Algorithmes et structures de données avancées : TD 1(sur machine)

Listes en Python - Temps d'exécution

Pour intialiser des listes avec `n` élements, vous disposez des deux manières suivantes :

```
l = [ 0  for i in range(n) ]
# ou bien
l = [0] * n
```

Pour chronométrer la durée d'une fonction, vous disposez du code suivant :

```
import time
tempsDebut = time.time()

#ici votre algorithme (affectations, boucles, conditions, appels de fonctions, ...)

tempsFin = time.time()
print 'Temps d exécution : %0.3f s' % ((tempsFin-tempsDebut)*1.0)
```

Exercice 1.1 *Les listes (liste à une dimension)*

1. Créer une liste `liste` avec 10 éléments et afficher la liste.

```
liste = [0] * 10
print liste
```

2. Changer chaque élément de la liste pour qu'il contienne une valeur aléatoire entre 1 et 15 (*Remarque : Utiliser la fonction* ***random.randint(inf, sup)*** *après avoir chargé le module correspondant avec* ***import random***).

```
import random
i=0
while i<10:
    x=random.randint(1,15)
    list[i] = x
    i=i+1
```

3. Afficher la nouvelle liste.

```
print liste
```

4. Ecrire un algorithme qui compte le nombre d'occurences de la valeur 10 dans la liste.

```
nombre=0
i=0
while i<10:
    if list[i]==10:
        nombre=nombre+1
    i=i+1
```

Exercice 1.2 *Temps d'exécution d'algorithmes sur les listes*

Dans cet exercice, nous allons analyser les temps d'exécutions de vos algorithmes pour les listes.
1. Ecrire un programme en python qui demande à l'utilisateur de saisir une valeur numérique qui sera stocké dans la variable `n`.

```
print "Saisir une valeur pour n"
n = input()
```

2. Créer une liste `liste` vide avec n élements.

```
list = [0] * n
```

3. Changer chacun des `n` éléments de la liste pour qu'il contienne une valeur aléatoire entre 1 et 15 (*Remarque : Utiliser la fonction* `random.randint(inf, sup)` *après avoir chargé le module correspondant avec* `import random`).

```
import random
i=0
while i<n:
```

```
    x=random.randint(1,15)
    list[i] = x
    i=i+1
```

4. Ecrire un algorithme qui compte le nombre d'occurences de la valeur 10 dans la liste, et qui affiche le nombre d'occurences à l'écran.

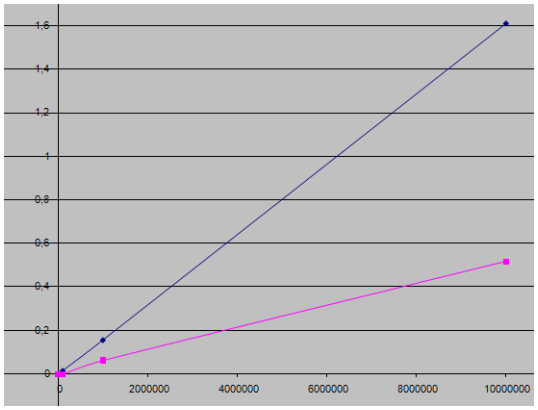
```
nombre=0
i=0
while i<n:
    if list[i]==10:
        nombre=nombre+1
    i=i+1
print "nombre d'elements 10 dans la liste : ",nombre
```

5. Chronométrer le temps d'exécution de votre algorithme à l'aide du code fourni en haut de la page, pour une saisie de valeur pour n=10, n=100, n=1000, n=10000, n=100000, n=1000000, n=10000000, n=100000000. *Remarque : Si votre logiciel ne répond plus après un certain temps, vous pouvez appuyer sur la combinaison de touche CTRL-C.*

```
n=10   #remplacer par la taille souhaitée
tempsDebut = time.time()
nombre=0
i=0
while i<n:
    if list[i]==10:
        nombre=nombre+1
    i=i+1
tempsFin = time.time()
print 'Temps d exécution : %0.3f s' % ((tempsFin-tempsDebut)*1.0)
print "nombre d'elements 10 dans la liste : ",nombre
```

6. Tracer un graphe en fonction de `n` pour le temps d'exécution avec un logiciel de tableur (p.ex. Excel).

Voir la courbe en bleu ci-dessous.



Exercice 1.3 *Temps d'exécution de fonctions prédédinies sue les listes*

1. Dans votre code précédent, remplacer votre algorithme qui compte les occurences de la valeur 10 dans la liste par la fonction prédefini `count`.

```
nombre = list.count(10)
```

2. Chronométrer de nouveau le temps d'exécution pour une saisie de valeur pour n=10, n=100, n=1000, n=10000, n=100000, n=1000000, n=10000000, n=100000000.

```
tempsDebut = time.time()
nombre = list.count(10)
tempsFin = time.time()
print 'Temps d exécution : %0.3f s' % ((tempsFin-tempsDebut)*1.0)
print "nombre d'elements 10 dans la liste : ",nombre
```

3. Tracer un graphe en fonction de `n` pour le temps d'exécution avec un logiciel de tableur (p.ex. Excel).

Voir la courbe en violet ci-dessous.