

# CI 2 : ALGORITHMIQUE & PROGRAMMATION

## ALGORITHMES D'INFORMATIQUE

1	Recherches dans une liste	2
1.1	Recherche d'un nombre dans une liste	2
1.2	Recherche du maximum dans une liste de nombre	3
1.3	Recherche par dichotomie dans un tableau trié	3
2	Gestion d'une liste de nombres	4
2.1	Calcul de la moyenne	4
2.2	Calcul de la variance	4
2.3	Calcul de la médiane	5
3	Chaînes de caractères	5
3.1	Recherche d'un mot dans une chaîne de caractères	5
4	Calcul numérique	6
4.1	Recherche du zéro d'une fonction continue monotone par la méthode de dichotomie	6
4.2	Recherche du zéro d'une fonction continue monotone par la méthode de Newton	6
4.3	Méthode des rectangles pour le calcul approché d'une intégrale sur un segment	7
4.3.1	Méthode des rectangles à gauche	7
4.3.2	Méthode des rectangles à droite	8
4.3.3	Méthode des rectangles – Point milieu	9
4.4	Méthode des trapèzes pour le calcul approché d'une intégrale sur un segment	9
4.5	Méthode d'Euler pour la résolution d'une équation différentielle	10
4.5.1	Méthode d'Euler explicite	10
4.6	Algorithme de Gauss – Jordan [4]	12
5	Algorithmes de tris	12
5.1	Tri par sélection	12
5.2	Tri par insertion	13
5.2.1	Méthode 1	13
5.2.2	Méthode 2	14
5.3	Tri shell	14
5.4	Tri rapide «Quicksort»	14
5.4.1	Tri rapide	14
5.4.2	Tri rapide optimisé	16
5.5	Tri fusion	17
6	Algorithmes classiques	19
6.1	Division euclidienne	19
6.2	Algorithme d'Euclide	19
6.3	Calcul de puissance	20
6.3.1	Algorithme naïf	20
6.3.2	Exponentiation rapide itérative	20

# 1 Recherches dans une liste

## 1.1 Recherche d'un nombre dans une liste

Pseudo Code

**Algorithme :** Recherche naïve d'un nombre dans une liste triée ou non

**Données :**

- n, int : un entier
- tab, liste : une liste d'entiers triés ou non triés

**Résultat :**

- un booléen : Vrai si le nombre est dans la liste, Faux sinon.

```
is_number_in_list(n,tab) :
    l ← longueur(tab)
    Pour i allant de 1 à l faire :
        Si tab[i] = n alors :
            Retourne Vrai
        Fin Si
    Fin Faire
    Retourne Faux
Fin fonction
```

python

```
def is_number_in_list(nb,tab):
    """Renvoie True si le nombre nb est dans la liste
    de nombres tab
    Keyword arguments:
    * nb, int — nombre entier
    * tab, list — liste de nombres entiers
    """
    for i in range(len(tab)):
        if tab[i]==nb:
            return True
    return False
```

python

```
def is_number_in_list(nb,tab):
    """Renvoie True si le nombre nb est dans la liste
    de nombres tab
    Keyword arguments:
    * nb, int — nombre entier
    * tab, list — liste de nombres entiers
    """
    i=0
    while i<len(tab) and tab[i]!=nb:
        i+=1
    return i<len(tab)
```

Remarque

Ces algorithmes sont modifiables aisément dans le cas où on souhaiterait connaître l'index du nombre recherché.

## 1.2 Recherche du maximum dans une liste de nombre

Pseudo Code

**Algorithme :** Recherche du maximum dans une liste de nombres

**Données :**

– tab, liste : une liste de nombres

**Résultat :**

– maxi, réel : maximum de la liste

```
what_is_max(tab) :
    n ← longueur(tab)
    i ← 2
    maxi ← tab[1]
    Tant que i < n faire :
        Si tab[i] > maxi alors :
            maxi ← tab[i]
        Fin si
        i ← i+1
    Fin tant que
    Retourner maxi
Fin fonction
```

python

```
def what_is_max(tab):
    """
    Renvoie le plus grand nombre d'une liste
    Keyword arguments:
    tab — liste de nombres
    """
    i=1
    maxi=tab[0]
    while i<len(tab):
        if tab[i]>maxi:
            maxi=tab[i]
        i+=1
    return maxi
```

