TP COMPLEXITE DES ALGORITHMES DE TRI

OBJECTIFS : L'objectif de ce Tp est d'étudier La performance de ces algorithmes de tris par sélection ou par insertion.

1. Découverte de la bibliothèque *time*

Pour commencer, ouvrir dans Pyzo un fichier nommé « complexite.py » . Importer la fonction *time()* de la bibliothèque *time* :

```
from time import time
```

Dans la console, exécuter cette fonction qui ne prend pas d'argument :

```
>>> time()
1632067575.2129228
```

Le nombre retourné est égal au nombre de secondes écoulées depuis la date appelée « *L'Epoch Unix* » qui est le 1er Janvier 1970 à 00:00. Pourquoi cette date ? L'année 1970 a été considérée comme un bon départ, compte tenu de l'essor qu'a pris l'informatique à partir de cette époque.

Au moment de l'écriture de ce Tp, le 19/09/2021, on obtenait 1 632 067 575 secondes, soit 453 352 heures, soit 18 889 jours, soit 51,75 années.

Une description plus précise de ce module est donnée sur cette page : https://apcpedagogie.com/le-module-time-en-python

Parmi les fonctions proposées, on peut noter : (compléter la liste des fonctions importées)

```
from time import time,ctime,sleep
```

- La fonction *ctime()* qui renvoie :
  - la date actuelle formatée si aucun argument donné : *ctime()* .
    - Qu'obtient-on si on exécute : *ctime(0)* ? :
  - la date formatée correspondant à l'instant défini par le nombre de seconde écoulée depuis le 1er janvier 1970.
    - Qu'obtient-on si on exécute : *ctime(0)* ? :
    - Qu'obtient-on si on exécute : ctime(2 000 000 000) ? :

- La fonction *sleep()* qui permet de réaliser une temporisation.
    - Qu'obtient-on si on exécute : *sleep(5)* ? :

2. Création d'une liste test de nombre aléatoire composée de nombres aléatoires compris entre 0 et 100

- Importer la fonction *randint()* de la bibliothèque random : `from random import randint`
  randint(0,100) permet par exemple de générer un nombre aléatoire compris entre 0 et 100 (essayer dans la console)

- Dans le fichier *complexite.py*, écrire l'instruction qui permet de créer une liste ℓ par compréhension, composée de 10 nombres aléatoires compris entre 0 et 100.

3. Importation des fonctions de tris par sélection et par insertion, écrites dans les séances précédentes

- Télécharger le fichier *bibliTri.py* : https://nsibranly.fr/public/documents/terminale/bibliTri.py et le copier dans le dossier de travail contenant déjà *complexite.py* . Ne pas oublier de cocher la case ☑ Changer le répertoire courant lors de l'exécution d'un fichier dans le menu *Exécuter* de *Pyzo*.

- Importer les fonctions *triSelection()* et *triInsertion()* dans le fichier complexite.py et y écrire ce bout de code qui permet de calculer le temps d'exécution de la fonction *triSelection* pour trier la liste ℓ de taille $n = 10$:

```
from time import time,ctime,sleep
from random import randint
from bibliTri import triSelection , triInsertion

l = [randint(0,100) for i in range(10)]
print(l)
depart = time()
l = triSelection(l)
temps = time() - depart
print(l)
print(f"Temps d'exécution : {temps}")
```

- Exécuter le code pour une liste ℓ de taille $n = 1000$: `Temps d'exécution :`

- Exécuter le code pour une liste ℓ de taille $n = 10\,000$: `Temps d'exécution :`

- Exécuter le code pour une liste ℓ de taille $n = 100\,000$: `Temps d'exécution :`

4. Comparaison des temps de calculs

L'objectif de cette partie est de déterminer les temps de calculs pour trier des listes de taille différentes, pour les algorithmes de tri par sélection, par insertion ou celui de la fonction *sorted()* native de python.

Pour faciliter ce travail, on intègre le code précédent dans une fonction *test()* :

La fonction utilisée est mise en argument

```
from time import time,ctime,sleep
from random import randint
from bibliTri import triSelection , triInsertion

def test(fonction,n) :
    '''
        Arguments :
            fonction : fonction qui retourne une liste triée
            n : taille de la liste d'essai composée de n nombres aléatoires
    '''
    l = [randint(0,100) for i in range(n)]
    if n < 100 : print(l)
    depart = time()
    l = fonction(l)
    temps = time() - depart
    if n < 100 : print(l)
    print(f"Temps d'exécution : {temps}")

# Main
test(triSelection,10)
```

Utiliser ce code pour déterminer les temps de calculs dans les cas suivants. Les case nombre d'opérations ne sont pas à compléter pour l'instant.

- Tri réalisé avec la fonction *triSelection()* :

Taille n de la liste ℓ	100	1000	10 000	100 000	1 000 000
Durée du tri en s					
Nombre d'opérations nécessaires pour trier					

Tri réalisé avec la fonction *triInsertion()* :

Taille n de la liste ℓ	100	1000	10 000	100 000	1 000 000
Durée du tri en s					
Nombre d'opérations nécessaires pour trier					

Tri réalisé avec la fonction *sorted()* :

Taille n de la liste ℓ	100	1000	10 000	100 000	1 000 000
Durée du tri en s					
Nombre d'opérations nécessaires pour trier					

Question : Comment évoluent les temps de calculs en fonction de la taille n de la liste à trier ?

On propose 3 réponses :

- Réponse 1 : Lorsque la taille n est multipliée par 10, les temps de calcul sont multipliés par $\approx$ 10 aussi
- Réponse 2 : Lorsque la taille n est multipliée par 10, les temps de calcul sont multipliés par $\approx$ 100
- Réponse 3 : Autre réponse

Analyser les résultats précédents et proposer une réponse pour chacune des 3 fonctions testées. Justifier de manière chiffrée.