Travail sur les listes

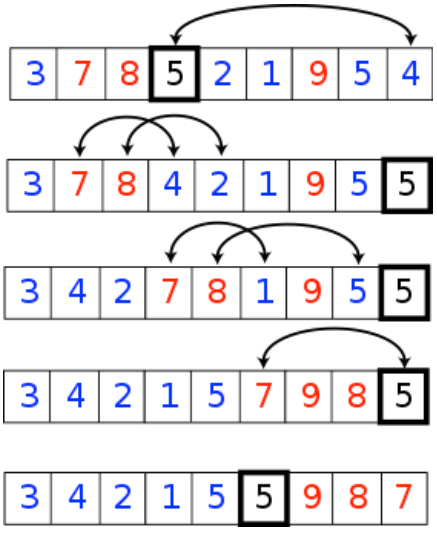
TRI (SORT en anglais)

list_examples.py

```
1  list_numbers = [1, 2, 3, 4, 5, 6, 7, 8]
2
3  print(list_numbers[1:3])
4  print(list_numbers[:4])
5  print(list_numbers[5:])
6  print(list_numbers[:-5])
7
```

Run: list_examples

```
/Users/pankaj/Documents/PycharmProjects/Pyth
[2, 3]
[1, 2, 3, 4]
[6, 7, 8]
[1, 2, 3]

Process finished with exit code 0
```



Travail sur les listes - TRI

1. SLICING avancé sur les listes

Nous avons vu les différentes opérations de base sur les listes, avec des méthodes python (`extend, append, pop, insert`...) mais aussi via le découpage en tranches = slicing.

Ici, nous allons poursuivre dans le slicing en allant un peu plus loin vers le **slicing avancé**. Comment manipuler des sous-listes en une seule instruction (qui cache évidemment des opérations avancées...) ?

```
L1 = [1, 2, 3, 11, 12, 13]
L2 = L1
L1[0:3] = L1[-1:-4:-1] # recopie les 3 derniers éléments
    # de L1 et les place dans les 3 premières cases
print(L1)
print(L2) # L2 est changée car l'opération est faite "en place"
```

```
[13, 12, 11, 11, 12, 13]
[13, 12, 11, 11, 12, 13]
```

Comparer les deux exemples suivants :

- `L1[:] = L1[-1::-1]`, l'opération se fait **en place**

Inversion de l'ordre des éléments d'une liste :

Exemple 1

```
# inversion de l'ordre d'une liste, exemple 1
L1 = [k for k in range(10)]
L2 = L1
L1[:] = L1[-1::-1] # la sous-liste de tous les éléments de L1
# devient la sous-liste des éléments dans l'ordre inverse
print(L1)
print(L2) # L2 est modifiée : l'opération a lieu "en place"
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
```

Exemple 2

```
# inversion de l'ordre d'une liste, exemple 2
L1 = [k for k in range(10)]
L2 = L1
L1 = L1[-1::-1] # la sous-liste de tous les éléments de L1
# devient la sous-liste des éléments dans l'ordre inverse
print(L1)
print(L2) # L2 n'est modifiée  !!
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Travail sur les listes - TRI

Indexation par un slicing :

```
# Dernier exemple : indexation par un sciling
L1 = ['A', 1, 'B', 2, 'C', 3, 'D', 4]
L2 = L1
L1[0::2] = L1[1::2] # les termes d'indice impairs sont
                    # mis dans les termes d'indice pairs
print(L1)
print(L2) # L'opération sur L1 s'effectue 'en place'
```

```
[1, 1, 2, 2, 3, 3, 4, 4]
[1, 1, 2, 2, 3, 3, 4, 4]
```

2. Fonctions importantes sur les listes

Nous allons introduire des fonction utilies et importantes afin de travailler sur les listes, à savoir :

✔ Mise à zéro des valeurs d'une liste inférieures à un seuil
✔ Recherche du miminum d'une liste de nombres
✔ Recherche du minimum et de l'indice d'un minimum

2.1. Mise à zéro des valeurs d'une listes inférieures à un seuil

```
def seuil(L, x): # L est une liste d'entiers, x est un entier
                 # la fonction seuil met à zéro tous les éléments de L qui sont␣
 ↪inférieurs à x
    n=len(L)
    for k in range(n):
        if L[k] < x:
            L[k] =0  # mise à zéro du k-ième élément de L
L0 = [2, 5, 3, 7, 1, 8]
seuil(L0, 5)
print(L0)
```

