DS3 d'informatique – corrigé

Exercice 1 : Carré magique

1. (a) Un algorithme permettant de calculer la somme des coefficients d'une ligne est le suivant :

```
def sommeLigne(T,i):
    S=0
    for j in range(len(T[0])):
        S=S+T[i][j]
    return S
```

Cette fonction effectue n itérations de la boucle, sa complexité est donc linéaire : $C(n) = O(n)$

(b) Un algorithme permettant de calculer la somme des coefficients d'une colonne est le suivant :

```
def sommeColonne(T,j):
    S=0
    for i in range(len(T)):
        S=S+T[i][j]
    return S
```

La complexité de cette fonction est encore $O(n)$.

Un algorithme permettant de calculer la somme des coefficients de la diagonale principale est le suivant :

```
def sommeDiag1(T):
    S=0
    for i in range(len(T)):
        S=S+T[i][i]
    return S
```

Complexité $O(n)$

Un algorithme permettant de calculer la somme des coefficients de la deuxième diagonale est le suivant :

```
def sommeDiag2(T):
    S=0
    for i in range(len(T)):
        S=S+T[i][len(T)-1-i]
    return S
```

Complexité $O(n)$. Remarquez bien que l'indice de la dernière colonne est `len(T)-1`.

2. Une fonction `ListeSommes(T)` qui renvoie une liste constituée de toutes les sommes par ligne, par colonne et par diagonale des éléments du tableau `T` est la suivante :

```
def ListeSomme(T):
    liste=[]
    for i in range(len(T)):
        liste.append(sommeLigne(T,i))
        liste.append(sommeColonne(T,i))
    liste.append(sommeDiag1(T))
    liste.append(sommeDiag2(T))
    return liste
```

3. Une fonction `TousEgaux(L)` qui prend en paramètre une liste non vide `L` et qui renvoie `True` si tous les éléments de `L` sont égaux et `False` sinon est la suivante :

```
def TousEgaux(L):
    test=True
    ref=L[0]
    for i in range(1,len(L)):
        if L[i]!=ref:
            test=False
    return Test
```

Complexité $O(n^2)$

4. Une fonction qui renvoie `True` si tous les nombres de 1 à n^2 (où n est lordre du carré) sont présents dans `T`, et `False` sinon est la suivante :

```
def tousPresents(T):
    n=len(T)
    nombres = [False]*n**2
    for i in range(n):
        for j in range(n):
            # Si le nombre n'est pas entre 1 et n², renvoie False
            if T[i][j]>n**2 or T[i][j]<1:
                return False
            # Si on a déjà vu ce nombre
            elif nombres[T[i][j]]==True:
                return False
            # Sinon on le marque comme vu
            else
```

```
            nombres[T[i][j]]=True
    # Si on arrive là, c'est bon.
    return True
```

Complexité $O(n^2)$

5.

```
def magique(T):
    n=len(T)
    Liste=ListeSomme(T)
    if TousEgaux(liste):
        print("carre magique d'ordre", len(T))
        if TousPresents(T):
            print("normal")
        print("et de constante magique", Liste[0])
    else:
        print("carre non magique")
```

Complexité $O(n^2)$.

Exercice 2 : La dichotomie

1.

```
def zeroDichotomie(f,a,b,eps):
  while(b-a>eps):
  c=(a+b)/2
    if f(c)==0:
      return c
    elif f(a)*f(c)<0 :
      b=c
    else:
      a=c
  return (a+b)/2
```

2. À chaque itération la taille de l'intervalle est divisée par 2. Donc après n itérations, la taille de l'intervalle est divisée par 2^n. On a donc $l_n = \frac{b_0-a_0}{2^n}$.
3. L'algorithme s'arrête lorsque $b_n - a_n \leqslant \varepsilon$ donc lorsque $\frac{b_0-a_0}{2^n} \leqslant \varepsilon$ soit $n \geqslant \log_2\left(\frac{b_0-a_0}{\varepsilon}\right)$
4. On utilisera la fonction python suivante :

```
def f(x):
  return sin(cos(x))-x**2
```

5. On peut choisir l'intervalle de départ $[0, \frac{\pi}{2}]$. En effet $f(0) = \sin(1) - 0 > 0$ et $f(\frac{\pi}{2}) = \sin(0) - \left(\frac{\pi}{2}\right)^2 < 0$. Donc f change de signe sur cet intervalle.
Pour cet intervalle de départ, le nombre d'itérations nécessaire sera :

$$n = \log_2\left(\frac{b_0 - a_0}{\varepsilon}\right) = 34$$