

TD : Tris et complexité

On estime à plusieurs dizaines de milliards le nombre de pages Internet indexées par les principaux moteurs de recherches. Avec un tel volume d'information, il est essentiel de pouvoir accéder rapidement aux résultats les plus pertinents, et pour cela d'être capable de trier efficacement un grand nombre d'éléments.

L'objectif de ce TD est de vous faire découvrir quelques algorithmes de tri et de vous initier à cette notion d' "efficacité" d'un programme, ce qu'on appelle formellement la complexité.

I. Génération aléatoire

Pour étudier les tris, il est pratique de disposer d'un grand nombre de listes pour faire des tests. Plutôt que de les créer manuellement élément par élément, on préférera disposer d'une fonction pour générer automatiquement des listes dont le contenu sera tiré au hasard.

Question 1. Ecrire une fonction `creerListeAleatoire`, telle que `creerListeAleatoire(n, a, b)` renvoie une liste d'entiers de longueur `n` dont chacun des éléments a été tiré au hasard entre `a` et `b` (inclus).

II. Estimation du temps de calcul

a. La fonction `time.clock`

Pour mesurer le temps qu'une fonction met à s'exécuter, Python dispose d'un outil très pratique : la fonction `time.clock`.

Cette fonction ne nécessite aucun argument : l'instruction `time.clock()` renvoie le nombre de secondes qui se sont écoulées depuis le lancement du programme. La valeur renvoyée est un nombre réel : par exemple, après une milliseconde d'exécution, cette valeur vaut `0.001`.

Pour pouvoir faire appel à cette fonction, il vous suffit d'ajouter la ligne `import time` au début de votre fichier.

Pour mesurer et afficher le temps d'exécution d'une fonction donnée, on peut donc utiliser une structure de la forme suivante :

```
temps1 = time.clock()

# Appel a la fonction a tester

temps2 = time.clock()
tempsDeCalcul = temps2 - temps1
```

Question 2. Imaginer une suite d'instructions qui mette environ un dixième de secondes à s'exécuter, et vérifier que c'est bien le cas grâce à la fonction `time.clock`.

Indice : Le temps d'exécution d'une boucle `for` dépend du nombre de passage dans cette boucle.

b. Premier exemple

Question 3.a. Imaginer une fonction `afficherCarre`, telle que `afficherCarre(n)` affiche la liste des entiers compris entre 1 et n^2 .

Question 3.b. Utiliser la fonction `time.clock()` pour étudier comment le temps de calcul évolue en fonction de n .

Remarque : Attention lors de vos tests : des valeurs trop faibles de n risquent de donner des temps de calcul trop courts pour être pertinents, et des valeurs trop élevées de n aboutiront à des temps de calcul beaucoup trop longs.

III. Tri par sélection

a. Principe

Le tri par sélection repose sur une idée assez simple :

- On trouve le plus grand élément de la liste
- On le met dans la dernière case
- On trouve le plus grand élément de la sous-liste $l[0 \dots n - 2]$ (c'est-à-dire toutes les cases sauf la dernière)
- On le met dans l'avant dernière case
- Etc.

b. Echanger deux éléments

Question 4. Ecrire une fonction `echanger`, telle que `echanger(l, i, j)` échange le contenu des cases d'indices i et j de la liste l .

Exemple : Si l est la liste $[4, 9, 6, 7]$, `echanger(l, 1, 3)` transforme la liste l en $[4, 7, 6, 9]$

c. Trouver le maximum

Question 5. Ecrire une fonction `trouverMaximum`, telle que `trouverMaximum(l)` renvoie la plus grande valeur contenue dans la liste `l`.

Exemple : Si `l1` est la liste `[4, 9, 6, 7]`, `trouverMaximum(l1)` renvoie 9.

Question 6. Ecrivez une fonction `trouverIndiceDuMaximum`, telle que `trouver_indice_du_maximum(l)` renvoie l'indice de la plus grande valeur contenue dans la liste `l`.

