TP 6 : tris de listes et récursivité

Algorithmes de tris de listes

Un algorithme est une suite finie d'instructions écrite dans le but de résoudre un problème donné, qui en prenant en entrée un nombre fini de données renvoie après un nombre fini d'opérations un résultat sous la forme d'un autre nombre fini de données.

On présente les algorithmes sous plusieurs formes : diagrammes de flot (voir TP 3 page 2), dans un langage de programmation quelconque (comme Python), ou simplement par une suite d'instructions textuelles comme ci-dessous.

On présente ici deux algorithmes simples de tri de liste.

Algorithme : tri par sélection *Données d'entrée :* une liste
- On recherche le plus grand élément de la liste
- On le met à la fin de la liste
- On cherche le plus grand élément de la liste sans son dernier élément
- On le place en avant dernière position
- On cherche le plus grand élément de la liste sans ses deux derniers éléments
- On le place en avant-avant dernière position
- Etc...

Sortie : la liste triée

Voici un exemple d'exécution de cet algorithme sur la liste $[4, 7, 1, 3, 19, 7, 2, 1]$
(en gras on montre la partie triée de la liste au fur et à mesure qu'elle se constitue) :

Étape	État de la liste	Maximum de la partie non triée
0	[4,7,1,3,19,7,2,1]	19
1	[4,7,1,3,7,2,1,**19**]	7
2	[4,1,3,7,2,1,**7,19**]	7
3	[4,1,3,2,1,**7,7,19**]	4
4	[1,3,2,1,**4,7,7,19**]	3
5	[1,2,1,**3,4,7,7,19**]	2
6	[1,1,**2,3,4,7,7,19**]	1
7	[1,**1,2,3,4,7,7,19**]	1
8	[**1,1,2,3,4,7,7,19**]	-

Exercice 1.

1. Voici un code qui effectue ce tri :

```
def maxi(l, n):
    sous_liste = maxi[ : n]
    return sous_liste.index(max(sous_liste))
def tri_selection(liste):
    n = len(liste)
    for k in range(n, 1, -1):
        ind = maxi(liste, k)
        liste[ind], liste[k-1] = liste[k-1], liste[ind]
    return liste
```

La commande `maxi(liste, n)` renvoie la position du maximum des n premiers éléments de `liste`.
La commande `tri_selection(liste)` ne renvoie rien mais après exécution, la liste est triée.
Saisissez ce code **dans l'éditeur de texte** puis testez le sur une liste de votre choix.

2. Dans l'interpréteur, entrez la commande `from random import *`. Vous pouvez désormais générer des réels aléatoirement entre 0 et 1 à l'aide de la commande `random()`. Utilisez cette commande pour remplir une liste aléatoire contenant 10000 réels à l'aide de
   `liste = [random() for k in range(10000)]`. Puis testez votre tri sur cette liste. Comparez la vitesse de votre tri avec celle de la méthode `sort()` en faisant `print(liste.sort())`.

Voici maintenant une autre version du tri d'une liste : le tri par insertion.

Tri par insertion
- On laisse le premier élément de la liste à sa place
- On place le deuxième élément de la liste à gauche ou à droite du premier selon si il est plus petit ou plus grand que le premier
- On recommence avec le troisième élément, inséré à la bonne place : position 1 ou 2 ou 3 selon sa valeur
- Etc...

Exercice 2.

1. Écrire une fonction `insertion(liste, n)` qui prend en argument une liste `liste` dont on suppose les `n` premiers éléments déjà triés et qui insère `liste[n]` à sa place parmi les `n` premiers éléments de `liste`.
   Indication : on pourra pour ce faire échanger `liste[n]` avec son voisin de gauche tant qu'il est strictement plus grand que ce voisin de gauche.
2. Écrire une fonction `tri_insertion` qui réalise le tri par insertion. Testez le comme dans l'exercice précédent et comparez à nouveau avec la méthode `sort()`.

