

Exercice 1

On considère le programme suivant:

```
def Algo ( n ) :
    for i in range ( 0 , n ) :
        Algo1 ( n )
    Algo2 ( n )
    Algo3 ( n )
```

- Sachant que **Algo1(n)** s'effectue en temps $O(\log(n))$, **Algo2(n)** en temps $O(n)$ et **Algo3(n)** en temps $O(n^2)$, Quelle est la complexité de **Algo(n)**?

Exercice 2 :

Quelle est la complexité du programme suivant :

```
def Algo ( n ) :
    res = 1 ;
    for i in range ( 0 , n ) :
        res = res + i
    for i in range ( 0 , n ) :
        for j in range ( 0 , i ) :
            res = res * ( i + j )
    return res
```

Exercice 3 :

Quelle est la complexité du programme suivant:

```
def f ( x , n ) :
    if n < 1 :
        return n
    S = 0
    for i in range ( 0 , n ) :
        S += x / ( i + n )
    return S + f ( x , n // 2 )
```

Exercice 4

1- Que vaut la complexité de l'algorithme suivant:

```
i=1
while i <= n :
    i=i*2
```

2- Le code ci-dessous consiste à programmer la fonction *remove*. Analyser sa complexité?

```
def SupprimerElement ( L , a ) :
    i=0
    if a in L :
        while L [ i ] != a :
            i=i+1
        L [ i : i + 1 ] = []
```

Exercice 5

Soient les 3 fonctions suivantes permettant de calculer la valeur d'un polynôme.

$$p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

en un point x (c'est-à-dire pour une valeur x donnée).

Les coefficients du polynômes sont stockés dans une liste : $A=[a_0, a_1, a_2, \dots, a_{n-1}, a_n]$

```
def f1(A,x) :#méthode naïve
    n =len(A)
    p=A[0]
    for i in range(1,n) :
        p = p + A[i]* x**i
    return p
```

$x**i$: n'est pas une opération élémentaire

```
def f2(A,x) :# elle utilise le fait que  $x^i = x^{i-1} * x$ 
    n =len(A)
    p=A[0] ;
    q = 1;
    for i in range(1,n) :
        q = q * x ;
        p = p +A[i]* q ;
    return p
```

#algorithme de Horner ;

il calcule, DANS CET ORDRE, les valeurs de : $a_n, a_nx + a_{n-1}$, puis $(a_nx+a_{n-1})x + a_{n-2}$, etc.

```
def f3(A,x) :# utilisant la méthode de Horner
    n =len(A) -1
    p = A[n] ;
    for i in range(n,0,-1) :
        p = p*x + A[i-1]
    return p
```

Comparer leur complexité au pire cas.

Exercice 6

Écrire une fonction qui prend un tableau d'entiers T en argument et renvoie le tableau des sommes cumulées croissantes correspondantes, autrement dit un tableau S de même taille dont la k -ième composante vaut :

$$S[k] = \sum_{i=0}^k t[i]$$

Evaluer la complexité de cette fonction.

Est-il possible d'en écrire une version plus efficace ? Donner alors sa complexité

Exercice 7

Soit u la suite définie par $u_0 = 1$ et pour $n \in \mathbf{N}$, $u_{n+1} = \sin(u_n)$.

1. Écrire une fonction nommée suite prenant $n \in \mathbf{N}$ en argument et renvoyant u_n .
2. Que calcule la fonction suivante :

```
def mystere(n) :
    s = 0
    for k in range(n+1) :
        s = s + suite(k)
    return s
```

3. Déterminer en fonction de n le nombre d'additions et de calculs de *sinus* nécessaires pour calculer *mystere(n)*.

4. Écrire une autre fonction qui renvoie les mêmes valeurs que *mystere* mais avec une meilleure complexité.