Un deuxième algorithme de tri

Dans la première partie de ce TP, nous allons aborder le tri par sélection. Nous nous interrogerons sur sa complexité temporelle et ferons son implémentation en Python.

Dans la deuxième partie, nous comparerons le tri par sélection et le tri par insertion. Une ouverture sur d'autres algorithmes de tri sera ensuite proposée.

Les exercices 1, 3 et 5 sont à faire dans un même fichier que vous appellerez `tris.py` et qui nous servira dans d'autres TP.

Les exercices 2 et 4 sont à faire sur feuille ou dans un fichier texte.

. LE TRI PAR SÉLECTION

Tri par sélection

Regarder la vidéo suivante pour une première approche du tri par sélection.

On peut écrire l'algorithme de la manière suivante, on remarque à nouveau qu'il y a 2 boucles (bleue et rouge) imbriquées.

```
1  On dispose d'une tab de nombres de longueur L
2  Pour i allant de 0 à (L-1)
3      initialiser index_min à i
4      pour j allant de (index_min+1) à L
5          si tab[j]<tab[index_min]
6              index_min = j
7      échanger tab[i] et tab[index_min]
```

Si vous souhaitez voir l'évolution des indices des boucles de l'algorithme ci-dessus, ce lien vous amène sur une page où vous pouvez faire avancer l'algorithme pas à pas. (Attendez peut-être quelques secondes pour pouvoir observer).

- Dans la partie gauche, utilisez le bouton next pour faire avancer votre algorithme pas à pas.
- Dans la partie droite, suivez le résultat de l'algorithme.

Soyez particulièrement attentifs à l'évolution du tableau à trier (en jaune) et à l'évolution des valeurs des itérateurs (i et j) et de la valeur stockée (élément) dans la partie bleue.

Encore une autre façon de voir ce tri avec des danses roumaines

. EXERCICE 1

1. Appliquer manuellement le tri par sélection au tableau [8,6,1,9,7,2] en notant sur votre cahier l'état du tableau à la fin de chaque passage par la boucle bleue.
2. Écrire la fonction de `tri_selection` dans un fichier que vous appellerez `tris.py`
3. Compléter le code suivant pour l'appliquer au tableau précédent.

```
# Question 3 : Ne s'exécutera que si ce module est le programme principal
# (donc pas s'il est importé dans un autre)
if __name__ == '__main__':
    # A COMPLETER
```

. EXERCICE 2

Nous allons nous pencher sur l'aspect théorique de cet algorithme de tri.

1. Quel est la complexité de l'algorithme de tri par sélection (dans le pire cas) ?
   Aide : Relire la partie du cours sur la complexité.
2. Expliquer pourquoi on est sûr de la terminaison de la boucle rouge.
   Aide : Relire la partie du cours sur la terminaison.
3. Expliquer pourquoi on est sûr de la terminaison de la boucle bleue.
4. Conclure sur la terminaison générale de l'algorithme de ce tri.

. COMPARAISON DES ALGORITHMES DE TRI

Un petit questionnaire pour voir si vous avez bien saisi les principes des 2 tris.

. EXERCICE 3 : TRI PAR SÉLECTION / TRI PAR INSERTION

Nous allons maintenant évaluer expérimentalement l'efficacité des deux algorithmes de tri pour vérifier si nos observations sont cohérentes avec les complexités temporelles que nous avons calculées.

1. Créer une fonction `liste_decroissante` qui prenne en paramètre un nombre n et qui renvoie un tableau de taille n contenant les n premiers entiers classés par ordre décroissant.
2. Faire afficher la courbe de la durée d'exécution de la fonction `tri_selection` en lui passant comme paramètre la liste renvoyée par la fonction `liste_decroissante` en faisant variant n de 100 à 1000 *(en 20 étapes).*
   Remarque : Les tableaux rangés par ordre décroissant correspondent « au pire cas » pour trier par ordre croissant.
   Aide : Relire les informations sur le module timeit données dans l'exercice 5 du TP 1.
3. La courbe obtenue est-elle cohérente avec la complexité temporelle calculées à l'exercice 2 ?
4. Créer une fonction `liste_aleatoire` qui prenne en paramètre un nombre n et qui renvoie un tableau de taille n contenant les n premiers entiers classés par ordre décroissant.
5. Créer une fonction `liste_croissante` qui prenne en paramètre un nombre n et qui renvoie un tableau de taille n contenant les n premiers entiers classés par ordre croissant.
6. Faire afficher, dans un même repère, en faisant variant n de 10 à 1000 *(en 20 étapes)* :
   - la courbe de la durée d'exécution de la fonction `tri_selection` en leur passant comme paramètre la liste renvoyée par la fonction `liste_aleatoire`
   - la courbe de la durée d'exécution de la fonction `tri_selection` en leur passant cette-fois comme paramètre la liste renvoyée par la fonction `liste_croissante`
   - la courbe de la durée d'exécution de la fonction `tri_insertion` en leur passant comme paramètre la liste renvoyée par la fonction `liste_aleatoire`
   - la courbe de la durée d'exécution de la fonction `tri_insertion` en leur passant cette-fois comme paramètre la liste renvoyée par la fonction `liste_croissante`

   Aide : Pensez à importer le module tri_insertion
   Attention : Choisir des légendes compréhensibles pour pouvoir interpréter le graphique
7. Interprétation du graphique : Les courbes des deux fonctions (`tri_selection` et `tri_insertion`) ont-elles la même forme dans les deux situations (`liste_aleatoire` et `liste_croissante`) ? Interpréter les résultats de cette expérimentation.

. EXERCICE 4 : DEUX AUTRES ALGORITHMES DE TRI

En vous aidant de l'article https://interstices.info/les-algorithmes-de-tri/

1. Citer deux autres algorithmes de tri. D'après l'article, leur complexité temporelle est-elle meilleur ou moins bonne que celles du tri par insertion et du tri par sélection ?
2. Parmi les quatre algorithmes de tri présentés dans cet article, lequel utilise la méthode « diviser pour régner » qui sera étudiée au programme de terminale NSI ?
3. Utiliser l'onglet « Tri temporel » de l'animation HTML5/JS présente dans cet article pour effectuer le tri de 20 tableaux de taille 1000 avec chacun des quatre algorithme de tri proposé.
4. Utiliser l'onglet « Journal » de cette animation HTML5/JS pour comparer les temps d'exécution mais aussi le nombre de comparaisons et le nombre de copies des tris effectués à la question 3.

. EXERCICE 5 (POUR LES PLUS RAPIDE)

Implémenter le tri à bulle en Python dans une fonction `tri_bulle`