Tri de listes

Ce TD propose de programmer différentes méthodes pour ranger une liste de nombres dans l'ordre croissant.

1. On pourra définir une première fonction, $randlist(n, a = 1, b = 1000)$ dont la syntaxe est la suivante :

```
def randliste(n,a=1,b=1000):
    from random import randrange
    return [randrange(a,b) for k in range(n)]
```

   On pourra alors vérifier le bon fonctionnement de celle-ci en testant plusieurs fois $randliste(10)$ qui doit renvoyer une liste de dix nombres aléatoires entre 1 et 999.

2. **Tri par sélection**
   (a) Écrire une deuxième fonction, $mini(l)$ qui prend en argument une liste l et renvoie son plus petit élément.
   (b) Le tri par sélection consiste, partant d'une liste donnée, à déterminer son plus petit élément, le mettre de coté dans une nouvelle liste. Ensuite, on recommence avec les autres éléments de la liste : on détermine le minimum de ceux qui restent et on le met dans la nouvelle liste en 2ème position. Et l'on continue ainsi jusqu'à ce qu'il n'y ait plus d'élément dans la liste initiale. La nouvelle liste que l'on a alors créée est rangée dans l'ordre croissant avec les éléments de la liste initiale.
   Ecrire une fonction $triselect(l)$ qui, à l'aide de la fonction $mini(l)$, réalise le tri dans l'odre croissant de la liste l avec la méthode précédemment décrite.

3. **Tri par insertion**
   (a) Écrire une fonction $insere(nb, l)$ qui prend en argument un nombre nb et une liste l de nombres rangés dans l'ordre croissant et qui renvoie la liste complétée du nombre nb placé à la position permettant que la liste renvoyée soit restée croissante.
   (b) Le tri par insertion se fait en partant d'une liste initiale et d'une liste vide. On vide progressivement les éléments de la liste initiale pour les insérer dans la deuxième liste qui était vide au départ, en prenant soin que la deuxième liste soit toujours rangé dans l'ordre croissant. Quand la liste initiale est vidée, la deuxième liste est le résultat attendu.
   Écrire une fonction $triinsert(l)$ qui prend en argument une liste l et renvoie la liste de ses éléments triés ainsi dans l'ordre croissant.

4. **Tribulle**
   Cette méthode se déroule en $n-1$ étapes pour une liste de longueur n.
   À l'étape $i \in [\![1, n]\!]$, on parcourt les $n-i$ premiers éléments de la liste : si le k-ième élément est plus grand que le $k+1$-ième élément, on les échange. Si à une étape donnée i, on a eu aucun échange à réaliser pour $k \in [\![1, n-i]\!]$, la liste est triée.
   Programmer cette méthode en prenant bien soin de placer une instruction *return* afin que si à la fin d'une étape i donnée, on n'a effectué aucun échange, le programme renvoie immédiatement la liste.

5. **Comparaison des temps de calcul** À l'aide de la fonction $clock()$ du module *time*, écrire une fonction $test(fonc, l)$ qui prend en argument une fonction $fonc$ parmi les trois fonctions de tri précédemment programmées et une liste l, et renvoie le temps d'éxécution de cette fonction appliquée à la liste l. Utiliser cela pour calculer et comparer les différentes fonctions de tri sur des listes aléatoires.