TP2 : Listes, récursivité, tris

1 Listes, dictionnaires

Exercice TP2.1 *Recherche dans une liste ou un tableau*

1. Écrire une fonction `maximum` qui prend en entrée une liste `L`, puis renvoie le plus grand élément de `L` (on n'utilisera pas la fonction `max` de Python).
2. Écrire une fonction `appartient` qui prend en argument une liste `L` et un élément `x` et qui renvoie `True` si cet élément appartient à `L` et `False` sinon. On utilisera impérativement une boucle conditionnelle.

Exercice TP2.2 *Dictionnaires*

Pour les questions suivantes, on pourra tester avec :

1. Écrire une fonction `occurences(L)` prenant en entrée une liste et renvoyant un dictionnaire dont les clés sont les éléments de `L` et les valeurs le nombre de fois où chaque élément apparaît dans `L`. Par exemple `occurences([1, 2, 8, 1, 5, 6, 4, 1, 0, 2, 1, 6, 0])` renvoie le dictionnaire qui s'exprime `{1: 4, 2: 2, 8: 1, 5: 1, 6: 2, 4: 1, 0: 2}`.
2. En utilisant la fonction précédente, écrire une fonction `sansdoublon(L)` prenant en entrée une liste et renvoyant une liste contenant les mêmes éléments, mais une seule fois.
3. Écrire une fonction `doublons(L)` prenant en entrée une liste et renvoyant une liste, contenant tous les éléments de `L` qui apparaissent au moins deux fois (l'ordre est à votre convenance). Les doublons initiaux apparaîtront une seule fois dans le résultat.
4. Écrire une fonction `majoritaire(L)` renvoyant l'élément d'une liste (supposée non vide) qui apparaît le plus souvent (un des éléments en cas d'égalité).

2 Récursivité

Exercice TP2.3 *Récursivité : PGCD*

Soient A et B deux entiers naturels non nuls. La division euclidienne de A par B définit le quotient Q et le reste R. Les diviseurs communs de A et B sont les diviseurs communs de B et R. Ainsi

$$PGCD(A, B) = PGCD(B, R).$$

C'est cette propriété qui permet d'établir l'algorithme d'Euclide.

1. Programmer l'algorithme d'Euclide en écrivant une fonction `PGCD(a,b)` qui rend le « plus grand commun diviseur » des deux entiers naturels non nuls `a` et `b`. On donnera une version itérative et une version récursive de cette fonction.
2. Écrire une fonction `decompte(a,b)` qui renvoie le nombre d'étapes nécessaires pour calculer le PGCD des nombres a et b.

Exercice TP2.4 *Récursivité : persistance*

Un entier naturel n étant donné, on calcule le produit prod(n) de ses chiffres dans son écriture décimale, puis le produit des chiffres de prod(n) dans son écriture décimale et on recommence ainsi jusqu'à obtenir un nombre entre 0 et 9. Le nombre minimal d'application de la fonction prod pour obtenir ce chiffre est appelé la **persistance** de n. Par exemple, la persistance de 9 est 0, celle de 97 est 3 et celle de 9575 est 5.

1. Donner une fonction `persistance(n)` qui recevant un entier rend sa persistance. On donnera une version récursive.
2. Quel est le plus petit entier de persistance 5 ?

3 Tris

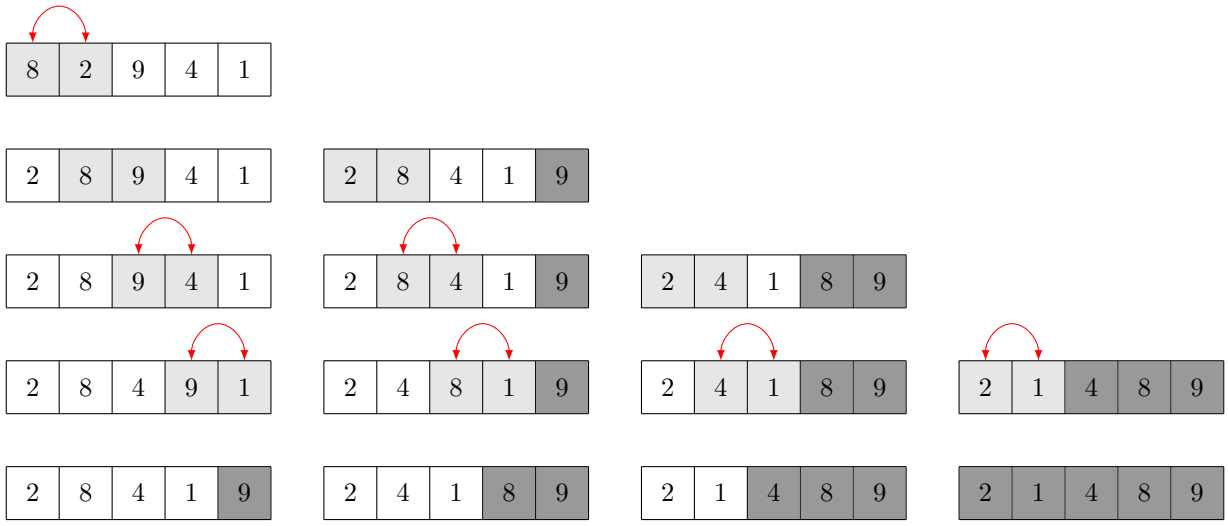
Exercice TP2.5 *Tri par paquets*

On suppose que tous les éléments de la liste `L` à trier sont dans l'intervalle $I = [a, b[$. Le tri par paquets consiste à découper l'intervalle I en `len(L)` sous-intervalles de même longueur puis à répartir les données en paquets correspondant à chacun de ces sous-intervalles. On triera alors chacun de ces paquets à l'aide d'un des algorithmes de tri implémentés en cours.

1. Si on note I_k le k-ième sous-intervalle de I, comment parvenir à déterminer dans quel intervalle se situe le réel $x \in [a, b[$?
2. Écrire une fonction `tri_paquets(L)` implémentant l'algorithme présenté ci-dessus.
3. Donner la complexité de l'algorithme `tri_paquets(L)`.

Exercice TP2.6 *Tri à bulles*

Le tri à bulles est un algorithme de tri. On considère un tableau de nombres. L'algorithme parcourt le tableau, et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. Après un premier passage, on voit que le plus grand élément se situe bien en fin de tableau (*cf.* figure ci-dessous). On peut donc recommencer un tel passage, en s'arrêtant à l'avant-dernier élément, et ainsi de suite. Au i-ème passage on fait remonter le i-ème plus grand élément du tableau à sa position définitive, un peu à la manière de bulles qu'on ferait remonter à la surface d'un liquide, d'où le nom d'algorithme de tri à bulles.



(a) $i = 1$ (b) $i = 2$ (c) $i = 3$ (d) $i = 4$

Sur la figure ci-dessus, les cases gris clair représentent les éléments comparés, les flèches rouges les échanges d'éléments, et les case gris sombre les éléments placés définitivement.

1. Écrire une procédure Python `tri_bulles` prenant en argument un tableau `T` et qui trie les éléments de `T` triés dans l'ordre croissant.
2. Écrire une fonction Python `liste_alea` prenant en argument deux entiers `n` et `N` et qui renvoie une liste aléatoire de nombres de $[\![1, N]\!]$ de longueur n.
3. Tracer le graphe donnant le temps d'exécution de `tri_bulles` en fonction de n pour n variant entre 1 et 100. On pourra prendre $N = 1000000$ et utiliser le package `time`.