`elt_min=elt`

b) La ligne 2 est
`elt_min=liste[1]` et
la ligne 5 est
`elt=elt_min`

c) La ligne 2 est
`elt_min=liste[0]` et
la ligne 5 est
`elt=elt_min`

d) La ligne 2 est
`elt_min=liste[0]` et
la ligne 5 est
`elt_min=elt`

a) ~~La ligne 2 est~~
~~`elt_min=liste[1]` et~~
~~la ligne 5 est~~
~~`elt_min=elt`~~

b) ~~La ligne 2 est~~
~~`elt_min=liste[1]` et~~
~~la ligne 5 est~~
~~`elt=elt_min`~~

c) ~~La ligne 2 est~~
~~`elt_min=liste[0]` et~~

~~la ligne 5 est~~

~~`elt=elt_min`~~

d) La ligne 2 est

`elt_min=liste[0]` et

la ligne 5 est

`elt_min=elt`

7. Exercices

Fonction `echange(liste,i,j)`

🐍 **Script Python**

```python
def echange(liste,i,j):
    liste[i],liste[j] = liste[j],liste[i]
```

Fonctionnement du tri par sélection

Enoncé **Correction**

1. Ecrire les étapes du tri par sélection pour la liste `[12,19,10,13,11,15,9,14]`
2. Même question pour la liste `["P","R","O","G","R","A","M","M","E"]`

1. 🐍 **Script Python**

```
[12, 19, 10, 13, 11, 15, 9, 14]
[9, 19, 10, 13, 11, 15, 12, 14]
[9, 10, 19, 13, 11, 15, 12, 14]
[9, 10, 11, 13, 19, 15, 12, 14]
[9, 10, 11, 12, 19, 15, 13, 14]
[9, 10, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 14, 19, 15]
[9, 10, 11, 12, 13, 14, 15, 19]
[9, 10, 11, 12, 13, 14, 15, 19]
```

2. 🐍 **Script Python**

```
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'E']
['A', 'R', 'O', 'G', 'R', 'P', 'M', 'M', 'E']
['A', 'E', 'O', 'G', 'R', 'P', 'M', 'M', 'R']
['A', 'E', 'G', 'O', 'R', 'P', 'M', 'M', 'R']
['A', 'E', 'G', 'M', 'R', 'P', 'O', 'M', 'R']
['A', 'E', 'G', 'M', 'M', 'P', 'O', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
```

Fonctionnement du tri par insertion

Enoncé **Correction**

1. Ecrire les étapes du tri par insertion pour la liste `[12,19,10,13,11,15,9,14]`
2. Même question pour la liste `["P","R","O","G","R","A","M","M","E"]`

1.

🐍 **Script Python**

```
[12, 19, 10, 13, 11, 15, 9, 14]
[12, 19, 10, 13, 11, 15, 9, 14]
[12, 10, 19, 13, 11, 15, 9, 14]
[10, 12, 19, 13, 11, 15, 9, 14]
[10, 12, 13, 19, 11, 15, 9, 14]
[10, 12, 13, 11, 19, 15, 9, 14]
[10, 12, 11, 13, 19, 15, 9, 14]
[10, 11, 12, 13, 19, 15, 9, 14]
[10, 11, 12, 13, 15, 19, 9, 14]
[10, 11, 12, 13, 15, 9, 19, 14]
[10, 11, 12, 13, 9, 15, 19, 14]
[10, 11, 12, 9, 13, 15, 19, 14]
[10, 11, 9, 12, 13, 15, 19, 14]
[10, 9, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 15, 19, 14]
[9, 10, 11, 12, 13, 15, 14, 19]
[9, 10, 11, 12, 13, 14, 15, 19]
```

2.

