

CHA. 1 – ALGORITHMES ET FONCTION LOGARITHME

On note \mathbb{Q} l'ensemble des rationnels, et \mathbb{R}_+ les réels strictement positifs. Une fonction $f : \mathbb{Q} \rightarrow \mathbb{R}_+$ est *puissance* si elle vérifie $f(x+y) = f(x)f(y)$.

Exercice 1 Démontrez que, pour toute fonction puissance $f(x)$

1. $f(0) = 1$
2. $f(x-y) = f(x)/f(y)$
3. $f(ax) = f(x)^a$ pour tout entier naturel a
4. $f(1) = f(1/a)^a$ pour tout entier naturel a
5. $f(x)$ est injective (à moins que $f(x) = 1$ pour tout x)

Pour toute fonction puissance $f(x)$, on notera $f(x) = b^x$, où $b = f(1)$ est la base de f , et on suppose que les fonctions puissances peuvent être étendues pour être bijectives de l'ensemble des réels \mathbb{R} dans celui des réels strictement positifs \mathbb{R}_+ . Une fonction $g : \mathbb{R}_+ \rightarrow \mathbb{R}$ est *logarithme* si elle vérifie $g(xy) = g(x) + g(y)$.

Exercice 2 Démontrez que, pour toute fonction logarithme $g(x)$

1. $g(1) = 0$
2. $g(x/y) = g(x) - g(y)$
3. $g(x^q) = qg(x)$ pour tout rationnel q
4. $g(x)$ est injective (à moins que $g(x) = 0$ pour tout x)

Pour toute fonction logarithme $g(x)$, il existe une base b telle que $g(b^x) = x$, on note alors $g(x) = \log_b(x)$.

Exercice 3 Démontrez que $b^{n+1} = 1 + \sum_{i=0}^n (b-1)b^i$.

Exercice 4 Démontrez que pour tout entier x et tout entier $b \geq 2$

$$x = \sum_{i=0}^{\lfloor \log_b x \rfloor} x_i b^i$$

où $0 \leq x_i < b$ est un entier, pour tout $i = 0, \dots, \lfloor \log_b x \rfloor$.

Exercice 5 Démontrez que

$$\left\lfloor \frac{x}{p^2} \right\rfloor = \left\lfloor \frac{\left\lfloor \frac{x}{p} \right\rfloor}{p} \right\rfloor$$

Exercice 6 Combien retourne la fonction suivante ?

```
def r(x,p,k):  
    y=x  
    for i in range(1,p+1):  
        y=y//k  
    return y
```

Exercice 7 Combien valent $\log_3(9)$, $\log_2(1024)$, $\log_3(27)$, $\log_2(101372)$, $\log_{10}(1000000000000)$?

Exercice 8 Montrez que $\log_b n = \log_b a \times \log_a n$.

Exercice 9 Montrez que $a^{\log_b n} = n^{\log_b a}$.

Exercice 10 Montrez que $(b^q)^{\log_b a} = b^{q \log_b a} = a^q$.

Exercice 11 Montrez que pour tous entiers n et $b > 1$, il existe $p \in \mathbb{N}$ tel que $n \leq b^p \leq bn$.

Pour toute fonction $f, g : \mathbb{N} \rightarrow \mathbb{R}_+$, on note $f = O(g)$ si $f(n) \leq c g(n)$ pour une constante $c \in \mathbb{R}_+$.

Exercice 12 Démontrez que $\log_b n = O(\log_2 n)$ pour toute base $b > 1$.

Pour toute base $b > 1$, on note $O(\log n) = O(\log_b n)$.

CHA. 2 – COMPLEXITÉ ALGORITHMIQUE $O(\cdot)$, $\Omega(\cdot)$, $\Theta(\cdot)$

Exercice 13 Classez les fonctions suivantes: n^3 , $2n^2$, $n+3n^4$, $3 \log n + n \log n$, $n^2 \log n$, $n^{1.34}$, $n^{2.6}$, $\log n + n^2$, $\log \log n + 2n$

1. constant $O(1)$
2. logarithmique $O(\log n)$
3. linéaire $O(n)$
4. sous-linéaire $O(n \log n)$
5. quadratique $O(n^2)$
6. polynomial $O(n^k)$, où k est une constante > 2

Exercice 14 Montrez que $2^{\frac{1}{2}}$, $5^{\frac{1}{2}}$, $\log_2 3$, $\log_{10} 2 \notin \mathbb{Q}$.

