10

Calculs de complexité

◇ Étudier la complexité (en temps) d'une fonction f, c'est déterminer son temps d'exécution T_f en fonction de la taille des données. En pratique :

- Ce temps d'exécution est compté en nombre d'*opérations élémentaires* (à préciser suivant les cas) ;
- La taille des données est également à préciser, pour une liste L c'est sa longueur ;
- On se contente d'un ordre de grandeur en utilisant la notation O.

On obtient souvent des relations du type $T_f(n) = O(n)$, $T_f(n) = O(n\ln(n))$, etc.

Exemple Python. Déterminer le maximum des éléments d'une liste L non vide de taille n. Complexité :

$$T_{\texttt{maximum}}(n) = O(n)$$

```
def maximum(L):
    m=L[0]
    for i in range(1,len(L)):
        if L[i]>m:
            m=L[i]
    return m
```

Exemple Python. Le nombre d'apparitions d'un élément x dans une liste L de taille n. Complexité :

$$T_{\texttt{nocc}}(n) = O(n)$$

```
def nocc(x,L):
    n=0
    for y in L:
        if x==y:
            n=n+1
    return n
```

Exemple Python. Recherche d'un élément majoritaire dans une liste L de taille n. Complexité :

$$T_{\texttt{majoritaire}}(n) = O(n^2)$$

```
def majoritaire(L):
    xmaj=L[0]
    nmaj=nocc(xmaj,L)
    for i in range(1,len(L)):
        if nocc(L[i],L)>nmaj:
            xmaj=L[i]
            nmaj=nocc(L[i],L)
    return xmaj
```

◇ On distingue parfois la complexité dans le meilleur cas et dans le pire cas. Il existe également une notion de complexité en moyenne.

Exemple Python. Rechercher si un élément x est présent dans une liste L de taille n. Complexité :

$$\begin{aligned} T_{\texttt{present}}(n) &= O(n) \text{ (pire cas)} \\ &= O(1) \text{ (meilleur cas)} \end{aligned}$$

```
def present(x,L):
    for y in L:
        if x==y:
            return True
    return False
```

Exemple Python. Construire la liste $[u_0, \dots, u_n]$ contenant les $n+1$ premiers termes de la suite (u_n) définie par $u_0 = 1$ et $u_{n+1} = 1 - 2u_n$. On donne deux versions.

```
def u(n):
    u=1
    for i in range(n):
        u=1-2*u
    return u
def liste_u(n):
    L=[]
    for i in range(n+1):
        L.append(u(i))
    return L
```

Ici : $T_{\texttt{u}}(n) = O(n)$ et $T_{\texttt{liste\_u}}(n) = O(n^2)$.

```
def liste_u(n):
    u=1
    L=[u]
    for i in range(n):
        u=1-2*u
        L.append(u)
    return L
def u(n):
    L=liste_u(n)
    return L[-1]
```

Ici : $T_{\texttt{liste\_u}}(n) = O(n)$ et $T_{\texttt{u}}(n) = O(n)$.

Exemple Python. Calcul rapide de A^N.

- Terminaison : n est entier, positif et strictement décroissant à chaque tour de boucle ;
- Correction :

$$y \times x^n = A^N$$

  est un invariant de boucle ;
- Complexité :

$$T_{\texttt{puiss\_rapide}}(N) = O(\log(N))$$

```
def puiss_rapide(A,N):
    (x,y,n)=(A,1,N)
    while n>0:
        if n%2==1:
            y=y*x
        x=x**2
        n=n/2
    return y
print puiss_rapide(4,3)
```


Exemple. On dispose de n objets numérotés de 0 à $n-1$, l'objet i a une masse $M[i]$ et une valeur $V[i]$. On a une masse maximale fixée $m > 0$ et on cherche à déterminer un sous-ensemble d'objets dont la masse totale est inférieure à m et dont la valeur totale est maximale. Algorithme : tester toutes les possibilités, c'est à dire tous les sous-ensembles de $[\![0, n-1]\!]$. Complexité : $T(n) = O(2^n)$

◇ Dans certain cas, on étudie aussi la complexité en espace (notation $E_f(n)$).