TP_{06} Performances des tris

L'objectif de ce TP est d'observer la performance des algorithmes de tri vu en cours sur des listes aléatoires.

1. Écrire une fonction `rand_list(n, b)` qui renvoie une liste de longueur `n` contenant des nombres aléatoires compris entre `1` et `b`.

   On pourra utiliser la fonction `randint` *du module* `random`.

2. Écrire une fonction `insertion_sort(L)` (respectivement `merge_sort(L)` et `quick_sort(L)`) triant une liste `L` en utilisant l'algorithme du tri par insertion (respectivement tri fusion et tri rapide).

Pour évaluer la performance d'un algorithme, nous utiliserons la fonction `perf_counter` du module `time`, qui ne prend aucun argument et renvoie le temps écoulé (en secondes) depuis la mise en route de la machine.

Ainsi, les lignes suivantes permettent de calculer le temps d'exécution d'un algorithme de tri appelé `sort` sur une liste `L` (si le module `time` est importé) :

```
start = time.perf_counter()
L = sort(L)
exec_time = 1000 * (time.perf_counter() - start) # en millisecondes
```

3. Définir une liste `L`, aléatoire, de longueur `500` et contenant des nombres entre `1` et `1000`.
4. Observer le temps d'exécution de tri par insertion sur la liste `L`. Exécuter les lignes d'instructions plusieurs fois. Qu'observe-t-on ? Pourquoi ?

On veut maintenant tracer des courbes de performances pour nos trois algorithmes de tri. Nous utiliserons donc le module `matplotlib.pyplot`, que vous penserez à importer.

5. Écrire une fonction `draw_perf(sort)` prenant une fonction de tri en argument et traçant le temps d'exécution de cette fonction (en millisecondes) pour des listes aléatoires de nombres entre 1 et 1000, de taille variant entre 0 et 1000 par pas de 10.
6. Tracer les courbes pour les trois algorithmes de tri.

Il est possible que des valeurs aberrantes apparaissent sur vos courbes : certaines listes aléatoires peuvent correspondre au pire cas de l'algorithme considéré alors que d'autres peuvent correspondre au meilleur cas.

7. Afin de lisser les courbes, modifier la fonction `draw_perf` pour qu'elle effectue la moyenne des temps d'exécutions pour 50 listes de chaque taille.
8. Tracer de nouveau les courbes.

Ces tracés correspondent à la complexité moyenne des algorithmes de tri (qui n'est pas explicitement au programme).

9. Quelles listes permettent d'atteindre le meilleur/pire cas de chaque algorithme de tri ?
10. Tracer des courbes de performances correspondant au meilleur/pire cas de chaque algorithme de tri.
11. Comparer avec les tracés de la complexité moyenne.