1 Trier avec les fonctions de bibliothèque

Bloc-Note 1

Python propose deux façons de trier une liste L d'objets comparables (entiers, caractères ...) :

- ☞ `sorted(L)` *retourne une nouvelle liste* triée distincte de L qui n'est pas modifiée ;
- ☞ `L.sort()` *trie sur place* la liste L qui est donc modifiée.

Deux paramètres optionnels sont intéressants :

- ☞ `reverse` permet de préciser l'ordre, par défaut il est croissant mais ont peut trier dans l'ordre décroissant avec `reverse = True`.
- ☞ `key` permet de personnaliser la fonction de comparaison. Par défaut les conteneurs de type `list` ou `tuple` sont comparés élément par élément de gauche à droite mais on peut comparer par exemple des `tuple` à deux éléments de droite à gauche avec `key = lambda t : (t[1], t[0])`. Les fonctions anonymes définies avec `lambda` sont utiles pour définir des fonctions de comparaison.

```
>>> L = ['b','c','a']
>>> M = sorted(L)
>>> M
['a', 'b', 'c']
>>> L
['b', 'c', 'a']
>>> L.sort()
>>> L
['a', 'b', 'c']
>>> L.sort(reverse=True)
>>> L
['c', 'b', 'a']
>>> from random import randint
>>> alea = [randint(-100, 100) for _ in range(10)]
>>> alea
[-16, -40, 79, -83, 85, -89, -96, 56, 6, 72]
>>> dec = sorted(alea, reverse=True)
>>> dec
[85, 79, 72, 56, 6, -16, -40, -83, -89, -96]
>>> L = [(6,3), (4,9), (8,7)]
>>> sorted(L)
[(4, 9), (6, 3), (8, 7)]
>>> sorted(L, key = lambda t : (t[1], t[0]))
[(6, 3), (8, 7), (4, 9)]
>>> sorted(L, key = lambda t : t[0] + t[1])
[(6, 3), (4, 9), (8, 7)]
```

Exemple 1

1. `L` est une liste de 1000 `tuple` de taille 2 représentant des coordonnées de points du plan.
   Quelle instruction permet d'assigner à `M` la liste de ces points triée par distance croissante à l'origine ?

```
>>> from random import randint
>>> L = [(randint(0, 1000), randint(0,1000)) for _ in range(100)]

>>> M = ..........................................................
```

2. `D` est une liste de dates au format (jour, mois, année).
   Quelle instruction permet de trier sur place `D` par ordre calendaire croissant ?

```
>>> D = [(randint(1,7),randint(1,12),randint(0,100)) for _ in range(50)]

>>>  ..........................................................
```

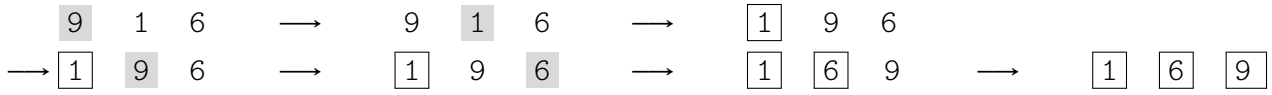
2 Tris quadratiques

Dans un programme, on trie avec les fonctions de bibliothèque mais il est formateur d'implémenter quelques algorithmes de tri classiques.

2.1 Tri par sélection

Bloc-Note 2

Le principe du **tri par sélection** est simple : on identifie le minimum et on l'échange de place avec l'élément d'indice 0, puis on identifie le deuxième plus petit élément et on l'échange avec l'élément d'indice 1 et ainsi de suite jusqu'à ce que le tableau soit trié.
Voici un exemple de déroulement du tri par sélection du tableau d'entiers `[9, 1, 6]`.



Exemple 2 *Implémentation du tri par sélection*

1. Compléter le code de la fonction `index_mini(t, debut)` qui prend en argument un tableau `t` d'entiers et qui retourne l'index du minimum de ce tableau à partir de la position `debut`.

```
def index_mini(t, debut):
    if t == []:
        return None
    imini = debut
    for j in range(debut + 1, len(t)):
        if ...........:
            imini = .......
    return imini
```

Voici un exemple de sortie :

```
In [25]: t = [10,-5,3,-1,0]

In [26]: index_mini(t, 0)
Out[26]: 1

In [27]: index_mini(t, 2)
Out[27]: 3
```

2. On donne ci-dessous la code de la fonction `tri_selection(t)` qui prend en argument un tableau `t` d'entiers et qui le trie sur place dans l'ordre croisant avec l'algorithme de tri par sélection.

```
def tri_selection(t):
    for i in range(0, len(t)):
        #appel de la fonction index_mini
        j = index_mini(t, i)
        #echange des valeurs de t[i] et t[j]
        t[j], t[i] = t[i], t[j]
```

Compléter le tableau d'état des variables ci-dessous (noter les valeurs de `j` et `t` à la fin de chaque tour de boucle) pour l'appel `tri_selection([6, 4, 2, 3])`:

i	j	t
0	.....	.....
1	.....	.....
2	.....	.....
3	.....	.....

3. Le dernier tour de boucle était-il nécessaire ? Pourquoi ? Modifier le code de la fonction `tri_selection(t)`.

.....

.....

.....

.....

.....

.....

.....

.....

4. On considère un tableau `t` de longueur n.

a. Combien de comparaisons sont effectuées par chaque appel de fonction `index_mini(t, i)` ?

.....

.....

.....

b. Combien de comparaisons au total sont effectuées par l'appel de fonction `tri_selection(t)` ?

.....

.....

.....

.....

.....

