TRIS

L'objectif de ce TP est d'étudier différents programmes pour trier une liste, c'est-à-dire d'obtenir les éléments rangés dans l'ordre croissant.

Commençons par quelques points de vocabulaire :

- La **clé du tri** est la relation d'ordre total sur les éléments de la liste à trier. Par exemple, si chaque élément de la liste correspond aux informations sur une personne, on pourrait utiliser comme clé de tri l'ordre alphabétique sur les noms de famille. Dans le sujet, On ne considérera que des exemples de listes d'entiers, sur lesquelles la clé de tri est l'ordre usuel.
- Un **tri en place** modifie directement la liste qu'il prend en argument, sans avoir besoin de créer de nouvelle liste dans laquelle stocker le résultat ou un calcul intermédiaire.
- Un **tri stable** conserve dans le même ordre que dans la liste initiale les éléments qui ont la même clé. Ainsi dans l'exemple des noms de famille, un tri stable préserve l'ordre des personnes qui auraient le même nom de famille.
- un **tri comparatif** se base sur des comparaisons entre les clés des éléments à trier.

1 Tri par sélection

On considère une liste `L` contenant n éléments. Le tri par sélection consiste à :

- rechercher le plus petit élément de la liste et le placer en première position ;
- rechercher le deuxième plus petit élément de la liste et le placer en deuxième position ;
- répéter ce procédé jusqu'à ce que tous les éléments aient été placés.

1. Écrire une fonction `indice_minimum` qui prend en argument une liste `L` (non supposée triée) et un indice `i`, et qui renvoie l'indice $j \geq i$ tel que `L[j]` soit minimal.

   Par exemple, `indice_minimum([2, 4, 8, 10, 5, 7], 2)` doit renvoyer 4, puisqu'on cherche le minimum à partir de l'indice 2, qui est donc 5, dont l'indice est 4.
2. En déduire une procédure `tri_selection` qui prend en argument une liste et qui la trie.

   On rappelle que l'instruction `a,b = b,a` permet d'échanger la valeur de `a` et `b` (y compris si ce sont des cases de liste).
3. Quelles sont les caractéristiques (en place, stable, comparatif ?) du tri par sélection ?
4. Évaluer le nombre maximal de comparaisons effectuées pour trier une liste de taille n.

2 Tri par insertion

Le tri par insertion est celui habituellement utilisé pour trier ses cartes. Il consiste à insérer successivement chaque élément dans la partie triée de la liste. On détaille l'algorithme sur un exemple :
On considère la liste non triée `L = [8, 3, 5, 9, 2]` comprenant $n = 5$ éléments. On parcourt la liste du deuxième élément au dernier élément. Lorsqu'on est à l'étape k (k variant entre 1 et $n-1$) les éléments entre les indices 0 et $k-1$ sont déjà triés, il faut donc insérer cet élément d'indice k dans la partie triée :

- Liste non triée : `L = [8, 3, 5, 9, 2]`
- $k = 1$, `L = [8, 3, 5, 9, 2]` $\longrightarrow$ `L = [3, 8, 5, 9, 2]`;
- $k = 2$, `L = [3, 8, 5, 9, 2]` $\longrightarrow$ `L = [3, 5, 8, 9, 2]`;
- $k = 3$, `L = [3, 5, 8, 9, 2]` $\longrightarrow$ `L = [3, 5, 8, 9, 2]`;
- $k = 4$, `L = [3, 5, 8, 9, 2]` $\longrightarrow$ `L = [2, 3, 5, 8, 9]`;

Chaque étape d'insertion nécessite une boucle pour intervertir l'élément à insérer avec son prédécesseur autant de fois que nécessaire.

1. Écrire une fonction `tri_insertion` qui prend en argument une liste `L` et et la trie selon la méthode décrite dans l'exemple ci-dessus.
2. Donner les caractéristiques du tri par insertion.
3. Donner le nombre maximal de comparaisons effectuées pour une liste de longueur n.