🐍 **Script Python**

```
['P', 'R', 'O', 'G', 'R', 'A', 'M', 'M', 'E']
['P', 'O', 'R', 'G', 'R', 'A', 'M', 'M', 'E']
['O', 'P', 'R', 'G', 'R', 'A', 'M', 'M', 'E']
['O', 'P', 'G', 'R', 'R', 'A', 'M', 'M', 'E']
['O', 'G', 'P', 'R', 'R', 'A', 'M', 'M', 'E']
['G', 'O', 'P', 'R', 'R', 'A', 'M', 'M', 'E']
['G', 'O', 'P', 'R', 'A', 'R', 'M', 'M', 'E']
['G', 'O', 'P', 'A', 'R', 'R', 'M', 'M', 'E']
['G', 'O', 'A', 'P', 'R', 'R', 'M', 'M', 'E']
['G', 'A', 'O', 'P', 'R', 'R', 'M', 'M', 'E']
['A', 'G', 'O', 'P', 'R', 'R', 'M', 'M', 'E']
['A', 'G', 'O', 'P', 'R', 'M', 'R', 'M', 'E']
['A', 'G', 'O', 'P', 'M', 'R', 'R', 'M', 'E']
['A', 'G', 'O', 'M', 'P', 'R', 'R', 'M', 'E']
['A', 'G', 'M', 'O', 'P', 'R', 'R', 'M', 'E']
['A', 'G', 'M', 'O', 'P', 'R', 'M', 'R', 'E']
['A', 'G', 'M', 'O', 'P', 'M', 'R', 'R', 'E']
['A', 'G', 'M', 'O', 'M', 'P', 'R', 'R', 'E']
['A', 'G', 'M', 'M', 'O', 'P', 'R', 'R', 'E']
['A', 'G', 'M', 'M', 'O', 'P', 'R', 'E', 'R']
```

```
['A', 'G', 'M', 'M', 'O', 'P', 'E', 'R', 'R']
['A', 'G', 'M', 'M', 'O', 'E', 'P', 'R', 'R']
['A', 'G', 'M', 'M', 'E', 'O', 'P', 'R', 'R']
['A', 'G', 'M', 'E', 'M', 'O', 'P', 'R', 'R']
['A', 'G', 'E', 'M', 'M', 'O', 'P', 'R', 'R']
['A', 'E', 'G', 'M', 'M', 'O', 'P', 'R', 'R']
```

Tri par ordre décroissant

Enoncé **Correction**

1. On donne ci-dessous l'implémentation du tri par sélection vu en cours :

   🐍 **Script Python**

   ```python
   def echange(liste,i,j):
   liste[i],liste[j] = liste[j],liste[i]

   def min_liste(liste,ind):
       elt_min = liste[ind]
       ind_min=ind
       for k in range(ind,len(liste)):
           if liste[k]<elt_min:
               elt_min=liste[k]
               ind_min=k
       return ind_min

   def tri_selection(liste):
       longueur = len(liste)
       for ind in range(longueur):
           ind_min =
   min_liste(liste,ind)
           echange(liste,ind,ind_min)
   ```

   Modifier cette (ces) fonction(s) afin d'effectuer un tri dans l'ordre décroissant.

2. Même question pour l'algorithme du tri par insertion ci-dessous :

   🐍 **Script Python**

   ```python
   def tri_insertion(liste):
       for ind in
   range(1,len(liste)-1):
           j = ind
           while liste[j+1]<liste[j]
   and j>=0:
               echange(liste,j,j+1)
               j=j-1
   ```

🐍 **Script Python**

```python
def echange(liste,i,j):
    liste[i],liste[j] = liste[j],liste[i]
```

```python
def max_liste(liste,ind):
    elt_max = liste[ind]
    ind_max=ind
    for k in range(ind,len(liste)):
        if liste[k]>elt_max:
            elt_max=liste[k]
            ind_max=k
    return ind_max

def tri_selection_inverse(liste):
    longueur = len(liste)
    for ind in range(longueur):
        ind_max = max_liste(liste,ind)
        echange(liste,ind,ind_max)
```

Tri dans une nouvelle liste

Enoncé Correction

Les algorithmes vus en cours modifient la liste donnée en paramètre, on dit qu'on effectue un *tri en place* c'est à dire directement dans la liste.

1. Modifier la fonction de tri par sélection vu en classe afin d'effectuer le tri en créant une nouvelle liste (et donc sans modifier la liste de départ)
2. Même question pour le tri par insertion

Aide

Comme une *nouvelle liste* est crée, on utilisera l'instruction `return` pour la renvoyer vers le programme principal.