```
[0, 5, 0, 7, 0, 8]
```

2.2. Recherche du minimum d'une liste de nombres

```
def getMin(L): # renvoie le minimum d'une liste de nombres
    res = L[0] # on initialise avec la valeur du 1er élément
    for el in L[1:]: # sous liste créée en retirant le premier élément
        if (el < res): # el est un candidat pour être le minimum
            res = el
    return res # toute la liste a été parcourue
```

```
L0 = [2, 4, 2.1, 1.2, 54., 8, 1.4, 2.01]
print(getMin(L0))
```

```
1.2
```

2.3. Recherche du minimum et de l'indice d'une liste de nombres

```
def getMin2(L): # renvoie le tuple (min(L), iMin) de la liste L
    iRes = 0   # indice du minimum initialisé à zéro
    res = L[0] # valeur du minimum initialisé à la valeur du 1er élément
    n = len(L) # nombre d'éléments de L
    for k in range(1,n): # on parcourt les éléments à partir du 2ème
                         # c'est-à-dire celui dont l'indice vaut 1
        if L[k] < res:
            iRes = k    # mise à jour de l'indice
            res = L[k] # mise à jour du résultat
    return (res,iRes) # rq : les parenthèses sont optionnelles
L0 = [2, 4, 2.1, 1.2, 54., 8, 1.4, 2.01]
minimum,indice = getMin2(L0)
print('Le minimum de L0 vaut \t\t', minimum); # \t = caractère de tabulation
print("L'indice du minimum de L0 vaut \t", indice)
```

```
Le minimum de L0 vaut           1.2
L'indice du minimum de L0 vaut  3
```

3. Algorithmes de tris : introduction

Nous verrons que certains algorithmes sont plus efficaces si les données sont déjà triées. Par exemple, la recherche d'un élément dans une liste quelconque possède un coût linéaire. Lorsque la liste est déjà triée, on peut utiliser une recherche dichotomique (que l'on verra plus tard) qui possède un coût logarithmique beaucoup plus faible en temps de calcul.

Hypothèse : les données à trier sont stockées dans une liste de nombres appelée TABLEAU. On verra plus tard dans le semestre les tableaux mais pour l'instant un tableau est une liste ne contenant que des éléments de même typage (`float` ou `int`). Par exemples `L=[1,3,56,34]` est un tableau car il ne contient que des valeurs de même type, ici des `int`.

Les principes algorithmes de TRI que vous verrez cette année sont :

- ✔ ***Tri par SELECTION***
- ✔ ***Tri par INSERTION***

} Complexité quadratique $O(n^2)$

- ✔ ***Tri RAPIDE (QUICK)***
- ✔ ***Tri par FUSION (MERGE)***

} Complexité pseudo linéaire $O(n\log n)$, bien plus rapide !!

La notation $O(n^2)$ (big-oh en anglais) signifie que le temps de calcul est de l'ordre de grandeur du carré de la taille du tableau (n éléments). Nous allons développer avec vous au semestre 1 les méthodes par sélection et insertion, puis au semestre 2 vous verrez les 2 autres.

Travail sur les listes - TRI

4. Algorithme de tri par SELECTION

Principe : on souhaite trier par ordre croissant une liste de n éléments. Pour ce tri, on utilise l'algorithme de recherche de l'indice du minimum dans une liste vu précédemment. Voici un exemple, avec ce lien web, sur une liste d'entier du fonctionnement de l'algorithme :

https://fr.wikipedia.org/wiki/Tri_par_s%C3%A9lection

Programme Python d'un tri simple avec 2 listes:

```
def tri_selection(L) :
    L2 = L.copy()
    Lsort = []
    while len(L2) > 0 :
        indice_min_L2 = 0
        for j in range(len(L2)) :
            if L2[j] < L2[indice_min_L2] :
                indice_min_L2 = j

        Lsort.append(L2[indice_min_L2])
        del L2[indice_min_L2]
    return Lsort

L=eval(input("entrer un tableau d'entiers: "))