Exemple : Si `l1` est la liste `[4, 9, 6, 7]`, `trouverIndiceDuMaximum(l1)` renvoie 1 (car le plus grand élément est dans la case d'indice 1, c'est-à-dire la deuxième).

Remarque : Si cette valeur maximale apparaît plusieurs fois dans la liste, l'indice renvoyé doit être celui de la première occurrence de ce maximum.

Question 7. Ecrivez une fonction `trouverIndiceDuMaximumDansSousListe`, telle que `trouverIndiceDuMaximumDansSousListe(l, iMax)` renvoie l'indice de la plus grande valeur contenue dans la liste `l`, parmi les cases dont l'indice est inférieur ou égal à `iMax`.

Exemple : Si `l2` est la liste `[4, 3, 7, 1, 9, 7]`, `trouverIndiceDuMaximumDansSousListe(l2, 3)` renvoie 2 (car le plus grand élément parmi les cases d'indices inférieur ou égal à 3 est situé dans la case d'indice 2).

Remarque : Si cette valeur maximale apparaît plusieurs fois dans la sous-liste, l'indice renvoyé doit être celui de la première occurrence de ce maximum.

d. Trier

Question 8. En combinant les fonctions des questions précédentes, écrire une fonction `triParSelection`, telle que `triParSelection(l)` trie la liste `l` selon la méthode décrite dans le paragraphe III.a.

Question 9. Utiliser la fonction `time.clock` pour observer comment évolue le temps nécessaire pour trier une liste en fonction de la longueur de celle-ci. D'après vos observations, quelle est la complexité du tri par sélection ?

Indice : Augmenter progressivement la taille des listes à trier lors de vos tests et penser à sauvegarder (les temps de calcul peuvent devenir assez longs avec une liste de grande taille).

IV. Affichage graphique

a. Fichier source

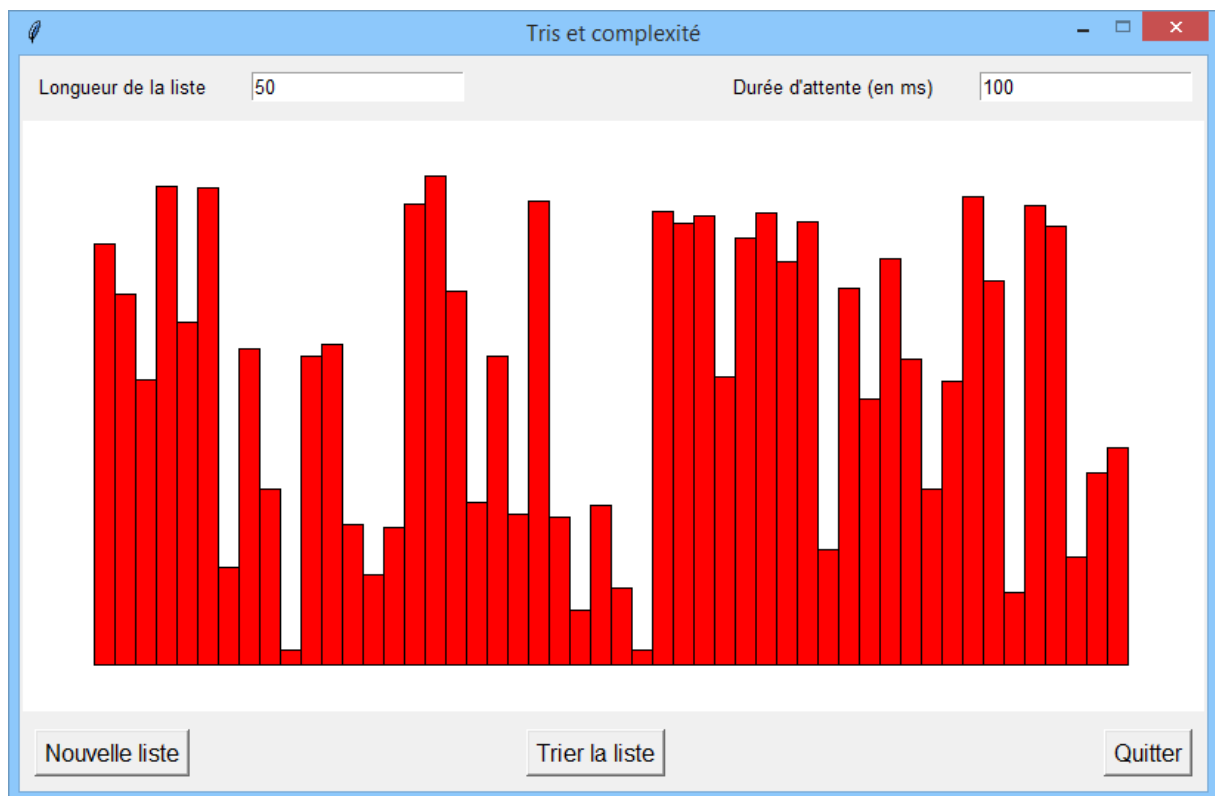
Localisez le fichier source disponible sur le support en ligne du cours et enregistrez-le dans votre répertoire de travail (votre fichier de travail et ce fichier source doivent être au même endroit).