## 1.3 Recherche par dichotomie dans un tableau trié

Pseudo Code

**Algorithme :** Recherche par dichotomie d'un nombre dans une liste triée

**Données :**

– nb, int : un entier

– tab, liste : une liste d'entiers triés

**Résultat :**

– m, int : l'index du nombre recherché

– None : cas où nb n'est pas dans tab

```
is_number_in_list_dicho(nb,tab) :
    g ← 0
    d ← longueur(tab)
    Tant que g < d alors :
        m ← (g+d) div 2
        Si tab[m]=nb alors :
            Retourne m
        Sinon si tab[m]<nb alors :
            g ← m+1
        Sinon, alors :
            d ← m-1
    Fin Si
    Fin Tant que
    Retourne None
Fin fonction
```

python

```
def is_number_in_list_dicho(nb,tab):
    """
    Recherche d'un nombre par dichotomie dans un
    tableau trié.
    Renvoie l'index si le nombre nb est dans la liste
    de nombres tab.
    Renvoie None sinon.
    Keyword arguments:
    nb,int — nombre entier
    tab, list — liste de nombres entiers triés
    """
    g, d = 0, len(tab)-1
    while g <= d:
        m = (g + d) // 2
        if tab[m] == nb:
            return m
        if tab[m] < nb:
            g = m+1
        else:
            d = m-1
    return None
```

## 2 Gestion d'une liste de nombres

### 2.1 Calcul de la moyenne

Pseudo Code

**Algorithme :** Calcul de la moyenne arithmétique des nombres d'une liste

**Données :**

– tab, liste : une liste de nombres

**Résultat :**

– res, réel : moyenne des nombres

**calcul\_moyenne(tab) :**

  n ← **longueur**(tab)

  res ← 0

**Pour** i allant de 1 à n **faire** :

    res ← res+tab[i]

**Fin faire**

**Retourner** res/n

**Fin fonction**

python

```
def calcul_moyenne(tab):
```

```
    """
```

```
    Renvoie la moyenne des valeurs d'une liste de nombres.
```

```
    Keyword arguments:
```

```
    tab — liste de nombres
```

```
    """
```

```
    res = 0
```

```
    for i in range(len(tab)):
```

```
        res = res+tab[i]
```

```
    return res/(len(tab))
```

### 2.2 Calcul de la variance

Soit une série statistique prenant les  $n$  valeurs  $x_1, x_2, \dots, x_n$ . Soit  $m$  la moyenne de ces valeurs. La variance est définie par :

$$v = \frac{1}{n} \sum_{i=1}^n (x_i - m)^2$$

Pseudo Code

**Algorithme :** Calcul de la variance des nombres d'une liste

**Données :**

– tab, liste : une liste de nombres

– m, réel : moyenne de la liste

**Résultat :**

– res, réel : variance

**calcul\_variance(tab, m) :**

  n ← **longueur**(tab)

  res ← 0

**Pour** i allant de 1 à n **faire** :

    res ← res+(tab[i]-m)\*\*2

**Fin faire**

**Retourner** res/n

**Fin fonction**

python

```
def calcul_variance(tab, m):
```

```
    """
```

```
    Renvoie la variance des valeurs d'un tableau.
```

```
    Keyword arguments:
```

```
    tab — liste de nombres
```

```
    m — moyenne des valeurs
```

```
    """
```

```
    res = 0
```

```
    for i in range(len(tab)):
```

```
        res = res+(tab[i]-m)**2
```

```
    return res/(len(tab))
```

## 2.3 Calcul de la médiane

Pseudo Code

**Algorithme :** Recherche de la valeur médiane d'une liste de nombres triés

**Données :**

– tab, liste : liste de nombres triés

**Résultat :**

– flt : valeur de la médiane

**mediane**(tab) :

n ← Longueur(tab)

**Si** n modulo 2 = 0 **Alors :**

i ← n/2

**Retourner** (tab[i] + tab[i+1])/2

**Sinon :**

i ← n div 2 + 1

**Retourner** (tab[i])

**Fin fonction**

python

```
def calcul_mediane(tab):
```

```
    """
```

```
    Calcule la médiane des éléments d'un tableau trié.
```

```
    Keyword arguments:
```

```
    tab — liste de nombres
```

```
    """
```

```
    if len(tab)%2 == 0 :
```

```
        i=len(tab)//2
```

```
        return (tab[i-1]+tab[i])/2
```

```
    else :
```

```
        i = len(tab)//2
```

```
    return tab[i]
```

## 3 Chaînes de caractères

### 3.1 Recherche d'un mot dans une chaîne de caractères

python

```
def index_of_word_in_text(mot, texte):
```

```
    """ Recherche si le mot est dans le texte.
```

```
    Renvoie l'index si le mot est présent, None sinon.
```

```
    Keyword arguments:
```

```
    mot — mot recherché
```

```
    texte — texte
```

```
    """
```

```
    for i in range(1 + len(texte) - len(mot)):
```

```
        j = 0
```

```
        while j < len(mot) and mot[j] == texte[i + j]:
```

```
            j += 1
```

```
        if j == len(mot):
```

```
            return i
```

```
    return None
```

