Algorithmes usuels

```python
# Maximum d'une liste
def maximum(L):
    """ Retourne le maximum d'une liste L de nombres.
    L : Liste non vide de nombres """
    candidat = L[0]
    for i in range(1, len(L)):
        if L[i] > candidat:
            candidat = L[i]
    return candidat


# Recherche d'un élément dans une liste
def recherche1(L, x):
    """ Retourne True si l'élément x apparait dans la liste L,
        False sinon.
    (algorithme avec sortie de boucle)
    L : Liste quelconque
    x : Élément à rechercher dans la liste """
    for e in L:
        if e == x:
            return True
    return False


def recherche2(L, x):
    """ Retourne True si l'élément x apparait dans la liste L,
        False sinon.
    L : Liste quelconque
    x : Élément à rechercher dans la liste """
    trouve = False
    i = 0
    while not trouve and i < len(L):
        if L[i] == x:
            trouve = True
        i = i+1
    return trouve
```

```python
def comptage(L):
    """ L est une liste
    renvoie le dictionnaire dont les clés sont les éléments de L
    et les valeurs le nombre d'itérations dans L """
    dico = {}
    for x in L:
        if x in dico:
            dico[x] = dico[x] + 1
        else:
            dico[x] = 1
    return dico


# Première position d'un élément dans une liste
def prem_indice1(L, x):
    """ Retourne la première position de x dans la liste L,
    longueur de L si x est absent de la liste L.
    L : Liste quelconque
    x : Élément à rechercher dans la liste """
    for i in range(len(L)):
        if L[i] == x:
            return i
    return len(L)


def prem_indice2(L, x):
    """ Retourne la première position de x dans la liste L,
    longueur de L si x est absent de la liste L.
    L : Liste quelconque
    x : Élément à rechercher dans la liste """
    trouve = False
    i = 0
    while not trouve and i < len(L):
        if L[i] == x:
            trouve = True
            pos = i
        i = i+1
    if trouve:
        return pos
    else:
        return len(L)
```

```python
# dichotomie
def expo_rapide(x, n):
    if n == 0:
        return 1
    else:
        y = x * x
        if n % 2 == 0:
            return expo_rapide(y, n // 2)
        else:  # n impair
            return expo_rapide(y, n // 2) * x


# recherche dans une liste triée
def recherche_triee(L, x):
    """ Paramètres :
    L liste triée dans l'ordre croissant,
    x élément recherché
    Sortie : booléen indiquant si x apparait dans la liste L"""
    trouve = False
    # on cherche dans L entre debut inclus et fin exclus
    debut = 0
    fin = len(L)
    while debut < fin and not trouve:
        milieu = (debut + fin) // 2  # dans [debut,fin[
        if x == L[milieu]:
            trouve = True
        elif x < L[milieu]:
            fin = milieu
        else:  # x > L[milieu]
            debut = milieu+1
    return trouve


def recherche_triee_inter(L, x, debut, fin):
    """ L liste triée dans l'ordre croissant
    recherche si x apparait dans la liste L entre les indices
        debut et fin"""
    if debut >= fin:
        return False
    else:
        m = (debut + fin) // 2
        if x == L[m]:
            return True
        elif x < L[m]:
            return recherche_triee_inter(L, x, debut, m)
        else:  # x > L[m]
            return recherche_triee_inter(L, x, m + 1, fin)
```

```python
def recherche_triee_rec(L, x):
    """ L liste triée dans l'ordre croissant,
    x élément recherché
    renvoie : booléen indiquant si x apparait dans la liste L"""
    return recherche_triee_inter(L, x, 0, len(L))


# recherche d'un motif
def sous_chaine(texte, mot, debut):
    """ Entrée : texte et mot sont des chaines de caractères,
    debut est un entier.
    Sortie : renvoie True si mot est la sous chaine extraite de
        texte
    à partir de la position debut.
