Feuille de cours 9 (informatique) : tris

Ce cours présente deux algorithmes de tris, i.e. deux méthodes permettant de ranger dans l'ordre croissant une liste de nombres réels. On peut bien sûr adapter ces algorithmes pour ranger la liste dans l'ordre décroissant (à faire en exercice et à vérifier sur l'ordinateur !).

On présente uniquement des versions "en place" des algorithmes. Cela signifie qu'on va travailler directement dans la liste prise en argument : on modifiera cette liste pour la trier et la renvoyer. Pour cela, on aura besoin de faire des *échanges* dans la liste.

Rappel : Comment faire pour échanger les éléments d'indices p et q d'une liste `L` ?

1 Tri par sélection

Rappel : écrire une fonction `ind_min` renvoyant l'indice du plus petit élément d'une liste de nombres.

L'algorithme du tri par sélection propose de trier une liste `L` en sélectionnant successivement ses plus petits éléments pour les placer au début. Il consiste à :

- trouver le minimum de la liste `L`
- l'échanger avec le premier élément de `L` c'est-à-dire avec `L[0]`
- trouver ensuite le deuxième plus petit élément de `L` : c'est en fait le minimum de la sous-liste de `L` ne contenant que les éléments d'indices supérieurs à 1
- l'échanger avec le deuxième élément de `L` c'est-à-dire avec `L[1]`
- continuer jusqu'à avoir traité tous les éléments de `L` !

Commençons par un exemple. Que contient successivement la liste `L` à chaque étape de cet algorithme pour `L` $= [6, 3, 1, 5, 4, 8, 7, 2]$?

Écrivons maintenant une fonction `tri_select` prenant en argument une liste de nombres `L` et la triant de sorte qu'à la fin de l'exécution de `tri_select(L)`, `L` contienne la liste triée.

Si la liste `L` comporte n éléments, combien d'opérations "élémentaires" (i.e. ici de comparaisons entre deux éléments de la liste) doit-on faire en tout pour obtenir la liste triée ?

2 Tri par insertion

Le principe du tri par insertion est de trier progressivement une liste en commençant par les éléments apparaissant en premier dans la liste (et non pas en commençant par les plus petits comme dans le tri par sélection).

Au fur et à mesure de l'algorithme, on *insère* tour à tour les éléments de la liste à leur place parmi les éléments les plus à gauche de la liste, qui sont eux déjà triés. C'est souvent de cette manière que l'on trie sa main de cartes.

Commençons par un exemple en décrivant ce que contient successivement la liste `L` lors de son tri par insertion si initialement `L = [6, 5, 3, 1, 8, 7, 2, 4]`.

Décrivons maintenant l'algorithme en Français :

- **Première étape**
  La sous-liste constituée du premier élément de L est triée (elle ne contient qu'un élément). On considère alors $L[1]$ et on le met à sa place dans la sous-liste triée. Pour cela, on compare $L[0]$ et $L[1]$.
  — Si $L[0] > L[1]$, on permute $L[0]$ et $L[1]$.
  — Sinon, on les laisse à leur place.
  Au terme de la première étape, la sous-liste $[L[0], L[1]]$ constituée des deux premiers éléments de L est triée.

- **Deuxième étape**
  On considère le premier élément de la partie non triée de la liste L, c'est-à-dire $L[2]$. On le met à sa place dans la sous-liste triée (qui est pour l'instant $[L[0], L[1]]$).
  Pour cela, on parcourt la sous-liste triée jusqu'à arriver à un élément plus petit que $L[2]$ et on insère l'élément $L[2]$ à cet endroit :
  — Si $L[2] > L[1]$, alors on laissse $L[2]$ en troisième position.
  — Si $L[1] > L[2] > L[0]$ alors on met $L[2]$ en deuxième position.
  — Si $L[1] > L[0] > L[2]$ alors on met $L[2]$ en première position.
  Au terme de la deuxième étape, la liste $[L[0], L[1], L[2]]$ des trois premiers éléments de L est triée.

- On itère n fois (avec n le nombre d'éléments dans la liste) afin de trier la totalité de la liste. À la fin de l'étape numéro k, les k premiers éléments de la liste sont dans l'ordre croissant. Lors de la k-ème étape, pour insérer l'élément $L[k]$ à la bonne place, il suffit – puisque les $k-1$ premiers éléments de la liste sont déjà placés par ordre croissant – de trouver le plus petit indice j tel que $L[j] > L[k] > L[j-1]$. On décale alors les éléments $L[j], L[j+1], \ldots, L[k-1]$ vers la droite et on place $L[k]$ à la place de $L[j]$.

Pour l'implémentation, notons `n` la longueur de la liste `L`. La k-ème étape de l'algorithme vise à placer l'élément `x = L[k]` à la bonne place. On commence à $k = 1$ puisque la sous-liste ne contenant que `L[0]` est triée. Ainsi on utilise la boucle for suivante :

Pour trouver le bon emplacement `j` pour `x`, on commence par supposer que `x` est déjà à la bonne place, c'est-à-dire que `j =`
On fait ensuite décroître `j` (c'est-à-dire) jusqu'à ce que l'élément à sa gauche (c'est-à-dire `L[ ]`) soit plus petit que `x`. Il faut donc faire décroitre `j` tant que
Enfin, il faut également s'assurer de ne pas sortir de la liste, i.e. que

En effectuant la recherche du bon emplacement `j`, on décale les éléments rencontrés qui doivent se placer in fine à droite de `x` donc on effectue l'affectation :
Enfin, une fois le bon emplacement `j` trouvé, on peut y placer l'élément `x`, donc on effectue l'affectation

Finalement, le code Python est le suivant :

Si la liste `L` comporte n éléments, combien d'opérations "élémentaires" (i.e. ici de comparaisons entre deux éléments de la liste) doit-on faire en tout pour obtenir la liste triée ?

3 Exercices

Exercice 1

1. Écrire une fonction `ind_max` renvoyant l'indice du maximum d'une liste de nombres.
2. Utiliser cette fonction pour écrire une fonction `tri_select_2` réalisant le tri d'une liste de nombres `L`, toujours dans l'ordre croissant, mais de la manière suivante :
   — sélectionner le plus grand élément de `L` et l'échanger avec le dernier élément de `L`,
   — sélectionner le deuxième plus grand élément de `L` et l'échanger avec l'avant-dernier élément de `L`,
   — etc.

   On commencera par décrire ce que doit contenir successivement la liste `L` lors de l'exécution de l'algorithme si initialement `L` vaut `[5,1,15,2,10,8,3]`. De plus, afin d'utiliser la fonction de la question 1, on pensera à utiliser des sous-listes.

Exercice 2

Écrire une fonction `tri_insert_2` réalisant le tri *dans l'ordre décroissant* d'une liste de nombres `L` en suivant une méthode similaire à celle du tri par insertion.