## 4 Calcul numérique

### 4.1 Recherche du zéro d'une fonction continue monotone par la méthode de dichotomie

Pseudo Code

**Algorithme :** Recherche de la solution de  $f(x)=0$  par dichotomie

**Données :**

- $f$ , fonction : fonction continue et monotone sur  $[a, b]$
- $a, b$ , réels : nombre réels tels que  $a < b$
- $\varepsilon$ , réel : tolérance du calcul

**Résultat :**

- $flt$  : solution de l'équation

**solveDichotomie**( $f, a, b, \varepsilon$ ) :

$g \leftarrow a$

$d \leftarrow b$

**Tant que**  $d-g > \varepsilon$  **Alors :**

$m \leftarrow (g+d)/2$

**Si**  $f(g)*f(m) \leq 0$  **Alors :**

$d \leftarrow m$

**Sinon :**

$g \leftarrow m$

**Fin Si**

**Fin Tant que**

**Fin fonction**



```
def solveDichotomie(f, a, b, eps):
```

```
    """
```

```
    Recherche par dichotomie de la solution de l'équation f(x)=0.
```

```
    Keywords arguments :
```

```
    Entrées :
```

```
        a, b, flt : Nombre réels tels que a<b
```

```
        f, function : fonction continue et monotone sur [a,b]
```

```
        eps, flt : tolérance de la résolution
```

```
    Sortie :
```

```
        flt : solution de la fonction
```

```
    """
```

```
    g = a
```

```
    d = b
```

```
    while (d-g) > eps:
```

```
        m = (g+d)/2
```

```
        if f(g) * f(m) <= 0 :
```

```
            d = m
```

```
        else :
```

```
            g = m
```

```
    return (g+d)/2
```

Pseudo Code

**Algorithme :** Recherche de la solution de  $f(x)=0$  par dichotomie

**Données :**

- $f$ , fonction : fonction continue et monotone sur  $[a, b]$
- $a, b$ , réels : nombre réels tels que  $a < b$
- $\varepsilon$ , réel : tolérance du calcul

**Résultat :**

- $flt$  : solution de l'équation

**solveDichotomie**( $f, a, b, \varepsilon$ ) :

$g \leftarrow a$

$d \leftarrow b$

**Tant que**  $d-g > \varepsilon$  **Alors :**

$m \leftarrow (g+d)/2$

**Si**  $f(g)*f(m) \leq 0$  **Alors :**

$d \leftarrow m$

**Sinon :**

$g \leftarrow m$

**Fin Si**

**Fin Tant que**

**Fin fonction**



```
def solveNewton(f, df, a, eps):
```

```
    """
```

```
    Recherche par la méthode de Newton de la solution de l'équation f(x)=0.
```

```
    Keywords arguments :
```

```
    Entrées :
```

```
        f, function : fonction à
```

```
        valeur de IR dans IR
```

```
        df, function : dérivée de f à
```

```
        valeur de IR dans IR
```

```
        a, flt : solution initiale
```

```
        eps, flt : tolérance de la résolution
```

```
    Sortie :
```

```
        flt : solution de la fonction
```

```
    """
```

```
    c = a-f(a)/df(a)
```

```
    while abs(c-a)>eps:
```

```
        a = c
```

```
        c = c-f(c)/df(c)
```

```
    return c
```

Remarque



```
def derive_fonctions(f,x,eps):
    return (f(x+eps)-f(x))/(eps)
```

La dérivée de  $f$  notée  $f'$  pourra être une fonction qui a été définie. On peut aussi calculer la dérivée de façon numérique. Ainsi, en tenant compte des précautions mathématiques d'usage, il est possible de procéder ainsi :

## 4.3 Méthode des rectangles pour le calcul approché d'une intégrale sur un segment

### 4.3.1 Méthode des rectangles à gauche

Pseudo Code

**Algorithme :** Calcul d'intégrale par la méthode des rectangles à gauche

**Données :**

- $f$ , fonction : fonction définie sur  $[a, b]$
- $a$ , réel : borne inférieure de l'intervalle de définition
- $b$ , réel : borne supérieure de l'intervalle de définition,  $b \geq a$
- $nb$ , entiers : nombre d'échantillons pour calculer l'intégrale

**Résultat :**

- $res$ , réel : valeur approchée de  $\int_a^b f(t)dt$

**integrale\_rectangles\_gauche**( $f,a,b,nb$ ) :

$res \leftarrow 0$

$pas \leftarrow (b-a)/nb$

$x \leftarrow a$

**Tant que**  $x < b$ , **Faire :**

$res \leftarrow res + f(x)$

$x \leftarrow x + pas$

**Fin Tant que**

$res \leftarrow res * pas$

**Retourner**  $res$

**Fin Fonction**



```
def integrale_rectangles_gauche(f,a,b,nb):
    """
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la
    méthode des rectangles à gauche.
    Keywords arguments :
    f — fonction à valeur dans IR
    a — flt, borne inférieure de l' intervalle d'intégration
    b — flt, borne supérieure de l' intervalle d'intégration
    nb — int, nombre d'échantillons pour le calcul
    """
    res = 0
    pas = (b-a)/nb
```



```
x = a
while x < b:
    res = res + f(x)
    x = x + pas
return res * pas
```