Les opérations arithmétiques $+$, \times , $-$, \div , l'affectation $=$, et le test `if` sont appelées les *opérations élémentaires*.

Exercice 15 Combien d'opérations élémentaires effectuent ces programmes en python ?

```
def A():
    a=10
    while a > 0:
        a = a - 1
def B():
    for i in range(10):
        A()
def fact(n):
    res = 1
    for i in range(1,n+1):
        res = i * res
    return res
```

Un entier t représente la taille des données d'un algorithme, si l'on peut coder les données de l'algorithme avec un vecteur $x \in \{0, 1\}^t$. On note $f = \Omega(g)$ si $f(n) \geq c g(n)$ pour une constante $c \in \mathbb{R}_+$.

Exercice 16 Soit $T(n)$ le nombre d'opérations élémentaires effectuées par `fact(n)`. Montrez que $T(n) = O(n)$ mais aussi que $T(n) = \Omega(2^t)$ où t représente la taille des données.

Un algorithme est (de complexité) $O(f(n))$ s'il effectue au plus $O(f(n))$ opérations élémentaires.

Exercice 17 Quelle est la complexité des algorithmes de l'exercice 15 et celle de `fibonacci` ?

```
def fibonacci(n):
    v0 = 0
    v1 = 1
    for i in range(n-1):
        v0, v1 = v1, v0 + v1
    return v1
```

Un algorithme est (de complexité) $\Omega(f(n))$ s'il effectue au moins $\Omega(f(n))$ opérations élémentaires.

Exercice 18 Montrez que `fibonacci` est $\Omega(\phi^n)$ où $\phi \simeq 1.61$ est le nombre d'or

```
def fibonacci_rec(n):
    if n <= 1:
        return 1
    else:
        return fibonacci_rec(n-1)+fibonacci_rec(n-2)
```

On note $f = \Theta(g)$ si $f = O(g)$ et $f = \Omega(g)$, et naturellement un algorithme est (de complexité) $\Theta(f(n))$ s'il effectue $\Theta(f(n))$ opérations élémentaires.

Exercice 19 Quelle est `A(n)` et `F(n)` et quelle est leur complexité ?

```
def A(n):
    x=(n+1)/2
    for i in range(10):
        x = (x + n/x)/2
    return x
def F(n):
    phi=(1+A(5))/2
    phic=(1-A(5))/2
    return (1/A(5))*(phi**n - phic**n)
```

Exercice 20 Que fait `D(n,b)` et quelle est sa complexité ?

```
def D(n,b):
    tab=[]
    while n>0:
        tab=[n%b]+tab
        n=n//b
    return tab
```

Exercice 21 Quelle est la complexité de l'algorithme calculant le produit $C = AB$

$$A = \left(a_{ij} \right)_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \quad B = \left(b_{ij} \right)_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}} \quad C = \left(\sum_{k=1}^{k=n} a_{ik} b_{kj} \right)_{\substack{1 \leq i \leq n \\ 1 \leq j \leq n}}$$

de deux matrices carrées ?

Exercice 22 Quelle condition doivent vérifier des entiers a, b pour que $\frac{a}{b} \leq \frac{a-1}{b-1}$?

Exercice 23 Montrez que, pour $c > 0$ et $f(n) = 1 + c + c^2 + \dots + c^n$ alors

1. $f(n) = \Theta(1)$ si $c < 1$
2. $f(n) = \Theta(n)$ si $c = 1$

3. $f(n) = \Theta(c^n)$ si $c > 1$

Exercice 24 Montrez que $\log(n!) = \Theta(n \log n)$.

Exercice 25 Montrez que $\sum_{i=1}^{i=n} \frac{1}{i} = \Theta(\log n)$.

Un algorithme est *exponentiel* si sa complexité est $\Omega(x^n)$ pour un réel $x > 1$ et un paramètre $n = \Theta(t)$ où t représente la taille des données.

