Exercice 1

Un tableau X est trié par ordre croissant si x(i) <= x(i+1)pour tout i

1. Elaborer un programme itératif permettant de vérifier qu'un tableau X est trié ou non

```
def Test(X):
    i=0
    while i < len(X)-1 :
        if X[i]>X[i+1]:
            return False
        i=i+1
    return True
```

2. Estimer sa complexité au pire cas et en moyenne, en prenant comme la probabilié que le tableau ne soit pas triée à partir de l'indice k est $\frac{1}{n+1}$ et qu'il le soit avec une probabilité en $\frac{1}{n+1}$

   Le pire cas : on a été au bout du while.

   $C(n) = 1 + \sum_{i=0}^{n-2}(2+2+2) + 2 + 0 = 3 + 6.(n-1) = 6n - 3 = O(n)$

   En moyenne :

   Le tableau n'est plus trié à l'indice k : $C_k = 1 + \sum_{i=0}^{k-1}(2+2+2)+(2+2+0)=6k+5$

   $C_m(n) = \sum_{k=0}^{n-2}\frac{1}{n+1}.C_k + \frac{1}{n+1}C(n) = \sum_{k=0}^{n-2}\frac{1}{n+1}.(6k+5) + \frac{1}{n+1}(6n-3) = \frac{1}{n+1}\left(6\frac{(n-2)(n-1)}{2} + 5(n-1) + 6n - 3\right) = O(n)$

Exercice 2

Pour convertir un nombre entier positif N de la base décimale à la base binaire, il faut opérer par des divisions successives du nombre N par 2. Les restes des divisions constituent la représentation binaire.

1. Ecrire une fonction récursive "Binaire" permettant de donner la liste des nombre binaire.

```
def binaire(N):
    if N<=1:
        return str(N)
    else :
        return binaire(N//2)+str(N%2)
```

2. Donner sa complexité

$C(N) = 1 + 3 + C\left(\left[\frac{N}{2}\right]\right) = 4 + C\left(\left[\frac{N}{2}\right]\right)$

C(0)=C(1)=1

On pose $i = \log_2(N)$, on a $T(i) = 4 + T(i-1)$.

$T(i) = 4i + T(0) = O(i)$

Donc $C(N) = O(\log_2(N))$

Exercice 3

Soit la suite (u_n) définie par : $\begin{cases} u_0 = u_1 = 1 \\ u_{n+2} = 3\,u_{n+1} + u_n \end{cases}$

1. Ecrire un programme récursif permettant de calculer le nième terme de la suite.

```
def suite(n):
    if n<=1:
        return 1
    else:
        return 3*suite(n-1)+suite(n-2)
```

Pour la formule de récurrence toujours se ramener à $u_n = \cdots$ avec n le plus grand indice.

2. Estimer sa complexité.

   $C(0) = C(1) = 1$

   $C(n) = 1 + 4 + C(n-1) + C(n-2) \iff C(n) - C(n-1) - C(n-2) = 5.$

   Equation homogène : $T(n) - T(n-1) - T(n-2) = 0.$

   L'équation caractéristique est $r^2 - r - 1 = 0$ de racines $r_1 = \frac{1+\sqrt{5}}{2}$ et $r_2 = \frac{1-\sqrt{5}}{2}$. donc $T(n) = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n$.

   Solution particulière : $\ell - \ell - \ell = 5 \Leftrightarrow \ell = -5$.

   $C(n) = A\left(\frac{1+\sqrt{5}}{2}\right)^n + B\left(\frac{1-\sqrt{5}}{2}\right)^n - 5 = O\left(\left(\frac{1+\sqrt{5}}{2}\right)^n\right)$

3. Modifer le programme pour avoir une complexité linéaire.

   Posons $v_n = u_{n+1}$, on a $\begin{cases} u_0 = v_0 = 1 \\ u_{n+1} = v_n \\ v_{n+1} = 3\,v_n + u_n \end{cases}$

   La fonction va renvoyer $[u_n, v_n]$

```
def suite2(n):
    if n==1:
        return [1,1]
    else:
        u,v=suite2(n-1)
        return [v,3*v+u]
```

   $C(0) = 1$

   $C(n) = 1 + (2 + \mathrm{C(n\text{-}1)}) + 2 = C(n-1) + 5 = 5\mathrm{n} + \mathrm{C(0)} = \mathrm{O(n)}$

Exercice 4

Etudiez le nombre d'additions réalisées par l'algorithme suivant dans le meilleur cas, le pire cas, puis dans le cas moyen en supposant que les tests ont une probabilité de $\frac{1}{2}$ d'être vrai

```
s=0
for i in range(n):
    if T[i]>a:
        s=s+T[i]
```