### 4.3.2 Méthode des rectangles à droite

Pseudo Code

**Algorithme :** Calcul d'intégrale par la méthode des rectangles à droite

**Données :**

- f, fonction : fonction définie sur  $[a, b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition,  $b \geq a$
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

**Résultat :**

- res, réel : valeur approchée de  $\int_a^b f(t)dt$

**integrale\_rectangles\_droite(f,a,b,nb) :**

res  $\leftarrow$  0

pas  $\leftarrow$  (b-a)/nb

x  $\leftarrow$  a + pas

**Tant que** x  $\leq$  b : **Faire**

res  $\leftarrow$  res + f(x)

x  $\leftarrow$  x + pas

**Fin Tant que**

res  $\leftarrow$  res \* pas

**Retourner** res

**Fin Fonction**



```
def integrale_rectangles_droite (f,a,b,nb):
    """
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la
    méthode des rectangles à droite.
    Keywords arguments :
    f — fonction à valeur dans IR
    a — flt , borne inférieure de l' intervalle d'intégration
    b — flt , borne supérieure de l' intervalle d'intégration
    nb — int, nombre d'échantillons pour le calcul
    """
    res = 0
    pas = (b-a)/nb
    x = a+pas
    while x <= b:
        res = res + f(x)
        x = x + pas
    return res * pas
```



### 4.3.3 Méthode des rectangles – Point milieu

Pseudo Code

**Algorithme :** Calcul d'intégrale par la méthode des rectangles – point milieu

**Données :**

- f, fonction : fonction définie sur  $[a, b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition,  $b \geq a$
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

**Résultat :**

- res, réel : valeur approchée de  $\int_a^b f(t)dt$

**integrale\_rectangles\_milieu(f,a,b,nb) :**

res  $\leftarrow$  0

pas  $\leftarrow$  (b-a)/nb

x  $\leftarrow$  a+pas/2

**Tant que** x < b : **Faire**

res  $\leftarrow$  res + f(x)

x  $\leftarrow$  x+pas

**Fin Tant que**

res  $\leftarrow$  res\*pas

**Retourner** res

python

```
def integrale_rectangles_milieu (f,a,b,nb):
```

```
    """
```

```
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la méthode du point milieu.
```

```
    Keywords arguments :
```

```
    f — fonction à valeur dans IR
```

```
    a — float, borne inférieure de l' intervalle d'intégration
```

```
    b — float, borne supérieure de l' intervalle d'intégration
```

```
    nb — int, nombre d'échantillons pour le calcul
```

```
    """
```

```
    res = 0
```

```
    pas = (b-a)/nb
```

```
    x = a+pas/2
```

```
    while x < b:
```

```
        res = res + f(x)
```

```
        x = x + pas
```

```
    return res*pas
```

### 4.4 Méthode des trapèzes pour le calcul approché d'une intégrale sur un segment

---

**Algorithme :** Calcul d'intégrale par la méthode des trapèzes

---

**Données :**

- f, fonction : fonction définie sur  $[a, b]$
- a, réel : borne inférieure de l'intervalle de définition
- b, réel : borne supérieure de l'intervalle de définition,  $b \geq a$
- nb, entiers : nombre d'échantillons pour calculer l'intégrale

**Résultat :**

- res, réel : valeur approchée de  $\int_a^b f(t)dt$

**integrale\_trapeze(f,a,b,nb) :**

res ← 0

pas ← (b-a)/nb

x ← a+pas

**Tant que** x < b, **Faire :**

res ← res + f(x)

x ← x+pas

**Fin Tant que**

res ← pas\*(res + (f(a) + f(b))/2 )

**Retourner** res

---

```
def integrale_trapeze(f,a,b,nb):
```

```
    """
```

```
    Calcul de la valeur approchée de l'intégrale de f(x) entre a et b par la méthode des trapèzes.
```

```
    Keywords arguments :
```

```
    f — fonction à valeur dans IR
```

```
    a — float, borne inférieure de l' intervalle d'intégration
```

```
    b — float, borne supérieure de l' intervalle d'intégration
```

```
    nb — int, nombre d'échantillons pour le calcul
```

```
    """
```

```
    res = 0
```

```
    pas = (b-a)/nb
```

```
    x = a+pas
```

```
    while x < b:
```

```
        res = res + f(x)
```

```
        x = x + pas
```

```
    res = pas*(res+(f(a)+f(b))/2)
```

```
    return res
```

En raison de la comparaison de réels, il pourrait être préférable de réaliser la boucle while sur un compteur d'échantillons.

