

Tris itératifs

L'objectif de ce TP est de redécouvrir trois algorithmes de tri **itératifs** (sélection, insertion, bulle), de les implémenter en Python, et de comparer expérimentalement leurs temps d'exécution.

Les questions de ce TP sont à traiter dans le fichier `.py` associé, dont chaque cellule est numérotée ; le sujet indique simplement, pour chaque question, le numéro de la cellule concernée.

Mesurer un temps d'exécution

Pour mesurer le temps que met une instruction (ou un bloc d'instructions) à s'exécuter, on utilise la fonction `time` du module `time` (importé sous l'alias `t`). Cette fonction renvoie le nombre de secondes écoulées depuis une date de référence fixe : il suffit donc de relever ce nombre juste avant et juste après un bloc d'instruction à chronométrer, puis de calculer la différence.

```
t0 = t.time()
# instructions
t1 = t.time()
duree = t1 - t0 # temps en secondes
```

1 Générer des listes aléatoires

Question 1. Écrire, dans la cellule **Q1**, une fonction `generateur` qui prend en argument un entier naturel n et qui renvoie une liste de n entiers aléatoirement choisis (de manière uniforme) dans l'ensemble $\llbracket 1, n \rrbracket$.

Il est possible d'écrire la fonction sur deux lignes à l'aide de la définition de listes par compréhension.

2 Le tri par sélection

Le principe du **tri par sélection** est de placer, à la k -ème itération, le k -ème plus petit élément de la séquence à trier en sélectionnant le minimum des valeurs encore non triées. On commence donc par chercher le minimum de la séquence à trier, qu'on place en première position. On cherche ensuite le minimum des valeurs restantes, qu'on place en seconde position, etc.

Question 2. Écrire, dans la cellule **Q2**, une fonction `minimum` qui prend en argument une liste de nombres et qui renvoie la valeur minimale de cette liste.

Question 3. Compléter le code de la cellule **Q3** de manière à écrire une fonction `tri_selection`, qui trie *sur place* une liste de nombres. La fonction ne devra rien renvoyer¹, elle devra simplement modifier la liste passée en argument. On dit que la fonction agit alors par *effet de bord*².

La structure de cette fonction est proche de celle de `minimum` : à chaque itération, on cherche l'indice du minimum de la portion de liste encore non triée. Vous pourrez vous inspirer de `minimum` pour compléter le code, sans toutefois l'appeler dans `tri_selection`.

Question 4. Dans la cellule **Q4**, calculer le temps pour trier par sélection une liste de 10^k nombres (générée aléatoirement via la fonction `generateur` de la question 1) pour $k \in \llbracket 2, 4 \rrbracket$.

3 Le tri par insertion

Le principe de l'algorithme de **tri par insertion** est d'insérer successivement (et à la bonne place) chaque élément parmi la collection d'objets précédemment triés, à la manière d'un joueur de cartes. En pratique :

- Le premier élément du tableau constitue à lui seul une collection déjà triée.
- Pour tout $k \geq 1$, on insère l'élément d'indice k en le comparant avec ceux qui le précèdent (pas nécessairement tous, puisqu'ils ont été préalablement triés) et qu'on décale (vers la droite) s'ils le dépassent.

Question 5. Compléter le code de la cellule **Q5** de manière à implémenter une version *sur place* de l'algorithme du tri par insertion sur une séquence de nombres (de type `list` ou un `array`).

1. En l'absence de `return`, une fonction renvoie par défaut `None`.

2. Il s'agit d'une (mauvaise) traduction de l'expression "*side effect*", mais le terme est resté dans la littérature.

Question 6. Dans la cellule **Q6**, calculer le temps pour trier à l'aide de l'algorithme de tri par insertion une liste de 10^k nombres pour $k \in \llbracket 2, 4 \rrbracket$. Comparer avec les résultats de la question 4.

Question 7. On rappelle qu'une liste triée se prête à une recherche par dichotomie. Dans la cellule **Q7 (bonus)**, écrire une variante `tri_insertion_dicho` qui recherche la position d'insertion de x par dichotomie parmi les éléments déjà triés, plutôt que par un parcours de gauche à droite, puis comparer son temps d'exécution avec celui de `tri_insertion` sur une même liste. Le décalage des éléments pour faire de la place à x reste-t-il nécessaire ? Le gain est-il à la hauteur de ce qu'on peut attendre ?

.....

.....

4 Le tri bulle

Question 8. Le code de la fonction `tri_bulle` est donné dans la cellule **Q8**. Décrire en quelques phrases le principe de cet algorithme.

.....

.....

Question 9. La version codée à la question précédente ne s'arrête pas en cours d'exécution si la liste est triée. Compléter la cellule **Q9** de manière à coder une version à trous de `tri_bulle_ameliore`, qui utilise une variable booléenne (un *drapeau*) pour arrêter l'algorithme lorsque plus aucun échange n'a eu lieu lors d'un passage.

Question 10. Dans la cellule **Q10**, calculer le temps pour trier à l'aide de `tri_bulle` puis de `tri_bulle_ameliore` une liste de 10^k nombres pour $k \in \llbracket 2, 4 \rrbracket$. Comparer entre elles, puis avec les résultats des questions 4 et 6.

5 Une propriété des tris : la stabilité

Un tri est dit **stable** si, lorsque deux éléments ont la même valeur, il conserve leur ordre relatif d'apparition dans la liste d'origine. Pour tester la stabilité d'un algorithme de tri, on peut trier une liste de couples (*valeur*, *indice*) selon leur première composante uniquement, en choisissant volontairement des valeurs dupliquées.

Question 11. Dans la cellule **Q11**, une liste de tests `liste_test` est fournie à cet effet. Adapter `tri_selection`, `tri_insertion` et `tri_bulle` pour qu'ils comparent uniquement la première composante des couples, et tester leur stabilité sur `liste_test`. Lequel de ces trois algorithmes n'est pas stable dans son implémentation la plus naturelle ? Pourquoi, précisément, l'échange systématique de deux éléments égaux pose-t-il problème ?

.....

.....

.....

6 Comparaison des performances

Question 12. On souhaite représenter graphiquement les temps moyens d'exécution des trois algorithmes implémentés dans ce TP. Pour cela, on décide de générer des listes aléatoires de longueur n où n est un entier variant de 100 à 1500 de 100 en 100. Pour chaque valeur de n , et pour chaque algorithme de tri, on générera 10 listes aléatoires de longueur n et on calculera la moyenne des temps pour trier ces listes. Compléter la cellule **Q12** et analyser les résultats.

.....

.....