Pour pouvoir utiliser les éléments déclarés dans le fichier source, ajoutez les lignes suivantes au début de votre fichier de travail :

```
import sortGUI
```

b. Une interface graphique

Ce fichier source va vous permettre de visualiser les effets des différents algorithmes de tri grâce à une petite interface graphique :



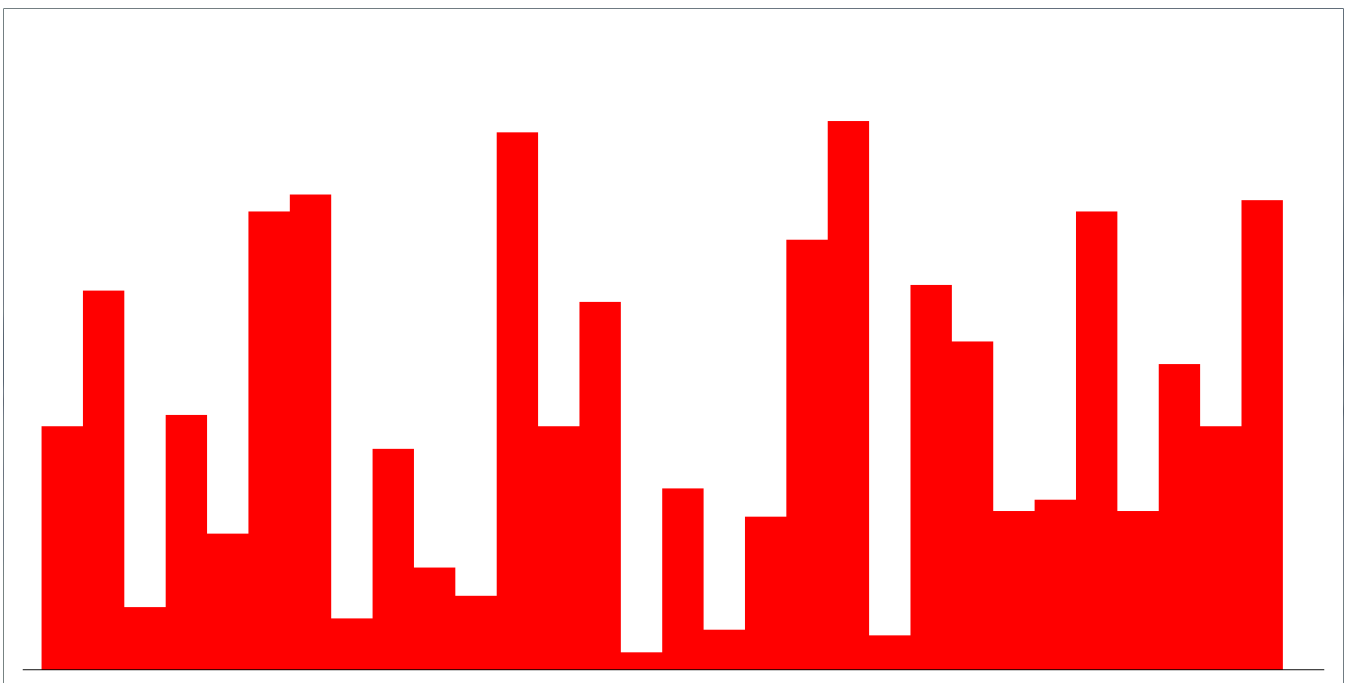
Pour ouvrir cette interface, il vous suffit d'écrire `sortGUI.Window(maMethodeDeTri)`, où `maMethodeDeTri` est le nom d'une méthode de tri (c'est-à-dire une fonction qui prend en argument une liste et qui trie cette liste).