## 4.5 Méthode d'Euler pour la résolution d'une équation différentielle

### 4.5.1 Méthode d'Euler explicite

Résolution de l'équation différentielle :

$$y(t) + \tau \frac{dy(t)}{dt} = y_f$$

### Algorithme : Méthode d'Euler explicite

#### Données :

- tau, réel : constante de temps
- y\_0, réel : valeur initiale de y
- y\_f, réel : valeur finale y
- t\_f, réel : temps de la simulation numérique
- nb, entier : nombre d'échantillons pour calculer les valeurs de y

#### Résultat :

- res, liste : liste des couples (t,y(t)).

**euler\_explicite**(tau,y\_0,y\_f,t\_f,nb):

**Initialiser** res

    t ← 0

    y ← y\_0

    pas ← t\_f/nb

**Tant que** t < t\_f **Faire** :

**Ajouter** (t,y) à res

        y ← y + pas \*(y\_f-y)/tau

        t ← t + pas

**Fin Tant que**

**Retourner** res

```
def euler_explicite (tau,y0,yf,tf,nb):
```

```
    """
```

```
    Résolution d'une équation différentielle d'ordre 1 en utilisant la méthode d'Euler explicite .
```

```
    Keywords arguments :
```

```
    tau — flt , constante de temps de l'équation différentielle
```

```
    y0 — flt , valeur initiale de y(t)
```

```
    yf — flt valeur finale de y(t)
```

```
    tf — flt temps de fin de la simulation
```

```
    nb — int, nombre d'échantillons pour la simulation
```

```
    """
```

```
    t = 0
```

```
    y = y0
```

```
    pas = tf / nb
```

```
    res = []
```

```
    while t < tf:
```

```
        res.append((t,y))
```

```
        y = y + pas*(yf-y)/tau
```

```
        t = t + pas
```

```
    return res
```

## 4.6 Algorithme de Gauss – Jordan [4]

```
def recherche_pivot(A,i):
    n = len(A) # le nombre de lignes
    j = i # la ligne du maximum provisoire
    for k in range(i+1, n):
        if abs(A[k][i]) > abs(A[j][i]):
            j = k # un nouveau maximum provisoire
    return j

def echange_lignes(A,i,j):
    # Li <—> Lj
    A[i][:], A[j][:] = A[j][:], A[i][:]

def transvection_ligne(A, i, j, mu):
    # L_i <— L_i + mu.L_j """
    nc = len(A[0]) # le nombre de colonnes
    for k in range(nc):
        A[i][k] = A[i][k] + mu * A[j][k]

def resolution (AA, BB):
    """Résolution de AA.X=BB; AA doit etre inversible"""
    A, B = AA.copy(), BB.copy()
    n = len(A)
    assert len(A[0]) == n
    # Mise sous forme triangulaire
    for i in range(n):
        j = recherche_pivot(A, i)
        if j > i:
            echange_lignes(A, i, j)
            echange_lignes(B, i, j)
        for k in range(i+1, n):
            x = A[k][i] / float(A[i][i])
            transvection_ligne(A, k, i, -x)
            transvection_ligne(B, k, i, -x)

    # Phase de remontée
    X = [0.] * n
    for i in range(n-1, -1, -1):
        X[i] = (B[i][0] - sum(A[i][j]*X[j] for j in range(i+1,n))) / A[i][i]
    return X
```



## 5 Algorithmes de tris

### 5.1 Tri par sélection

```
# Tri par sélection
def tri_selection(tab):
    for i in range(0, len(tab)):
        indice = i
        for j in range(i+1, len(tab)):
            if tab[j] < tab[indice]:
                indice = j
        tab[i], tab[indice] = tab[indice], tab[i]
```





return tab

## 5.2 Tri par insertion

### 5.2.1 Méthode 1

#### Algorithme : Tri par insertion – Méthode 1

##### Données :

- tab, liste : une liste de nombres

##### Résultat :

- tab, liste : la liste de nombres triés

```

tri_insertion(tab) :
    n ← longueur(tab)
    Pour i de 2 à n :
        x ← tab[i]
        j ← 1
        Tant que j ≤ i-1 et tab[j] < x :
            j ← j+1
        Fin Tant que
        Pour k de i-1 à j-1 par pas de -1 faire :
            tab[k+1] ← tab[k]
        Fin Pour
        tab[j] ← x
    Fin Pour
    
```