Exercice 26 Que fait l'algorithme `C(n,p)` ? montrez qu'il est exponentiel.

```
def C(n,p):
    if p==n or p==0:
        return 1
    return C(n-1,p-1)+C(n-1,p)
```

Exercice 27 Montrer plus simplement qu'à l'exercice 18 que `fibonacci` est exponentiel.

Le premier algorithme dont la complexité a été étudiée (en 1844 par Lamé) est celui d'Euclide, dont la validité est basée sur (1) $\text{pgcd}(a, 0) = a$, et (2) $\text{pgcd}(a, b) = \text{pgcd}(b, r)$ où $a = bq + r$, $0 \leq r < b$.

Exercice 28 Montrer que Euclide est $O(\log b)$.

```
def Euclide(a,b):
    if b==0:
        return a
    else:
        return Euclide(b,a%b)
```

La complexité d'un algorithme contenant des boucles `while` dépend du nombre de leurs itérations.

Exercice 29 Programmez une fonction `log(n)` renvoyant le plus petit entier t tel que $n \leq 2^t$, puis montrez que `bincount` est $\Omega(n)$, $O(n^2)$ et $\Theta(n)$.

```
def bincount(n):
    tab=[]
    for size in range(log(n)):
        tab.append(0)
    for i in range(n-1):
        j=0
        while tab[j]==1:
            tab[j]=0
            j=j+1
        tab[j]=1
```

Exercice 30 Donnez une version polynomiale de l'algorithme de l'exercice 26.

$$\text{CHA. 3 - DIVIDE-AND-CONQUER } T(n) = aT(n/b) + \Theta(n^c)$$

C'est l'équation satisfaite par le temps d'exécution $T(n)$ d'un algorithme de type *divide-and-conquer*, avec $T(1) = O(1)$, où le *temps d'exécution* $T(n)$ est le nombre opérations élémentaires en fonction d'un paramètre $n = \Theta(t)$ avec t représentant la taille des données. Un tel algorithme divise le problème d'origine de taille n en a sous-problèmes de taille b , et $\Theta(n^c)$ est le temps nécessaire à la division du problème et à la reconstitution de sa solution à partir des solutions des sous-problèmes.

Exercice 31 Pour tri fusion on a $T(n) = 2T(n/2) + \Theta(n)$. En déduire que $T(n) = \Theta(n \log n)$.

```
def tri_fusion(tab):
    n=len(tab)
    if n >1:
        mi = n//2
        L = tab[:mi]
        R = tab[mi:]
        tri_fusion(L)
        tri_fusion(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                tab[k] = L[i]
                i+= 1
            else:
                tab[k] = R[j]
                j+= 1
            k+= 1
        while i < len(L):
            tab[k] = L[i]
            i+= 1
            k+= 1
        while j < len(R):
            tab[k] = R[j]
            j+= 1
            k+= 1
```

Exercice 32 Montrez qu'on a $T(n) = \Theta(\log n)$ pour $T(n) = T(n/2) + \Theta(1)$.

Le temps d'exécution d'un algorithme divide-and-conquer admet une expression explicite, assez simple si n est une puissance de b .

Exercice 33 Montrez que $T(b^t) = \sum_{i=0}^{t-1} a^i (b^{t-i})^c$.

On peut généraliser le cas du tri fusion où $\log_b a = \log_2 2 = 1 = c$.

Exercice 34 Montrez que si $c = \log_b a$, alors $T(n) = \Theta(n^c \log n)$.

Indication: utiliser les exercices 9, 10 et 33.

Les opérations arithmétiques $+$, $-$, \times , \div sont des opérations élémentaires si les entiers sur lesquelles elles opèrent ont une taille maximum constante. Si cette taille maximum est une variable n , les opérations élémentaires sont les opérations sur les chiffres simples. Pour deux entiers $x, y < 10^{n+1}$, les algorithmes appris à la petite école effectuent l'addition $x + y$ en $\Theta(n)$ additions de chiffres, et la multiplication $x \times y$ en $\Theta(n^2)$ multiplications de chiffres.

Exercice 35 Soient $x = a.10^{\frac{n}{2}} + b$ et $y = c.10^{\frac{n}{2}} + d$ avec $a, b, c, d < 10^{\frac{n}{2}+1}$, et n pair. Montrez que

$$xy = ac.10^n + ((a + b)(c + d) - ac - bd).10^{\frac{n}{2}} + bd$$

et en déduire un algorithme divide-and-conquer pour la multiplication des entiers à n chiffres.