3 Tri par partition-fusion

Le tri fusion est un algorithme récursif basé sur la principe de « diviser pour régner » :

- Si la liste est vide ou admet un seul élément alors elle est triée ;
- sinon, on divise la liste en deux moitiés, que l'on trie récursivement puis qu'on fusionne en les interclassant, de sorte à obtenir une version triée de la liste de départ.

1. Écrire une fonction `fusion` qui prend en argument deux listes `L1` et `L2` **supposées triées** et retourne la fusion triée des deux listes. Cette fonction devra être de coût linéaire en la somme des longueurs des listes. En particulier, on ne pourra pas utiliser une fonction de tri préalablement écrite.

   Indication : Stocker en mémoire les deux indices `i1` et `i2` correspondant au parcours de chaque liste, et à chaque étape ajouter la plus petite des deux valeurs `L1[i1]` et `L2[i2]` au résultat.
2. Écrire une fonction récursive `tri_fusion` implémentant le tri fusion.
3. Donner les caractéristiques de l'algorithme.

4 Tri rapide

Le tri rapide est également un algorithme récursif basé sur la principe de « diviser pour régner » :

- Si la liste est vide ou admet un seul élément alors elle est triée ;
- sinon :
  - on définit le **pivot** p comme le dernier élément de la liste.
  - On divise la liste en 3 : les éléments strictement inférieurs à p, ceux égaux à p, et ceux strictement supérieurs. On trie récursivement la première et la dernière liste, et on concatène les résultats.

1. Appliquer à la main l'algorithme à la liste `L = [10, 3, 5, 6, 8, 12, 4, 7]`.
2. Écrire une fonction `repartition` prenant en argument une liste et renvoyant les 3 listes décrites ci-dessus.
3. En déduire une implémentation du tri rapide.
4. Avec notre choix de pivot, que se passe-t-il si la liste est triée ? Est-ce qu'un autre choix de pivot serait préférable ?

5 Tri par comptage

Le tri par comptage est un algorithme de tri d'entiers. On considère une liste `L` contenant n entiers entre 0 et p, par exemple $p = 20$ et `L = [10, 20, 12, 12, 16, 16, 12, 20, 15]`. On relève le nombre d'occurrences dans `L` de chaque entier entre 0 et p dans une liste `D`, puis on remplit une autre liste `L1`, la liste triée, que l'on construit à partir de `D`.

1. Écrire une fonction `tri_comptage` qui prend en argument un entier p, une liste `L` d'entiers entre 0 et p, et qui renvoie une nouvelle liste, correspondant à la liste `L` triée.
2. Quelles sont les caractéristiques du tri par comptage ?

6 Tri rapide en place

On souhaite à présent obtenir une version en place du tri rapide.

1. Écrire une fonction `repartition_en_place` prenant en argument une liste `L` ainsi que deux indices `a` et `b`, qui utilise `L[b]` comme pivot et permute les éléments de `L` entre ces indices de façon à ce qu'à gauche du pivot ne se trouve que des éléments qui lui sont inférieur, et à sa droite que des éléments qui lui sont strictement supérieurs. De plus la fonction renverra l'indice final du pivot.
2. Écrire une procédure `tri_sous_liste` prenant en argument une liste `L` et deux indices `a` et `b`, et qui trie en place selon la méthode du tri rapide la partie de `L` entre les indices `a` et `b` inclus.
3. En déduire une procédure `tri_rapide_en_place` prenant en argument une liste `L` et la triant par un tri rapide en place.

7 Comparaison des temps de calcul

1. Importer les bibliothèques `timeit`, `matplotlib.pyplot` et `random`.
2. Écrire une fonction `mesure_temps` prenant en argument 4 entiers k, n, p, d, et, pour chaque i entre 1 et n, détermine le temps de calcul pour chaque fonction de tri écrite pour trier k listes contenant id entiers tirés aléatoirement entre 0 et p. La fonction affichera ces temps de calcul en fonction de la longueur de la liste pour chaque algorithme de tri.

   On pourra utiliser les fonctions `randint` de `random`, `default_timer` de `timeit`, `legend` et `plot` (avec son argument optionnel `label`) de `matplotlib.pyplot`.
3. Tracer ces courbes pour $k = 100, n = 100, p = 1000, d = 2$.