Pseudo Code

```
def tri_insertion_01(tab):
```

```
    """
```

```
    Trie une liste de nombre en utilisant la méthode
    du tri par insertion .
```

```
    En Python, le passage se faisant par référence, il
    n'est pas indispensable de retourner le tableau.
```

```
    Keyword arguments:
```

```
    tab — liste de nombres
```

```
    """
```

```

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



### Estimation de la complexité

- Meilleur des cas, le tableau est trié à l'envers, la complexité est linéaire :  $\mathcal{O}(n)$ .
- Pire des cas, le tableau est trié, la complexité est quadratique :  $\mathcal{O}(n^2)$ .

## 5.2.2 Méthode 2

### Algorithme : Tri par insertion – Méthode 2

#### Données :

- tab, liste : une liste de nombres

#### Résultat :

- tab, liste : la liste de nombres triés

```

tri_insertion(tab) :
    n ← longueur(tab)
    Pour i de 2 à n :
        x ← tab[i]
        j ← i
        Tant que j > 1 et tab[j-1] > x :
            tab[j] ← tab[j-1]
            j ← j-1
        Fin Tant que
        tab[j] ← x
    Fin Pour
    
```

Pseudo Code

```
def tri_insertion_02(tab):
```

```
    """
```

```
    Trie une liste de nombre en utilisant la méthode
    du tri par insertion .
```

```
    En Python, le passage se faisant par référence,
    il n'est pas indispensable de retourner le tableau .
```

```
    Keyword arguments:
```

```
    tab — liste de nombres
```

```
    """
```

```

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

python

## Estimation de la complexité

- Meilleur des cas, le tableau est trié, la complexité est linéaire :  $\mathcal{O}(n)$ .
- Pire des cas, le tableau est trié à l'envers, la complexité est quadratique :  $\mathcal{O}(n^2)$ .

## 5.3 Tri shell

```

def shellSort ( array ) :
    "Shell sort using Shell's ( original ) gap sequence: n/2, n/4, ..., 1."
    "http://en.wikibooks.org/wiki/Algorithm_Implementation/Sorting/Shell_sort#Python"
    gap = len(array) // 2
    # loop over the gaps
    while gap > 0:
        # do the insertion sort
        for i in range(gap, len(array)):
            val = array[i]
            j = i
            while j >= gap and array[j - gap] > val:
                array[j] = array[j - gap]
                j -= gap
            array[j] = val
        gap //= 2
    
```

python

## 5.4 Tri rapide «Quicksort»

### 5.4.1 Tri rapide

### Algorithme : Tri Quicksort – Segmentation

#### Données :

- tab, liste : une liste de nombres
- i, j, entiers : indices de début et de fin de la segmentation à effectuer

#### Résultats :

- tab, liste : la liste de nombre segmenté avec le pivot à sa place définitive
- k entier : l'indice de la place du pivot

**segmente**(tab, i, j) :

g ← i+1

d ← j

p ← tab[i]

**Tant que** g ≤ d **Faire**

**Tant que** d ≥ 0 et tab[d] > p **Faire**

        d ← d-1

**Fin Tant que**

**Tant que** g ≤ j et tab[g] ≤ p **Faire**

        g ← g+1

**Fin Tant que**

**Si** g < d **alors**

        Échange( tab, g, d )

        d ← d-1

        g ← g+1

**Fin Si**

**Fin Tant que**

k ← d

    Échange( tab, i, d )

**Retourner** k

### Algorithme : Tri Quicksort – Tri rapide

#### Données :

- tab, liste : une liste de nombres
- i, j, entiers : indices de début et de fin de la portion à trier

#### Résultats :

- tab, liste : liste triée entre les indices i et j

**tri\_quicksort**(tab, i, j) :

**Si** g < d **alors**

        k ← **segmente**(tab, i, j)

**tri\_quicksort**(tab, i, k-1)

**tri\_quicksort**(tab, k+1, j)

**Fin Si**

Pseudo Code

**def** **segmente**(tab, i, j) :

    """

    Segmentation d'un tableau par rapport à un pivot.

    Keyword arguments:

    tab ( list ) — liste de nombres

    i, j ( int ) — indices de fin et de début de la segmentation

    Retour :

    tab ( list ) — liste de nombres avec le pivot à sa place définitive

    k ( int ) — indice de la place du pivot

python

```

"""
g = i+1
d = j
p = tab[i]
while g <= d :
    while d >= 0 and tab[d] > p:
        d = d-1
    while g <= j and tab[g] <= p:
        g = g+1
    if g < d :
        tab[g], tab[d] = tab[d], tab[g]
        d = d-1
        g = g+1
k = d
tab[i], tab[d] = tab[d], tab[i]
return k

def tri_quicksort (tab, i, j):
    """
    Tri d'une liste par l' utilisation du tri rapide (Quick sort ).
    Keyword arguments:
    tab ( list ) — liste de nombres
    i, j ( int ) — indices de fin et de début de la zone de tri
    Retour :
    tab ( list ) — liste de nombres avec le pivot à sa place définitive
    """
    if i < j :
        k = segmente(tab, i, j)
        tri_quicksort (tab, i, k-1)
        tri_quicksort (tab, k+1, j)