Lsort=tri_selection(L)
print("la liste triée est: ",Lsort)
```

```
>>>
entrer un tableau d'entiers: [2,7,1,2]
la liste triée est:  [1, 2, 2, 7]
>>>
```

Programme Python d'un tri en place:

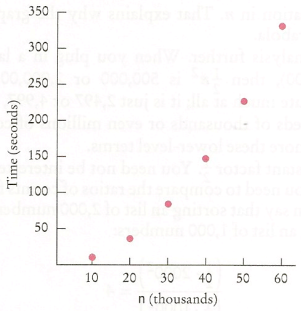
```
def tri_selection_en_place(L):
    n=len(L)
    compteur=0
    for k in range(n-1):
        imini=k
        for i in range(k+1,n):
            compteur+=1
            if L[i]<L[imini]:
                imini=i
        L[imini],L[k]=L[k],L[imini]
    return L,compteur


L=eval(input("entrer un tableau d'entiers: "))


L,compteur=tri_selection_en_place(L)
print("la liste triée est: ",L)
print("le nombre de comparaisons est de: ",compteur)

#print(tri_selection(L))
```

```
>>>
entrer un tableau d'entiers: [2,7,1,2]
la liste triée est:  [1, 2, 2, 7]
le nombre de comparaisons est de:  6
```

Travail sur les listes - TRI

Remarques : Le tri s'effectue en place. Le nombre de comparaison est bien $\frac{n(n-1)}{2}$. C'est une récurrence. Ici $\frac{4\times 3}{2} = 6$ … OK

Cet algorithme peut trier n'importe quelle tableau de nombres (entiers ou réels). Si la vitesse d'execution n'est pas un problème pour vous, vous pouvez vous arrêter ici, cette algorithme fera très bien le travail. Le problème est que cette algorithlme a de piètres qualités en terme de temps de calcul et cela devient critique quand il faut trier de tableaux de taille importantes. Le temps de calcul est proportionnel au carré du nombre d'éléments du tableau comme le montre le graphe ci-contre. C'est pour cette raison que les informaticiens et les mathématiciens ont cherchés des algorithmes moins gourmants en temps de calcul.



n	Seconds
10,000	9
20,000	38
30,000	85
40,000	147
50,000	228
60,000	332

Figure 1 Time Taken by Selection Sort

5. Algorithme de tri par INSERTION

Le tri par insertion est utilisé dans les jeux de carte par beaucoup de personnes pour trier une main.

Le tri par insertion considère chaque élément du tableau et ***l'insère à la bonne place parmi les éléments déjà triés***. Ainsi, au moment où on considère un élément, les éléments qui le précèdent sont déjà triés, tandis que les éléments qui le suivent ne sont pas encore triés. Pour trouver la place où insérer un élément parmi les précédents, il faut le comparer à ces derniers, et les décaler afin de libérer une place où effectuer l'insertion. Le décalage occupe la place laissée libre par l'élément considéré. En pratique, ces deux actions s'effectuent en une passe, qui consiste à faire « remonter » l'élément au fur et à mesure jusqu'à rencontrer un élément plus petit.

https://fr.wikipedia.org/wiki/Tri_par_insertion

Travail sur les listes - TRI

Programme Python d'un tri simple avec 2 listes:

```
def tri_insertion(L) :
    L2 = L.copy()  # ou L2=L1[:]  c'est copie vraie

    for i in range(1,len(L)) :
        for j in range(i) :
            if L2[i]<L2[j] :
                L2.insert(j,L2[i])
                del L2[i+1] # i+1 car il a été décalé à droite par l'insertion


    return L2

L=eval(input("entrer un tableau d'entiers: "))


Lsort=tri_insertion(L)
print("la liste triée est: ",Lsort)
```

```
>>>
entrer un tableau d'entiers: [2,1,7,2]
la liste triée est:  [1, 2, 2, 7]
```

Programme Python d'un tri en place :

```
def tri_insertion_en_place(L):
    compteur=0
    for i in range(1,len(L)) :
        j = i
        while L[j] < L[j-1] and j > 0 :
            compteur+=1
            L[j] , L[j-1] = L[j-1], L[j]
            j -= 1
    return L,compteur


L=eval(input("entrer un tableau d'entiers: "))


L,compteur=tri_insertion_en_place(L)
print("la liste triée est: ",L)
print("le nombre de comparaisons est de: ",compteur)
```

```
>>>
entrer un tableau d'entiers: [2,7,1,2]
la liste triée est:  [1, 2, 2, 7]
le nombre de comparaisons est de:  3
```

Remarque : pour un petit tableau on réduit beaucoup le nombre de comparaisons, donc le temps d'execution !!