★ **Question 10.** Ouvrir une fenêtre graphique en utilisant votre méthode de tri par sélection.

Cette interface se veut assez intuitive :

- Le champ "Longueur de la liste" permet de changer le nombre d'éléments dans la liste à trier (avec un maximum de 500 éléments)
- Le champ "Durée d'attente (en ms)" indique, en millisecondes, la durée d'attente entre deux rafraîchissements de l'affichage (avec un maximum de 1000)
- Le bouton "Nouvelle liste" permet de générer une nouvelle liste aléatoire
- Le bouton "Trier la liste" permet de trier la liste actuellement affichée
- Le bouton "Quitter" permet de fermer la fenêtre

Comme on peut le voir sur l'exemple ci-dessous, la $i^{\text{ème}}$ case de la liste correspond à la $i^{\text{ème}}$ barre de l'histogramme, qui est d'autant plus haute que la valeur de cette case est importante (ici on a une liste de taille 30 avec des valeurs comprises entre 0 et 100) :



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
43	67	11	45	24	81	84	9	39	18	13	95	43	65	3	32	7	27	76	97	6	68	58	28	30	81	28	54	43	83

Vous remarquerez que votre méthode de tri est "trop rapide" pour l'affichage : la liste est instantanément triée. Pour remédier à ce problème, vous pouvez utiliser la fonction `time.sleep` : l'instruction `time.sleep(n)` force le programme à attendre n secondes (où n est un nombre réel).

★ **Question 11.** Utiliser la fonction `time.sleep` (par exemple avec un délai de 0,2 secondes à chaque échange) pour observer les effets du tri par sélection dans l'interface graphique.

V. Tri à bulles

L'idée de ce tri est de faire "remonter" les éléments les plus grands vers la fin de la liste, un peu comme des bulles vers la surface d'un liquide.

Pour cela,

- On parcourt la liste du début à la fin
- Si on trouve un indice i tel que $l[i] > l[i+1]$, alors on inverse ces deux éléments
- Si on arrive à la fin de la liste sans faire le moindre échange, c'est que la liste est triée
- Sinon on refait un passage
- etc.

Question 12. Ecrire une fonction `triABulles`, telle que `triABulles(l)` trie la liste `l` selon la méthode décrite ci-dessus.

Question 13. Utilisez la fonction `time.clock` pour observer comment évolue le temps nécessaire pour trier une liste en fonction de la longueur de celle-ci. D'après vos observations, quelle est la complexité du tri à bulles ?

★ **Question 14.** Créer une variante de la fonction précédente qui utilise la fonction `time.sleep` pour observer les effets du tri à bulles dans l'interface graphique.

VI. Tri par insertion

a. Principe

Ce tri est aussi appelé "tri du joueur de carte", car il correspond à ce que fait naturellement un joueur de cartes à qui on distribue progressivement des cartes :

- Le joueur a dans sa main des cartes déjà triées
- Il reçoit une nouvelle carte
- Il l'insère au bon endroit dans sa main
- Sa main est à nouveau triée
- Il reçoit une nouvelle carte
- etc.