```



## 5.4.2 Tri rapide optimisé

### Algorithme : Tri Quicksort – Tri rapide optimisé

#### Données :

- tab, liste : une liste de nombres
- i, j, entiers : indices de début et de fin de la portion de liste à trier

#### Résultats :

- tab, liste : liste triée entre les indices i et j

**tri\_quicksort\_optimized**(tab, i, j) :

Si i < j alors

k ← **segmente**(tab, i, j)

Si k - i > 15 alors

**tri\_quicksort**(tab, i, k - 1)

Sinon

**tri\_insertion**(tab, i, k - 1)

Fin Si

Si j - k > 15 alors

**tri\_quicksort**(tab, k + 1, j)

Sinon

**tri\_insertion**(tab, k + 1, j)

Fin Si

Fin Si



## 5.5 Tri fusion

---

### Algorithme : Tri Fusion – Fusion de deux listes

---

**Données :**

- tab, liste : une liste de nombres  $\text{tab}[g : d]$  avec  $g$  indice de la valeur de gauche,  $d$  indice de la valeur de droite
- $m$ , entier : indice tel que  $g \leq m < d$  et tel que les sous-tableaux  $\text{tab}[g : m]$  et  $\text{tab}[m+1 : d]$  soient ordonnés

**Résultats :**

- tab, liste : liste triée entre les indices  $g$  et  $d$

**fusion\_listes**(tab,  $g, d, m$ ) :

$n1 \leftarrow m - g + 1$

$n2 \leftarrow d - m$

**Initialiser tableau G**

**Initialiser tableau D**

**Pour**  $i$  **allant de** 1 **à**  $n1$  **faire**

$G[i] \leftarrow \text{tab}[g+i-1]$

**Fin Pour**

**Pour**  $j$  **allant de** 1 **à**  $n2$  **faire**

$D[j] \leftarrow \text{tab}[m+j]$

**Fin Pour**

$i \leftarrow 1$

$j \leftarrow 1$

$G[n1+1] \leftarrow +\infty$

$D[n2+1] \leftarrow +\infty$

**Pour**  $k$  **allant de**  $g$  **à**  $d$  **faire**

**Si**  $G[i] \leq D[j]$  **alors**

$\text{tab}[k] \leftarrow G[i]$

$i \leftarrow i + 1$

**Sinon**

**Si**  $G[i] > D[j]$  **alors**

$\text{tab}[k] \leftarrow D[j]$

$j \leftarrow j + 1$

**Fin Si**

**Fin Si**

**Fin Pour**

---

### Algorithme : Tri Fusion

Algorithme récursif du table de tri.

#### Données :

- tab, liste : une liste de nombres non triés tab[g :d]
- g,d, entiers : indices de début et de fin de la liste

#### Résultats :

- tab, liste : liste triée entre les indices g et d

**tri\_fusion**(tab,g,d) :

**Si** g<d **alors**

    m ← (g+d) div 2

**tri\_fusion**(tab,g,m)

**tri\_fusion**(tab,m+1,d)

**fusion\_listes**(tab,g,d,m)

**Fin Si**

```
def fusion_listes (tab,g,d,m):
    """
    Fusionne deux listes triées.
    Keyword arguments:
    tab ( list ) — liste : une liste de nombres tab[g:d] avec g indice de la
    valeur de gauche, d indice de la valeur de droite
    g,d,m (int) — entiers : indices tels que g<=m<d et tel que les
    sous-tableaux tab[g:m] et tab[m+1:d] soient ordonnés
    Résultat :
    tab ( list ) : liste triée entre les indices g et d
    """
    n1 = m-g+1
    n2 = d-m
    G,D = [],[]
    for i in range (n1):
        G.append(tab[g+i])
    for j in range (n2):
        D.append(tab[m+j+1])
    i,j=0,0
    G.append(999999999999)
    D.append(999999999999)
    for k in range (g,d+1):
        if G[i]<=D[j]: # and i<=n1
            tab[k]=G[i]
            i=i+1
        elif G[i]>D[j]: # and j<=n2
            tab[k]=D[j]
            j=j+1

def tri_fusion (tab,g,d):
    """
    Tri d'une liste par la méthode du tri fusion
    Keyword arguments:
    tab ( list ) — liste : une liste de nombres non triés tab[g:d]
    g,d (int) — entiers : indices de début et de fin de liste si on veut trier
    tout le tableau g=0, d=len(tab)-1
    Résultat :
    tab ( list ) : liste triée entre les indices g et d
    """
```



```
"""
if g<d:
    m=(g+d)//2
    tri_fusion (tab,g,m)
    tri_fusion (tab,m+1,d)
    fusion_listes (tab,g,d,m)
```