Meilleur cas : $C_{\min}(n) = 1 + \sum_{i=0}^{n-1}(1+0) = n+1 = O(n)$ (le meilleur cas est le else qui n'existe pas 0 opérations)

Pire cas : $C_{\max}(n) = 1 + \sum_{i=0}^{n-1}(1+2) = 3n+1 = O(n)$

En moyenne : $C_{\text{moy}}(n) = 1 + \sum_{i=0}^{n-1}\left(1 + \left(\frac{1}{2}0 + \frac{1}{2}2\right)\right) = 2n+1 = O(n)$

Remarque : $C_{\min} = O(n)$ et $C_{\max}$ aussi, on peut écrire $C(n) = \Theta(n)$

Exercice 5

La constante e peut être calculée par la limite de la somme suivante, qui s'appuie sur la factorielle $n! = n.(n-1).(n-2)\ldots 2.1$ (avec $0! = 1$) :

$$e = \lim_{n\to+\infty} \sum_{i=0}^{n} \frac{1}{i!}$$

1. Ecrire une fonction nommée `factorielle` qui prend en paramètre n et renvoie $n!$.

   $u_n = n! \Leftrightarrow \begin{cases} u_0 = 1 \\ u_{n+1} = (n+1)u_n \end{cases} \Leftrightarrow \begin{cases} u_0 = 1 \\ u_n = n.u_{n-1} \end{cases}$

```
def facto(n):
    if n==0:
        return 1
    else:
        return n*facto(n-1)
```

En donner sa complexité.

$C_1(0)=1$ et $C_1(n)=1+2+C_1(n-1)=3n+1=O(n)$

2. Ecrire un programme en python qui demande un entier n à l'utilisateur et affiche $\sum_{i=0}^{n}\frac{1}{i!}$ en utilisant la fonction `factorielle`.

$S_n=\sum_{i=0}^{n}\frac{1}{i!}\Leftrightarrow\begin{cases}S_0=1\\S_{n+1}=S_n+\frac{1}{(n+1)!}\end{cases}$

```
n=int(input('Entrer un entier :'))
S=1
i=0
while i<n:
    S=S+1/facto(i+1)
    i=i+1
print(S)
```

Donner la complexité de votre programme.

$C_2(n)=1+1+1+\sum_{i=0}^{n-1}(1+(4+C_1(i+1))+2)+1=4+\sum_{i=0}^{n-1}(7+3(i+1)+1)=4+11n+3\frac{n(n+1)}{2}=O(n^2)$

3. Ecrire le même programme sans la fonction `factorielle` de complexité $O(n)$.

$\begin{cases}S_0=1\\u_0=1\\u_{n+1}=(n+1)u_n\\S_{n+1}=S_n+\frac{1}{u_{n+1}}\end{cases}$

```
n=int(input('Entrer un entier :'))
S=1
u=1
i=0
```

```
while i<n:
    u=u*(n+1)
    S=S+1/u
    i=i+1
print(S)
```

$C_3(n) = 1+1+1+1+\sum_{i=0}^{n-1}(1+3+3+2)+1 = 5+9n = O(n)$

Exercice 6

Ecrire des programmes récursifs calculant le maximum d'une liste :

1. En considérant que le maximum est le plus grand entre le dernier terme et le maximum des (n-1) premiers termes. Estimer sa complexité.

```
def maxi1(L):
    if len(L)==1:
        return L[0]
    else:
        m=maxi1(L[:-1])
        if m>L[-1]:
            return m
        else:
            return L[-1]
```

$C(1) = 1$

$C(n) = 1+(1+C(n-1))+1 = 3+C(n-1) = 3(n-1)+1 = O(n)$

2. En considérant que le maximum est le plus grand entre les maximums des deux moitiés du tableau. Estimer sa complexité.

```
def maxi2(L):
    if len(L)==1:
        return L[0]
    else:
        m1=maxi2(L[ : (len(L)//2)])
        m2=maxi2(L[(len(L)//2) : ])
        if m1>m2:
            return m1
        else:
            return m2
```

$C(1) = 1$

$C(n) = 1 + \left(2 + C\left(\frac{n}{2}\right)\right) + \left(2 + C\left(\frac{n}{2}\right)\right) + 1 = 5 + 2C\left(\frac{n}{2}\right)$

Pour $i = \log_2(n),\ T(i) = 5 + 2T(i-1)$, donc $T(i) = A2^i - 5 = O(2^i)$

Donc $C(n) = O(2^{\log_2(n)}) = O(n)$