    On suppose que debut + len(mot) <= len(texte) """
    for k in range(len(mot)):
        if texte[debut + k] != mot[k]:
            return False
    return True


def recherche_motif1(texte, mot):
    """Entrée : 2 chaines de caractères mot et texte
    Sortie : booléen indiquant si mot est inclus dans texte"""
    for debut in range(len(texte) - len(mot) + 1):
        if sous_chaine(texte, mot, debut):
            return True
    return False


def recherche_motif2(mot, texte):
    """Entrée : 2 chaines de caractères mot et texte
    Sortie : booléen indiquant si mot est inclus dans texte"""
    trouve = False
    n = len(texte)
    p = len(mot)
    debut = 0
    while (not trouve) and debut + p <= n:
        # on compare mot et la tranche texte[debut:debut + p]
        k = 0
        ok = True  # mot coincide sur les k premiers caractères
        while ok and k < p:
            ok = texte[debut + k] == mot[k]
            k = k + 1
        trouve = ok
        debut = debut + 1
    return trouve
```

```python
# Tris
def tri_insertion(L):
    """ tri les éléments de L dans l'ordre croissant (procédure)
    L est une liste d'objets comparables par < """
    for k in range(1, len(L)):
        x = L[k]
        # chercher l'emplacement de x et décaler la queue
        i = k
        while i > 0 and L[i - 1] > x:  # évaluation paresseuse
            L[i] = L[i - 1]
            i = i - 1
        # placer x
        L[i] = x
    return None


def tri_selection(L):
    """ tri les éléments de L dans l'ordre croissant (procédure)
    L est une liste d'objets comparables par < """
    n = len(L)
    for i in range(n - 1):
        min = i
        for j in range(i, n):
            if L[j] < L[min]:
                min = j
        L[i], L[min] = L[min], L[i]
    return None


def fusion(G, D):
    i, j = 0, 0
    res = []
    while i < len(G) and j < len(D):
        if G[i] < D[j]:
            res.append(G[i])
            i = i + 1
        else:
            res.append(D[j])
            j = j + 1
    if i == len(G):
        res.extend(D[j:])
    else:
        res.extend(G[i:])
    return res
```

```python
def tri_fusion(A):
    n = len(A)
    if n < 2:
        return A
    else:
        G = tri_fusion(A[:n//2])
        D = tri_fusion(A[n//2:])
        return fusion(G, D)


def partition(A):
    x = A[0]
    inf, sup = [], []
    for k in range(1, len(A)):
        if A[k] <= x:
            inf.append(A[k])
        else:
            sup.append(A[k])
    return inf, x, sup


def tri_rapide(A):
    n = len(A)
    if n < 2:
        return A
    else:
        inf, x, sup = partition(A)
        inf = tri_rapide(inf)
        sup = tri_rapide(sup)
        return inf + [x] + sup


def tri_comptage(L, sup):
    """ tri les éléments de L dans l'ordre croissant (procédure)
    L est une liste d'entiers compris entre 0 et sup (exclu) """
    # compter les éléments
    comptage = [0] * sup
    for x in L:
        comptage[x] = comptage[x] + 1
    # liste triée
    i = 0
    for x in range(sup):
        for k in range(comptage[x]):
            L[i] = x
            i = i + 1
    return None
```

```python
# en plus pour la culture : tri rapide en place et k ième élément
def partition_2(A, p, q):
    """ partition en place """
    x = A[p]
    i = p
    for j in range(p + 1, q):
        if A[j] <= x:
            i = i + 1
            A[i], A[j] = A[j], A[i]
    A[p], A[i] = A[i], A[p]
    return i


def quicksort_inter(A, p, q):
    """ tri A entre les indices p et q"""
    if p < q:
        r = partition_2(A, p, q)
        quicksort_inter(A, p, r)
        quicksort_inter(A, r + 1, q)
    return None


def quicksort(A):
    """ tri A en place """
    quicksort_inter(A, 0, len(A))


def quick_k(A, p, q, k):
    """
    renvoie le k ième plus petit élément des éléments de A
    dont l'indice est compris entre p et q
    """
    if p >= q - 1:
        return A[p]
    else:
        r = partition_2(A, p, q)
        i = r - p  # rang de A[r] dans la partie
        if i == k:
            return A[r]
        elif k < i:
            return quick_k(A, p, r, k)
        else:  # k > i
            return quick_k(A, r + 1, q, k - i - 1)
```

```python
## Lecture dans un fichier
# le fichier tableau.csv se trouve dans le répertoire courant
# on veut sommer les nombres qui apparaissent dans la 2e colonne
# les colonnes sont séparés par des tabulations : '\t'

somme = 0
with open('tableau.csv', 'r') as fichier:
    for ligne in fichier:
        cellules = ligne.split('\t')  # liste de str
        somme = somme + float(cellules[1])  # transtypage et +

## pour ouvrir un fichier, on peut aussi procéder ainsi :
fichier = open('tableau.csv', 'r')
# traitement
fichier.close()

## pour accéder au contenu du fichier :
# pour tout avoir dans une chaîne de caractères
contenu = fichier.read()
# pour lire une ligne et passer à la suivante
ligne = fichier.readline()
# pour avoir la liste de toutes les lignes
liste_lignes = fichier.readlines()

## Écriture dans un fichier : on l'ouvre avec l'argument 'w'
fichier = open('tableau2.csv', 'w')
somme = 0
for k in range(10):
    somme = somme + k**2
    fichier.write(str(k) + '\t' + str(somme) + '\n')
fichier.close()
```