## 6 Algorithmes classiques

### 6.1 Division euclidienne

Pseudo Code

**Data :**  $a, b \in \mathbb{N}^*$   
 reste  $\leftarrow a$   
 quotient  $\leftarrow 0$   
**tant que** reste  $\geq b$  **faire**  
     reste  $\leftarrow$  reste  $- b$   
     quotient  $\leftarrow$  quotient  $+ 1$   
**fin**  
 Retourner quotient, reste

### 6.2 Algorithme d'Euclide

Cet algorithme permet de calculer le PGCD de deux nombres entiers. Il se base sur le fait que si  $a$  et  $b$  sont deux entiers naturels non nuls,  $\text{pgcd}(a, b) = \text{pgcd}(b, a \bmod b)$ .

Pseudo Code

Fonction PGCD : algorithme d'Euclide  
**Données :**  $a$  et  $b$  : deux entiers naturels non nuls  
 tels que  $a > b$   
**Résultat :** le PGCD de  $a$  et  $b$   
  
**Euclide\_PGCD(a,b)**  
     **Répéter**  
          $r \leftarrow a \bmod b$   
          $a \leftarrow b$   
          $b \leftarrow r$   
     **Jusqu'à**  $r == 0$   
     **Retourner**  $a$



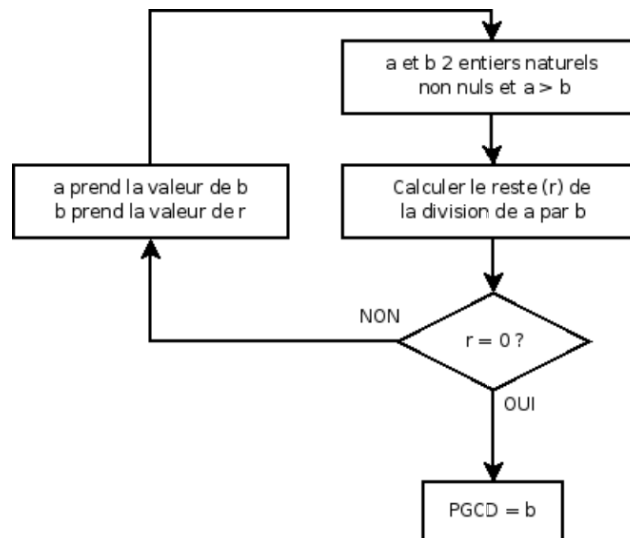
Codage en Python de l'algorithme d'Euclide :

```
def Euclide_PGCD(a,b): # on définit le nom de la
                        # fonction et ses variables
                        # d'entrées/d'appel
    r=a%b               # on calcule le reste dans
                        # la division de a par b

    while r!=0:         # tant que r est non nul :
        a=b             # b devient le nouveau a
        b=r             # r devient le nouveau b
        r=a%b           # on recalcule le reste

    return(b)           # une fois la boucle terminée,
                        # on retourne le dernier b

print(Euclide_PGCDpgcd(1525,755))
                        # on affiche le résultat
                        # retourné par la fonction
```



## 6.3 Calcul de puissance

### 6.3.1 Algorithme naïf

```

def exponentiation_naive(x,n):
    """
    Renvoie x**n par la methode naive.
    Keyword arguments:
    Entrées :
        x, flt : un nombre réel
        n, int : un nombre entier
    Sortie :
        res, flt : resultat
    """
    res = 1
    while n>=1:
        res = res * x
        n=n-1
    return res
  
```



### 6.3.2 Exponentiation rapide itérative

```

def exponentiation_rapide_iteratif(x,n):
    """
    Renvoie x**n par la methode d'exponentiation rapide.
    Keyword arguments:
    Entrées :
        x, flt : un nombre réel
        n, int : un nombre entier
    Sortie :
        res, flt : resultat
    """
    if n==0 :
  
```





```
    return 1
else :
    res = 1
    a = x
    while n>0:
        if n%2 == 1:
            res = res*a
        a=a*a
        n=int(n/2)
    return res
```

## Références

- [1] Patrick Beynet, Cours d'informatique de CPGE, Lycée Rouvière de Toulon, UPSTI.
- [2] Adrien Petri et Laurent Deschamps, Cours d'informatique de CPGE, Lycée Rouvière de Toulon.
- [3] Damien Iceta, Cours d'informatique de CPGE, Lycée Gustave Eiffel de Cachan, UPSTI.
- [4] Benjamin WACK, Sylvain CONCHON, Judicaël COURANT, Marc DE FALCO, Gilles DOWEK, Jean-Christophe FILLIÂTRE, Stéphane GONNORD, Informatique pour tous en classes préparatoires aux grandes écoles, Éditions Eyrolles.