

Tris récursifs

L'objectif de ce TP est de mettre en œuvre deux algorithmes de tri **récursifs**, le tri rapide et le tri fusion, et de les comparer expérimentalement aux tris itératifs du TP précédent.

Comme au TP précédent, chaque question renvoie au numéro de la cellule à compléter dans le fichier `.py` associé. On mesure toujours le temps d'exécution à l'aide du module `time` (`t.time()` avant et après l'instruction à chronométrer). On réutilisera la fonction `generateur` du TP précédent, rappelée en tête du fichier `.py`.

1 Rappel sur la récursivité

Une **fonction récursive** est une fonction qui contient (au moins) un appel à elle-même dans sa définition. En pratique, à l'instar d'une suite par récurrence, une fonction récursive doit être définie par :

- un ou plusieurs (un nombre fini) cas de base (initialisation) ;
- un ou plusieurs appels à elle-même permettant de toujours se ramener au(x) cas de base (hérédité).

Les suites définies par récurrence fournissent des exemples simples, comme la suite $(n!)_{n \in \mathbb{N}}$ vérifiant :

$$0! = 1 \text{ et } \forall n \in \mathbb{N}^*, n! = n \times (n-1)!.$$

On peut alors implémenter une définition récursive en Python :

```
def factoriel(n):
    if n == 0:
        return 1
    else:
        return n * factoriel(n-1)
```

Question 1. Dans la cellule **Q1**, écrire une fonction récursive `somme_Euler` qui prend en argument deux entiers n et p et qui renvoie la somme $\sum_{k=0}^n k^p$. On convient qu'une somme sur un ensemble vide d'indices est nulle.

Question 2. Dans la cellule **Q2**, écrire une fonction récursive `palindrome` qui renvoie `True` si une séquence (liste ou chaîne de caractères) est un palindrome et `False` dans le cas contraire.

Par exemple, les instructions `palindrome("kayak")` et `palindrome([0,1,2,2,1,0])` devront renvoyer `True`, tandis que `palindrome("maths")` et `palindrome([0,1,2,0])` devront renvoyer `False`.

2 Tri rapide (*quicksort*)

Le tri rapide est un algorithme basé sur le paradigme « *diviser pour régner* » : on partitionne la séquence à trier en deux sous-séquences, on trie **récursivement** chacune d'elles, puis on les recombine pour obtenir la liste triée.

La partition utilisée par le tri rapide repose sur le choix d'une valeur appelée **pivot** : on partitionne la séquence à trier en deux sous-séquences selon que leurs valeurs sont strictement inférieures ou bien supérieures ou égales au pivot. Recombiner les deux sous-séquences une fois triées ne demande alors aucun travail supplémentaire : il suffit de les **concaténer** de part et d'autre du pivot.

Pour des raisons de simplicité, on choisira la première valeur de la liste comme pivot.

Question 3. Dans la cellule **Q3**, écrire une fonction `partition` qui prend en argument une liste `lst` de nombres et qui renvoie un triplet (`pivot`, `inf`, `sup`) où `pivot` est le premier élément de `lst`, `inf` (resp. `sup`) est la liste des valeurs de `lst`, privée de son premier élément, strictement inférieures (resp. supérieures ou égales) à `pivot`.

Question 4. Dans la cellule **Q4**, écrire une fonction récursive `tri_rapide` qui prend en argument une liste de nombres et qui renvoie une version triée de cette liste par l'algorithme du tri rapide.

On ne cherchera pas à écrire un tri en place.

Question 5. Compléter le code de cellule **Q5**, de manière à calculer le temps pour trier à l'aide de l'algorithme de tri rapide une liste de 10^k nombres (générée aléatoirement via `generateur`) pour $k \in \llbracket 4, 6 \rrbracket$.

3 Le tri fusion (*mergesort*)

Le tri fusion est, comme le tri rapide, basé sur le paradigme *diviser pour régner*. Contrairement au tri rapide, qui partitionne selon un pivot, le tri fusion partitionne la liste à trier en deux sous-listes de longueurs qui diffèrent d'au plus une unité, indépendamment des valeurs qu'elles contiennent. C'est alors la **fusion** des deux sous-listes triées – plus coûteuse que la simple concaténation utilisée par le tri rapide – qui fait tout le travail.

Son principe est donc le suivant :

- on partitionne la liste à trier en deux sous-listes de longueurs qui diffèrent d'au plus une unité ;
- on trie **récurivement** les deux sous-listes ;
- on fusionne les deux sous-listes préalablement triées.

Question 6. La cellule **Q6** contient une version à trous d'une fonction `fusion` qui prend en argument deux listes **triées** `lst1` et `lst2` et qui renvoie une liste triée composée de tous les éléments des deux listes `lst1` et `lst2`. Compléter cette cellule.

Question 7. Compléter le code de la cellule **Q7** de manière à écrire une fonction récursive `tri_fusion` qui prend en argument une liste de nombres et qui renvoie une version triée de cette liste par l'algorithme du tri fusion. *On ne cherchera pas à écrire un tri en place.*

Question 8. La fonction `fusion` écrite dans la cellule Q6 n'est pas efficace en raison d'un trop grand nombre d'appels récursifs. La cellule **Q8** contient le code d'une fonction `fusion2`, qui réalise la même tâche que `fusion` mais de manière itérative et plus rapide. Expliquer son principe en quelques phrases, notamment le rôle des variables `i1` et `i2`.

Question 9. Écrire dans la cellule **Q9** une fonction `tri_fusion2` qui utilise `fusion2` à la place de `fusion`.

Question 10. Dans la cellule **Q10**, calculer le temps pour trier par fusion une liste de 10^k nombres (générée aléatoirement via `generateur`), à l'aide de `tri_fusion2` puis calculer le temps pour trier par fusion une liste de 10^3 puis 10^4 nombres (générée aléatoirement via `generateur`), à l'aide de `tri_fusion`. Comparer les temps d'exécution obtenus entre eux ainsi qu'avec ceux du tri rapide (cf. question 5), et commenter.

*En cas de besoin, on pourra utiliser la fonction `setrecursionlimit` du module `sys` afin d'augmenter le nombre maximal d'appels récursifs autorisés (40000 appels récursifs maximum devraient suffire pour le test ci-dessus ; cette instruction est déjà présente dans la cellule **Q10**). Si une erreur de dépassement de la pile d'appels récursifs apparaît, elle vient de `fusion` : cette fonction s'appelle elle-même une fois par élément consommé dans l'une des deux listes, et non une fois par moitié de liste comme le fait `tri_fusion`.*

4 Comparaison des performances

Question 11. On souhaite compléter le graphique du TP précédent avec les deux algorithmes de ce TP. Les fonctions `tri_selection`, `tri_insertion` et `tri_bulle` du TP précédent sont rappelées en tête de la cellule **Q13**. On génère 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 $N = 20$ listes aléatoires de longueur n et on calculera la moyenne des temps pour trier ces listes. Compléter la cellule **Q11** et proposer un classement des cinq algorithmes de tri étudiés du plus rapide au moins rapide.