I Tri en python

1.1 Tri simple

```
# L contient des données ''comparables'' entre elles
L.sort()
```

Exercice 1
Compléter la fonction `mediane(L)` qui prend en argument une liste de nombres triée et retourne la médiane de ces nombres. Si `L` est de longueur paire, on utilisera la moyenne arithmétique des deux éléments centraux.

Exercice 2
Remplir la fonction `recherche_dicho_rec` qui implémente l'algorithme de recherche dichotomique dans une liste triée de manière récursive.
Rappelons que l'idée maîtresse est de représenter la demi-liste dans laquelle on va poursuivre le recherche sous forme de deux indices : celui du début et celui de la fin de la sous-liste considérée. En récursif, cette idée se traduit par la donnée et le passage de deux arguments supplémentaires.

1.2 Données multiples

Exercice 3
On se place dans un cas plus réaliste : la fonction `liste_alea(n)` retourne une liste de n couples `(nom, note)` où les noms sont classés par ordre alphabétique.
On souhaite calculer la médiane des notes d'une telle liste. Tester la méthode `sort` sur une telle liste.
Le problème vient du fait que, par défaut python compare les couples (et plus généralement les tuples) élément par élément (un peu comme l'ordre alphabétique). Or nous voulons trier par rapport à la deuxième valeur. Heureusement pour nous, tout est prévu :

```
L.sort(key=mafonction)
```

où `mafonction` est une fonction python qu'il faut définir au préalable et qui prend comme seul argument un élément de la liste L et retourne la valeur qui doit être utilisée pour le tri.
Compléter la fonction `mediane_eleve`

Exercice 4
Compléter la fonction `tri_colonne`. Le principe est similaire au problème précédent, sauf qu'on passe cette fois l'indice de la colonne qui doit servir de référence au tri.

II Bases de données, approche en python

2.1 Représentation en python

Une table d'une base de données sera représentée en python par une liste de tuples (ou une liste de listes). Chaque tuple représente une ligne de cette table et tous les tuples seront donc de même longueur.
On ne se préoccupera pas ici des noms des champs de chaque table. Les données seront repérées par l'indice de leurs lignes/colonnes.
Par exemple la table `notes`

id	nom	note
1	eleve1	12
2	eleve2	8
3	eleve1	17

sera représentée en python par

```
NOTES = [
(1, ''eleve1'', 12),
(2, ''eleve2'', 8),
(3, ''eleve1'', 17)
]
```

2.2 Rappels sur les jointures

La syntaxe suivante :

```
SELECT (ici des noms de colonnes)
FROM table1
JOIN table2
```

retourne la table qui est le produit cartésien des deux tables `table1` et `table2` (en ne créant que les colonnes voulues, ce qui ne nous préoccupera pas pour l'instant). C'est à dire que pour chaque ligne de `table1`, on la concatène avec chaque ligne de `table2` pour créer une ligne de la table jointure.
Si on considère la table `devoirs`

id	nom	noteId
1	devoir1	1
2	devoir1	2
3	devoir2	3

alors la jointure de ces deux tables est une table composée de 6 colonnes et 9 lignes :

notes.id	notes.nom	notes.note	devoirs.id	devoirs.nom	devoirs.noteId
1	eleve1	12	1	devoir1	1
1	eleve1	12	2	devoir1	2
1	eleve1	12	3	devoir2	3
2	eleve2	8	1	devoir1	1
2	eleve2	8	2	devoir1	2
...	...	...	...	...	...

Exercice 5
Compléter la fonction `produit` qui réalise cette opération. Donner en fonction de l_1 et l_2 (les nombres de lignes respectifs des deux tables) le nombre de lignes de la table produit.

Exercice 6
Le fichier *donnees.py* contient la représentation python d'une base de donnée (concernant des résultats de bac) contenant trois tables :

1. academies
   - id
   - nom
2. departements
   - id (c'est une chaîne de caractères)
   - nom
   - academieId
3. lycees
   - id
   - nom
   - departementId
   - tauxDeReussite
   - ville

```
from donnees import ACADEMIES, DEPARTEMENTS, LYCEES
```

Tester le code précédent sur les tables `academies` et `departements`. Combien de lignes contiendrait la table produit de ces 3 tables ? Ne pas tester ! Par contre il s'agit de la table à considérer pour faire, par exemple la moyenne des taux de réussites dans l'académie de Rouen.
Evaluer la taille en mémoire de la table produit obtenue. On se servira de la fonction `taille_tuple`

2.3 Jointure avec condition

En pratique, une jointure s'écrit

```
SELECT (ici des noms de colonnes)
FROM table1
JOIN table2
ON table1.c1 = table2.c2
```

où c1 et c2 sont des noms de colonnes dans `table1` et `table2` respectivement. La table construite ne contient alors que les lignes dans lesquelles les valeurs de `table1.c1` et `table1.c2` sont égales. Par convention on ne répète pas la colonne des valeurs égales.

Exercice 7
Compléter la fonction `jointure`. Avec les notations de l'exercice précédent, combien de tests d'égalité entre les valeurs des colonnes devons-nous effectuer ?
Tester cette fonction sur les tables `academies` et `departements`, en imposant l'égalité des colonnes `academies.id` et `departements.academieId`. On doit obtenir autant de lignes qu'il y a de département et 4 colonnes.
Joindre cette table à `lycees` et évaluer la taille de la table obtenue. Comparer à la taille (que vous estimerez) de la table produit (sans créer celle-ci, encore une fois)

Exercice 8
On considère maintenant que les tables sont triées suivant les valeurs des colonnes égales (avant d'effectuer la jointure, le faire dans la console avant si nécessaire).
Implémenter un algorithme qui effectue la jointure plus efficacement sous cette hypothèse dans la fonction `jointure_tri`.

2.4 Les autres parties d'une requête

Exercice 9
Compléter la fonction `filtre` qui prend une table, un indice i et une valeur v en entrée et qui retourne la table composée des lignes dont la valeur à l'indice i est v.
Cette fonction nous permet d'implémenter la clause WHERE.

Exercice 10
Même chose pour la fonction `projection` pour la clause SELECT

Exercice 11
Obtenir en python la liste des taux de réussites des lycées de l'académie de "Rouen".
En SQL, on pourrait utiliser

```
SELECT lycees.tauxDeReussite
FROM lycees
JOIN departements on departement.id = lycees.departementId
JOIN academies on academies.id = departements.academieId
WHERE academies.nom = ''Rouen''
```

Calculer ensuite la moyenne, la médiane de ces taux et obtenir les lycées au dessus et les lycées au dessous de la médiane.