🐍 **Script Python**

```
def tri_selection(liste):
    liste_nouv=[]
    for elt in liste:
        liste_nouv.append(elt)
    longueur = len(liste_nouv)
    for ind in range(longueur):
        ind_min =
min_liste(liste_nouv,ind)
        echange(liste_nouv,ind,ind_min)
    return liste_nouv


def tri_insertion(liste):
    liste_nouv=[]
    for elt in liste:
        liste_nouv.append(elt)
    for ind in
range(0,len(liste_nouv)-1):
```

```python
        j = ind
        while liste_nouv[j+1]<liste_nouv[j] and j>=0:
            echange(liste_nouv,j,j+1)
            j=j-1
    return liste_nouv
```

Liste triée

Enoncé **Correction**

Ecrire une fonction `est_triee` qui prend en argument une `liste` et qui renvoie `True` si `liste` est triée par ordre croissant et `False` dans le cas contraire.

> **Attention**
>
> On ne doit pas trier la liste, simplement vérifier si elle l'est déjà ou pas.

🐍 **Script Python**

```python
def est_trie(liste):
    long=len(liste)
    for ind in range(long-1):
        if liste[ind+1]<liste[ind]:
            return False
    return True
```

Epreuve Pratique

Écrire une fonction tri_selection qui prend en paramètre une liste tab de nombres entiers et qui renvoie le tableau trié par ordre croissant.
On utilisera l’algorithme suivant :

- on recherche le plus petit élément du tableau, et on l'échange avec l'élément d'indice 0 ;
- on recherche le second plus petit élément du tableau, et on l'échange avec l'élément d'indice 1 ;
- on continue de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exemple :

🐍 **Script Python**

```
>>> tri_selection([1,52,6,-9,12])
[-9, 1, 6, 12, 52]
```

🐍 **Script Python**

```
def tri_selection(tab):
    for i in range(len(tab)-1):
        indice_min = i
        for j in range(i+1, len(tab)):
            if tab[j] < tab[indice_min]:
                indice_min = j
        tab[i], tab[indice_min]
```

```
= tab[indice_min], tab[i]
    return tab

#ou version plus découpée, se rapprochant plus de la description de l'algo :

def minimum(tab, i):
    ind_minimum = i
    for j in range(i+1, len(tab)):
        if tab[j] < tab[ind_minimum]:
            ind_minimum = j
    return ind_minimum

def echange(tab, i, j):
    tab[i], tab[j] = tab[j], tab[i]

def tri_selection(tab):
    for i in range(len(tab)-1):
        ind_minimum = minimum(tab, i)
        echange(tab, i, ind_minimum)
    return tab
```

On considère l'algorithme de tri de tableau suivant : à chaque étape, on parcourt depuis le début du tableau tous les éléments non rangés et on place en dernière position le plus grand élément.

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 grand élément avec le dernier.

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

- Étape 2 : on parcourt tous les éléments **sauf le dernier**, on permute le plus grand élément trouvé avec l'avant dernier.

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

Et ainsi de suite. La code de la fonction `tri_iteratif` qui implémente cet algorithme est donné ci-dessous.

```python
def tri_iteratif(tab):
    for k in range(...,
0 ,-1):
        imax = ...
        for i in
range(0, ...):
            if tab[i] > ... :
                imax = i
        if tab[max]
> ... :
            ..., tab[imax]
= tab[imax], ...
    return tab
```

Com
donn

🐍 **Script Python**

```python
>>> tri_iteratif([41, 55,
21, 18, 12, 6, 25])
[6, 12, 18, 21, 25, 41, 55]
```

On rappelle que l'instruction `a, b = b, a` échange les contenus de `a` et `b`.

🐍 **Script Python**

```python
def echange(tab, i, j):
    tab[i], tab[j] = tab[j], tab[i]

def tri_iteratif(tab):
    for k in range(len(tab)-1, 0, -1):
        indice_max = k
        for i in range(0, k):
            if tab[i] > tab[indice_max]:
                indice_max = i
        echange(tab, k, indice_max)
    return tab
```