Pour rivaliser de vitesse avec la méthode de Python, il va nous falloir utiliser un algorithme plus puissant.

Fonctions récursives et l'approche diviser pour régner

Une fonction **récursive** est une fonction qui s'appelle elle-même. Par exemple on a $0! = 1$ et

$$\forall n \in \mathbb{N}^* \ , \ n! = (n-1)! \times n \ .$$

Autrement dit si on a déjà calculé factorielle $n-1$, il suffit de multiplier cela par n pour obtenir $n!$.
On en déduit le petit code Python suivant qui calcule la factorielle d'un entier :

```
def factorielle(n):
    if n == 0:
        return 1
    else:
        return n*factorielle(n-1)
```

Exercice 3.

1. Testez ce programme en calculant les factorielles des 100 premiers entiers.
2. Donnez à votre professeur le nombre de chiffres de 100!. On utilisera `len(str(p))` pour trouver le nombre de chiffres du nombre entier `p`.
3. Testez le programme suivant, légèrement modifié, avec $n = 6$ et méditez sur l'ordre des affichages.

```
def factorielle(n):
    if n == 0:
        resultat=1
    else:
        print('-'*n+'> appel de factorielle ', n-1)
        resultat = n * factorielle(n-1)
        print('-'*n+'> sortie de factorielle ', n-1)
    return resultat
```

La récursivité est à la base d'un type d'algorithmes appelé « diviser pour régner ». Le principe est de découper les données d'entrée du problème en deux paquets distincts et d'appliquer la procédure souhaitée à chaque paquet séparément. La procédure s'appelle alors elle-même et découpe donc à nouveau chacun de ces deux paquets de données en deux paquets, et ainsi de suite.... Voici un exemple très simple : l'exponentiation rapide.

Étant donné un réel a quelconque et un entier naturel n, on a

$$a^n = \begin{cases} \left(a^{\frac{n}{2}}\right)^2 & \text{si } n \text{ est pair} \\ a \times \left(a^{\frac{n-1}{2}}\right)^2 & \text{si } n \text{ est impair} \end{cases}$$

On en déduit un algorithme d'exponentiation de type « diviser pour régner ».

```
def expo(a,n):
    if n == 0:
        return
    else:
        if .........:
            return
        else:
            return
```

Un algorithme de tri rapide

Le tri rapide (quicksort en anglais) est une des méthodes de tri les plus rapides. L'idée est de choisir un élément dans la liste (disons le premier élément) puis de le mettre définitivement à sa place en mettant (sans les trier) à sa gauche tous les éléments inférieurs à ce nombre et à sa droite tous les éléments supérieurs. On recommence ensuite le tri sur les deux sous-listes obtenues.

Par exemple avec la liste $[10, 1, 5, 19, 3, 3, 2, 17]$:
— on commence par choisir le pivot 10
— on constitue les deux sous-listes `liste1 = [1, 5, 3, 3, 2]` et `liste2 = [19, 17]`.
— on appelle récursivement le tri sur ces deux sous-listes
— on concatène les deux listes avec le pivot avec l'opérateur « + » : `liste1+[10]+liste2` permet de recoller les listes entre elles, en ajoutant le pivot choisi au milieu
L'algorithme est alors

```
def tri_rapide(liste):
    if liste == [] :
        return []
    else:
        liste1 = [x for x in liste[1:] if x < liste[0]]
        liste2 = [x for x in liste[1:] if x >= liste[0]]
        return tri_rapide(liste1)+[liste[0]]+tri_rapide(liste2)
```

Exercice 4.

1. Testez cet algorithme au **PAPIER CRAYON** sur la liste `[-8, 5, 1, 7, -10, -12, -15, 3, -9]`
2. Codez cet algorithme. Vérifiez votre code sur la liste précédente.
3. Faites à nouveau le comparatif entre ce tri et celui donné par `sort()` sur une liste aléatoire de taille 100000.