Pour trier une liste, on peut donc utiliser la méthode suivante :

- Les k premières cases de la liste sont déjà triées (au début, k vaut 1)
- On considère l'élément situé à l'indice k de la liste (soit donc le $k+1^e$ élément)
- On déplace cet élément vers le début de la liste jusqu'à atteindre la position souhaitée
- Les $k+1$ premières cases de la liste sont à nouveau triées
- On considère l'élément situé à l'indice $k+1$ de la liste (soit donc le $k+2^e$ élément)
- etc.

Par exemple, sur le schéma ci-contre,

- Les trois premières cases de la liste (en bleu clair) sont triées
- On considère le 4^e élément (en bleu foncé) : sa valeur est 3
- On déplace ce 3 de droite à gauche jusqu'à la position souhaitée
- Les quatre premières cases de la liste (en bleu clair) sont triées
- On considère le 5^e élément (en bleu foncé) : sa valeur est 7
- etc.

1	4	9	3	7	5
1	4	3	9	7	5
1	3	4	9	7	5
1	3	4	9	7	5

b. Implémentation

Question 15. Ecrire une fonction `deplacerElement`, telle que `deplacerElement(l, k)` modifie la liste `l` (dont on suppose les k premières cases triées) afin de placer l'élément situé à l'indice k à sa place dans la partie triée de la liste.

Question 16. Ecrire une fonction `triParInsertion`, telle que `triParInsertion(l)` trie la liste `l` selon la méthode décrite ci-dessus.

Question 17. Utilisez la fonction `time.clock` pour observer comment évolue le temps nécessaire pour trier une liste en fonction de la longueur de celle-ci. D'après vos observations, quelle est la complexité du tri par insertion ?

★ **Question 18.** Créer une variante de la fonction précédente qui utilise la fonction `time.sleep` pour observer les effets du tri par insertion dans l'interface graphique.

VII. Tri fusion

a. Principe

Le tri fusion est un tri basé sur l'idée *Diviser pour régner* : pour trier une liste de longueur n , on la divise en deux sous-listes de longueur $n/2$, on trie récursivement ces sous-listes, et on fusionne les versions triées de ces sous-listes pour obtenir une version triée de la liste initiale.

Pour des raisons de simplicité, on se concentrera ici sur les listes dont la longueur est une puissance de 2. Pour trier une liste l de longueur 2^k , l'algorithme devient donc :

- Si $k = 1$ (liste de taille 2), il suffit d'échanger si besoin est les deux éléments
- Si $k > 1$ (liste de taille 4 ou plus),
 - On crée deux sous-listes l_1 et l_2 correspondant aux deux moitiés de l
 - On trie récursivement l_1 et l_2
 - On fusionne l_1 et l_2 dans une nouvelle liste l_F
 - On copie le contenu de l_F dans l

b. Implémentation

Question 19. Ecrire une fonction `creerSousListes`, telle que `creerSousListes(l)` renvoie un couple de listes (l_1, l_2) , où l_1 et l_2 sont les deux moitiés de la liste l .

Remarque : Pour récupérer les résultats de cette fonction dans une autre fonction, il suffit d'écrire `(l1, l2) = creerSousListes(l)`

Question 20. Ecrire une fonction `fusionnerListes`

- prenant en argument deux listes d'entiers l_1 et l_2 (supposés triés et de même taille n)
- renvoyant une liste l triée de longueur $2n$, composée des éléments des listes l_1 et l_2

Question 21. Imaginez une fonction `triFusion`, telle que `triFusion l` trie la liste l selon la méthode décrite ci-dessus.

Question 22. Utilisez la fonction `time.clock` pour observer comment évolue le temps nécessaire pour trier une liste en fonction de la longueur de celle-ci. D'après vos observations, quelle est la complexité du tri fusion?