Exercice 36 Montrez qu'on a $T(n) = \Theta(n^{\log_2 3})$ pour $T(n) = 3T(n/2) + \Theta(n)$.

Puisque $\log_2 3 \simeq 1.59$ on a amélioré l'algorithme classique de multiplication.

Exercice 37 Montrez que $T(n) = \Theta(n^c) \times \sum_{i=0}^{i=\log_b n} (\frac{a}{b^c})^i$.

La forme explicite de $T(n)$ dépend d'une série géométrique de raison $\frac{a}{b^c}$.

Exercice 38 Montrez que

1. $T(n) = \Theta(n^c)$ si $c > \log_b a$
2. $T(n) = \Theta(n^c \log n)$ si $c = \log_b a$ (exercice 34)
3. $T(n) = \Theta(n^{\log_b a})$ si $c < \log_b a$

Exercice 39 Concevez un algorithme *divide-and-conquer* pour obtenir le produit $C = AB$ de deux matrices carrées A, B de taille $n = 2^p$ basé sur

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

où X_{ij} ($X = A, B, C$) est la sous-matrice carrée de X formée des 2^{p-1} premières (resp. dernières) lignes si $i = 1$ (resp. si $i = 2$), et des 2^{p-1} premières (resp. dernières) colonnes si $j = 1$ (resp. si $j = 2$).

Exercice 40 Comparez le temps d'exécution des algorithmes des exercices 21 et 39.

Exercice 41 Montrez comment vérifier rapidement que les 4 sous-matrices de C de l'exercice 39 satisfont

$$\begin{aligned} C_{12} &= P_5 + P_3 \\ C_{21} &= P_2 + P_4 \\ C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 \end{aligned} \quad \text{avec} \quad \begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}) \\ P_2 &= (A_{21} + A_{22})(B_{11}) \\ P_3 &= (A_{11})(B_{12} - B_{22}) \\ P_4 &= (A_{22})(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})(B_{22}) \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}) \end{aligned}$$

grâce au schéma

$$\begin{aligned} \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \end{pmatrix} &= \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & \cdot & \cdot \end{pmatrix} \\ \begin{pmatrix} \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} &= \begin{pmatrix} \cdot & \cdot & + & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \\ \begin{pmatrix} + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} &= \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} - \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & + & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & + & \cdot & \cdot \\ \cdot & + & \cdot & - \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \\ \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & + \end{pmatrix} &= \begin{pmatrix} + & \cdot & \cdot & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & + \end{pmatrix} + \begin{pmatrix} \cdot & \cdot & \cdot & \cdot \\ + & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ - & \cdot & \cdot & \cdot \end{pmatrix} - \begin{pmatrix} \cdot & \cdot & + & + \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} + \begin{pmatrix} - & \cdot & + & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot \end{pmatrix} \end{aligned}$$

Exercice 42 Concevez un algorithme *divide-and-conquer* pour obtenir le produit de deux matrices carrées de taille n de complexité $\Theta(n^{\log_2 7})$.

Puisque $\log_2 7 \leq 2.81$ on a amélioré la multiplication de matrices.

Exercice 43 Montrez expérimentalement que les temps de calcul de `fibonacci` de l'exercice 18 forme une suite géométrique de raison le nombre d'or.

Exercice 44 Implémentez $F(n)$ de l'exercice 19 et comparez les valeurs obtenues pour un $n \geq 71$ avec `fibonacci(n)` de l'exercice 17. Pourrait on obtenir $\sqrt{5}$ avec suffisamment de précision ? La diagonalisation $X = PDP^{-1}$ permettrait-elle de calculer précisément $X^n = PD^nP^{-1}$? pour

$$X = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

Exercice 45 Concevoir un algorithme en python donnant X^n en $\Theta(\log n)$ si l'on suppose la taille maximum des entiers constantes, où X est la matrice de l'exercice 44.

Exercice 46 Montrez que l'algorithme de l'exercice 45 permet de déterminer le n ème terme de la suite de Fibonacci en $O(n^{1.59})$ même en prenant en compte la variabilité de la taille des entiers.