5. **a.** Avec un navigateur, ouvrir la page :

https://interstices.info/jcms/c_6973/les-algorithmes-de-tri

pour tester l'algorithme de tri par sélection dans l'applet.

b. Écrire une fonction `index_maxi` qui prend en argument un tableau `t` d'entiers et qui retourne l'index du maximum de ce tableau jusqu'à la position `fin`.

```
def index_maxi(t, fin):
    ................
    ................
    ................
    ................
    ................
    ................
    ................
    ................
    ................
```

c. Ecrire une fonction `tri_selection2(t)` qui trie sur place un tableau d'entiers `t` dans l'ordre croissant en utilisant la fonction `index_maxi(t, fin)` .

```
def tri_selection2(t):
    ................
    ................
    ................
    ................
    ................
    ................
    ................
    ................
    ................
```

2.2 Tri par bulles

Bloc-Note 3

Le **tri par bulles** est une variante du **tri par sélection** qui consiste à balayer successivement de gauche à droite le tableau à trier en faisant remonter vers la droite le plus grand élément non trié par échanges d'éléments adjacents.
Dans l'exemple d'exécution ci-dessous, on voit que l'algorithme peut s'arrêter lorsque aucune « bulle » n'est remontée lors du dernier balayage.

3	5	2	1	8	9	7	3	liste de départ
3	2	1	5	8	7	3	9	5 et 9 remontent
2	1	3	5	7	3	8	9	3 et 8 remontent
1	2	3	5	3	7	8	9	2 et 7 remontent
1	2	3	3	5	7	8	9	5 remonte
1	2	3	3	5	7	8	9	aucun élément ne remonte, liste triée

Exemple 3 *Implémentation du tri par bulles*

1. Compléter la fonction `tri_bulles(t)` qui trie sur place une liste d'entiers `t` dans l'ordre croissant avec l'algorithme de tri par bulles.

```
def tri_bulles(t):
    n = len(t)
    for bulle in range(n - 1):
        for k in range(0, ............):
            if .............. :
                ................
```

2. On considère u tableau `t` de longueur n.

   a. Combien de comparaisons sont effectuées par le $k^{\text{ème}}$ balayage ?

   .....

   .....

   b. Combien de comparaisons au total sont effectuées par l'appel de fonction `tri_bulles(t)` ?

   .....

   .....

   .....

   .....

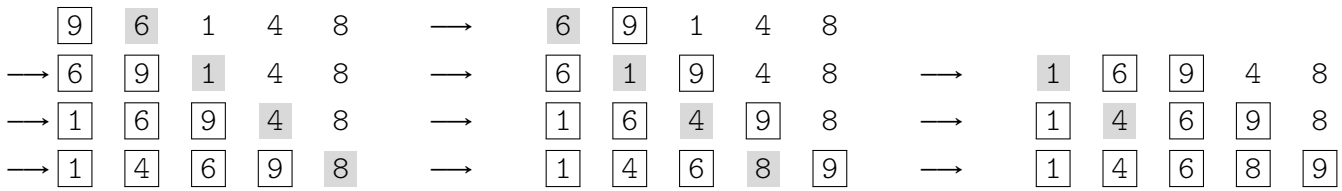
3. Modifier la fonction `tri_bulles` en une fonction `tri_bulles2` où les balayages s'arrêtent lorsque aucune « bulle » n'est remontée lors du dernier balayage.

```
def tri_bulles2(t):
    n = len(t)
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
    ......................
```

2.3 Tri par insertion

Bloc-Note 4

Prenons l'exemple du joueur qui doit trier les cartes de sa donne. Si les cartes de la donne sont posées en pile sur la table, l'algorithme de **tri par insertion** consiste en prendre chaque carte dans l'ordre de la donne et à l'insérer à sa place dans la liste des cartes déjà piochées.
Autrement dit si on dispose d'un tableau de n entiers dont les k premiers éléments sont triés, en insérant le $k+1$-ème élément à sa place parmi les k premiers, on obtient un tableau avec $k+1$ éléments triés. Au départ, le premier élément peut être considéré comme trié donc si on répète cette opération d'insertion $n-1$ fois, on peut faire un *tri sur place* du tableau.
Plusieurs méthodes sont possibles pour insérer le $k+1$-ème élément à sa place, la plus simple est sans doute de procéder par décalages successifs des éléments adjacents :



Exemple 4 *Implémentation du tri par insertion*

1. Exécuter l'application *tri par insertion* sur la page dédiée du site Interstices :

https://interstices.info/jcms/c_6973/les-algorithmes-de-tri

2. Compléter le code de la fonction suivante qui doit insérer à sa place l'élément `t[k]` d'un tableau `t` de nombres telle que `t[0:k]` est trié dans l'ordre croissant.

```
def insertion(t, k):
    ....................
    ....................
    ....................
    ....................
    ....................
    ....................
    ....................
```

3. En utilisant la fonction `insertion(tab, element, k)`, compléter le code de la fonction suivante pour qu'elle réalise le tri par insertion sur place du tableau de nombres `t`.

```
def tri_insertion(t):
    n = len(t)
    for k in range(1, n):
        ....................
        ....................
        ....................
        ....................
```

4. On considère une liste `t` de longueur n.

a. Combien de comparaisons sont effectuées par `insertion(t, e, k)` ?

.....

.....

.....

b. Combien de comparaisons au total sont effectuées par l'appel de fonction `tri_insertion(t)` ?

.....

.....

.....

.....

5. Parmi les trois algorithmes de tri présentés, quels sont ceux qui ont déterminer les k plus petits éléments après k tours de la boucle externe ?

.....

.....

.....

.....