TP 16 - Tris de listes

➜ Pour gérer une liste d'entiers aléatoires on peut écrire :

```
import numpy as np
import numpy.random as rd

n=rd.randint(10,21)
L=list(rd.randint(21,size=n))
```

- n est la taille de la liste (aléatorie entre 10 et 20)
- L est une liste d'entiers entre 0 et 20

Attention ! Les paramètres d'entrée d'une fonction sont des variables locales lorsqu'elles sont d'un type *simple* (`int`, `float`, ...). Dans le cas d'une liste ou d'une tableau, les modifications de ses coefficients restent effectives à la fin de la fonction.

⇒ Lors de cette séance, on n'utilisera aucune fonction déjà programmée sur les listes (`sort`, `insert`, ...)

➜ Notion de **complexité temporelle** : dans ce TP, nous allons dénombrer le nombre de tests utilisés afin de réaliser un tri. Cet outils nous permet de comparer l'efficacité des différents tris.

- Le résultat final sera souvent exprimé dans un premier temps par un équivalent puis dans un deuxième temps par un *grand O* : $\mathcal{O}(\ )$.
- La situation retenue pour exprimer cette quantité sera celle du cas le pire : la liste initiale est dans la configuration la plus défavorable pour le tri considéré.

Tri par sélection

Prog

Méthode

➜ Le tri par sélection consiste à :
- on cherche le minimum de la liste et, par échange, on le place à la première place
- puis on place le minimum des éléments restants à la second place
- on recommence, ainsi de suite, jusqu'à ce que les éléments restants soient réduits à un élément.

▶ Version itérative vue au TP 3
▶ Version récursive vue au TP 9

Exercice 1 Considérons la version itérative du tri par sélection.

1. Écrire la variante proposée ici du tri par sélection. On utilisera la fonction `pmin(L,a,b)` suivante qui renvoie la position d'un élément minimal de la liste `L` entre les positions a et b-1.

```
def pmin(L,a,b):
    v,p=L[a],a
    for i in range(a+1,b):
        if L[i]<v:
            v,p=L[i],i
    return(p)
```

2. Compléter le suivi de variables dans le tableau suivant :

i	a	b	pmin(L,a,b)	L
X	X	X	X	[3,-1,1,4,1]
0				
1				
⋮				

3. Dénombrer le nombre de tests lors de l'appel de l'instruction `pmin(L,a,b)`.
4. En déduire la complexité du tri par sélection sur une liste de longueur n.
5. Proposer un invariant de boucle.

Tri par comparaison

Méthode

➜ Le tri par comparaison consiste à :
- pour chaque élément, on parcoure la liste et détermine sa position dans la liste triée : la position de `L[i]` est le nombre d'éléments $j \neq i$ tels que :
  - `L[j]<=L[i]` lorsque $j < i$,
  - `L[j]<L[i]` lorsque $j > i$,
- puis on place chaque élément à sa place dans la liste triée.

▶ Version itérative vue au TP 3

Exercice 2 Donner la complexité temporelle du tri par comparaison.

Tri à identifier

Prog

Exercice 3

1. Détailler les étapes du tri suivant appliqué à la liste `[2,5,0,1,4]` :

```
def tri(L):
    n=len(L)
    for i in range(1,n):
        j=i
        aux=L[i]
        while j>0 and L[j-1]>aux:
            L[j]=L[j-1]
            j=j-1
        L[j]=aux
```


Lycée Saint-Exupery – maffre.saintexupery@gmail.com – MPSI

i	j	aux	L
1	1	?	
			?
2	2	?	
	1		?
	0		?
			?
3			
4			

2. Proposer une description et un nom à ce tri.
3. Proposer un invariant de boucle.
4. Étudier la complexité temporelle.
5. Proposer une amélioration de cet algorithme et évaluer son impact sur la complexité.
6. Donner le script de cette amélioration. ❄❄❄

Tri rapide ou *quicksort* (récursif) — Prog

Méthode

➜ Le tri rapide consiste à :
- si la liste contient au plus un élément, alors on retourne la liste ; sinon, on considère le premier élément ;
- on crée deux sous-listes, une contenant les éléments qui lui sont inférieurs et l'autre pour les autres éléments (ceux qui lui sont strictement supérieurs) ;
- on applique le tri aux deux sous-listes obtenues et on concatène le résultat.

Exercice 4

1. Compléter la suite des appels récursifs et les étapes de la remontée :

Profondeur	Liste
0	[2,-1,3,4,0,3,1,-2,-1]
1	[-1,0,1,-2,-1],[2],[3,4,3]
2	...
3	...
2	...
1	[-2,-1,-1,0,1],[2],[3,3,4]
0	[-2,-1,-1,0,1,2,3,3,4]

2. Écrire un script récursifs de ce tri : `tri_rapide(L)`.
3. Étudier sa complexité temporelle.

Tri fusion — Prog

Exercice 5 Créer une fonction `fusion(L1,L2)` qui prend en entrées deux listes **triées** et retourne la liste *réunion* triée des deux listes.
Par exemple : Si `L1=[0,2,5]` et `L2=[2,4]` alors le résultat est `[0,2,2,4,5]`.

⇛ Vous pouvez créer une version itérative ou récursive.

Méthode

➜ Le tri fusion consiste à :
- si la liste contient au plus un élément, alors on retourne la liste ; sinon, on la découpe en deux sous-listes de tailles comparables ;
- on applique le tri sur chacune d'elle et on fusion les résultats

Exercice 6

1. Compléter la suite des étapes de découpe, puis celles de fusions :

Profondeur	Liste
0	[2,-1,3,4,0,3,1,-2,-1]
1	[2,-1,3,4],[0,3,1,-2,-1]
2	...
3	...
4	...
3	...
2	...
1	[-1,2,3,4],[-2,-1,0,1,3]
0	[-2,-1,-1,0,1,2,3,3,4]

2. Proposer un script récursif de la fonction `tri_fusion(L)`.
3. Étudier la complexité temporelle.

Tri à bulles

Méthode

➜ Le tri a bulles consiste à :
- on parcourt la liste de gauche à droite ;
- dès que deux éléments consécutifs sont mal ordonnés, on les échange ;
- on recommence le parcours jusqu'à ce que la liste soit triée.

Exercice 7 *Tri à bulles*

1. Compléter le tableau décrivant chaque échange d'élément du tri :

Parcours	Liste
1	[3,1,0,8,5,2,2,2]
	[1,3,0,8,5,2,2,2]
	...
2	...
⋮	...

2. Compléter le script de la fonction `tri_bulles(L)` :

```
def tri_bulles(L):
    n=len(L)
    test=True
    while test:
        ...
        for i in range(n-1):
            if ...
                L[i],L[i+1],test=...
    return(L)
```

A quoi correspond la variable `test` ?
3. Étudier la complexité temporelle.