CHA. 4 – BACKTRACKING OU FORCE BRUTE

Exercice 47 Soient b et n des variables globales entières ≥ 2 , $A=[0 \text{ for } i \text{ in } \text{range}(n)]$ un tableau entier global, et la fonction suivante:

```
def count(i):
    for j in range(b):
        A[i]=j
        if i==n-1:
            print A
        else:
            count(i+1)
```

Énoncez une propriété $p(i)$ de `count(i)`, dépendante d'un entier i , satisfaisant :

1. $p(n - 1)$ est vraie,
2. si $p(i + 1)$ est vraie, alors $p(i)$ est vraie.

Exercice 48 Combien d'appels à `count(i)` sont déclenchés par l'appel à `count(0)` ?

Exercice 49 Soient n une variable globale entière ≥ 2 , $A=[0 \text{ for } i \text{ in } \text{range}(n)]$ un tableau entier global, et $B=[True \text{ for } i \text{ in } \text{range}(n)]$ un tableau booléen global. Donnez le code d'une fonction `permut(i)` telle que l'appel à `permut(0)` affiche les $n!$ permutations.

Exercice 50 On considère un ensemble de n investissements potentiels (c_i, g_i) caractérisés par le coût c_i et le gain g_i , pour $i = 0, \dots, n - 1$. On suppose que chaque investissement est rentable, c'est-à-dire $c_i < g_i$. On dispose d'un budget $b < \sum_{i=0}^{n-1} c_i$ pour investir. Un portefeuille est un sous-ensemble $X \subseteq \{0, \dots, n - 1\}$ tel que $\sum_{i \in X} c_i \leq b$.

1. Donner un programme python qui affiche tous les portefeuilles.
2. Modifier le pour déterminer le portefeuille le plus rentable.

CHA. 5 – TRI

La complexité d'un algorithme peut s'exprimer en fonction de différents paramètres caractéristiques des données.

Exercice 51 Programmez une fonction `max(tab)` renvoyant la plus grande valeur contenue dans un tableau, puis comparez le nombre d'opérations élémentaires effectuées par `tri den` pour `A=[6,8,7,4,5,0,3]` et `A=[16,58,71,44,5,10,30]`. Montrer que `tri den` est exponentiel.

```
def tri_den(A):
    n=len(A)
    k=max(A)
    B=[]
    for j in range(n):
        B.append(0)
    C=[]
    for i in range(k+1):
        C.append(0)
    for j in range(n):
        C[A[j]]=C[A[j]]+1
    for i in range(1,k+1):
        C[i]=C[i-1]+C[i]
    for j in range(n):
        B[C[A[j]]-1]=A[j]
    A=B
```

Sous quelle hypothèse `tri den` est $O(n)$?

La complexité d'un algorithme peut être différente pour des entrées de même taille, mais par défaut la complexité concerne le pire cas possible.

Exercice 52 La complexité de `tri insertion` est $\Theta(n)$ dans le meilleur cas et $\Theta(n^2)$ dans le pire cas, montrer qu'il est en fait $\Theta(n^2)$ en moyenne.

```
def tri_insertion(tab):
    for j in range(1,len(tab)):
        x=tab[j]
        i=j-1
        while i>=0 and tab[i]>x:
            tab[i+1]=tab[i]
            i=i-1
        tab[i+1]=x
```

CHA. 6 – STRUCTURES DE DONNÉES ARBORESCENTES

Une fonction $p : T \setminus \{r\} \rightarrow T$ est père si pour tout $x \in T$, il existe un entier k tel que $p^k(x) = r$; où l'on note $p^0(x) = x$ et $p^{k+1}(x) = p(p^k(x))$.

Exercice 53 Soit $T = [n]$ l'ensemble des n premiers entiers, $r = 1$ et $p(x) = \lfloor \frac{x}{2} \rfloor$. Montrez que $p(x)$ est une fonction père, et donnez le plus grand entier $h(n)$ telle que $p^{h(n)}(x) = r$ pour un $x \in T$.

Exercice 54 Montrez que $p : T \setminus \{r\} \rightarrow T$ est une fonction père si et seulement si $p^k(x) = x$ implique $k = 0$.