Algorithmes

Travail préalable : créer 2 fonctions ***alea_liste(n)*** *et* ***alea_entier_liste(n)*** *qui retournent chacune une liste de n nombres aléatoires* x_i*, avec* $0 \leq x_i \leq 1$ *dans le premier cas et* x_i *entiers tels que* $0 \leq x_i \leq n$ *dans le second cas*

I- Algorithmes de parcours d'un tableau

Pour les exercices suivants, on utilisera la liste :
test=[19, 16, 28, 26, 50, 16, 0, 36, 1, 45, 38, 27, 37, 16, 45, 41, 3, 19, 49, 43, 19, 44, 40, 23, 33, 25, 30, 38, 28, 49, 31, 37, 8, 48, 34, 12, 25, 6, 37, 23, 32, 36, 44, 45, 36, 29, 17, 11, 32, 27]

1) Écrire une fonction ***indice(liste,valeur)*** qui retourne l'indice de la premiere occurrence de ***valeur*** dans ***liste***.
Exemple : ***indice(test,36)*** doit retourner la valeur 7, car test[7]=36.

2) Écrire une fonction ***maximum(liste)*** qui retourne la plus grande valeur contenue dans ***liste***.
Quelle valeur ***maximum(test)*** doit-elle retourner ?

3) Écrire une fonction ***moyenne(liste)*** qui retourne la moyenne des valeurs contenues dans ***liste***.

II- Evaluer le coût d'un algorithme

Le module **time** contient une fonction ***time_ns()*** retournant le temps écoulé en nanosecondes depuis le 01/01/1970. Il est possible grâce à elle d'évaluer le temps que met une série d'instructions à se réaliser :

1) En combien de temps ***moyenne(test)*** s'est executé ?

2) Mesurer maintenant le temps que met ***moyenne(test)*** pour des listes plus grandes (générées à l'aide alea_entier_liste(n))

n	t (ns)
500	
5000	
50 000	
500 000	
5 000 000	

3) Que constatez-vous ? Les résultats sont-ils toujours les mêmes ?

```
from time import time_ns
def fonction_a_evaluer( ) :
      …
      …
      …
a=time_ns( )
fonction_a_evaluer( )
print("temps
écoulé :",time_ns-a,"ns")
```

III- Algorithme de tri

Il existe plusieurs méthodes de tri :

→ La plus naïve est de chercher le maximum d'une liste, puis une fois celui-ci trouvé, échanger sa position avec l'élément en première position de la liste (tri décroissant) ou dernière position (tri croissant).

En pseudo-code cela donne :

```
procédure tri_selection(tableau t)
    n ← longueur(t)
    pour i de 0 à n - 2
        min ← i
        pour j de i + 1 à n - 1
            si t[j] < t[min], alors min ← j

        si min ≠ i, alors échanger t[i] et t[min]
```

→ on peut aussi prendre consécutivement les éléments de la liste et les remplacer correctement un à un parmi les éléments déjà ordonnés de la liste : c'est le tri par insertion

```
procédure tri_insertion(tableau T)
   n ← taille(T)
   pour i de 1 à n - 1

      # mémoriser T[i] dans x
      x ← T[i]

      # décaler vers la droite les éléments de T[0]..T[i-1] qui
sont plus grands que x en partant de T[i-1]
      j ← i
      tant que j > 0 et T[j - 1] > x
           T[j] ← T[j - 1]
           j ← j - 1

       # placer x dans le "trou" laissé par le décalage
       T[j] ← x
```

→ on peut enfin balayer progressivement la liste en inversant 1 à 1 chaque couple de nombres consécutivement de façon à ce qu'ils soient dans l'ordre de rangement, et cela jusqu'à ce que la liste soit ordonnée : c'est le tri à bulles

```
procédure tri_à_bulles(Tableau T)
   pour i allant de (taille de T)-1 à 1
       pour j allant de 0 à i-1
           si T[j+1] < T[j]
               (T[j+1], T[j]) = (T[j], T[j+1])
```

1) Essayer dans chaque cas de traduire les pseudo-codes en fonctions

2) Essayer de classer en détaillant la méthodologie les algorithmes en fonction de leur rapidité.