```python
# /home/jl/Dropbox/Cours/0MPSI/2017/Info-MPstar/Cours/recursivite.py
## Maximum d'une liste

# Première possibilité

def max1(L):
    "renvoie le maximum de L"
    if len(L) == 1:
        return L[0]
    a = L.pop()
    m = max1(L)
    L.append(a) # C'est mieux si L reste intacte !
    if a > m:
        return a
    return m

# Complexités spatiale et temporelle linéaire.

# Deuxième possibilité

def max2(L, debut=0):
    "renvoie le maximum de L[debut:]"
    if debut == len(L) - 1:
        return L[-1]
    m = max2(L, debut + 1)
    if L[debut] > m:
        return L[debut]
    return m

# Complexités spatiale et temporelle linéaire.

## Suite de Syracuse

# Première possibilité

def syracuse(n, u0):
    "retourne le terme d'indice n de la suite de Syracuse itérée à
partir de u0"
    if n == 0:
        return u0
    u = syracuse(n - 1, u0)
    if u % 2 == 0:
        return u // 2
    return 3*u + 1

# Deuxième version (récursive terminale)

def syracuse2(n, u0):
    "retourne le terme d'indice n de la suite de Syracuse itérée à
partir de u0"
    if n == 0:
        return u0
    if u0 % 2 == 0:
        return syracuse2(n - 1, u0 // 2)
    return syracuse2(n - 1, 3*u0 + 1)

# Complexités spatiale et temporelle linéaires

## Suite de Héron

```

```python
# Première possibilité

def Heron1(n):
    "retourne le terme d'indice n de la suite de Héron"
    if n == 0:
        return 1
    u = Heron1(n - 1)
    return (u**2 + 2) / (2*u)

# Deuxième version (récursive terminale)

def Heron2(n, u=1):
    "retourne le terme d'indice n de la suite de Héron"
    if n == 0:
        return u
    return Heron2(n - 1, (u**2 + 2) / (2*u))

# Complexités spatiale et temporelle linéaires

## Algorithme d'Euclide étendu

# Si a = bq + r et si b·u + r·v = d alors a·v + b·(u - q·v) = d.

def euclide_etendu(a, b):
    "renvoie (d, u, v) tel que a·u + b·v = d = pgcd(a, b)"
    if b == 0:
        return (a, 1, 0)
    q, r = divmod(a, b)
    d, u, v = euclide_etendu(b, r)
    return (d, v, u - q*v)

# Complexités spatiale et temporelle : voir théorème de Lamé
# Terminaison : la suite des restes est entière naturelle strictement décroissante

## Méthode de Newton

# Version nombre de termes

def newton(f, df, x0, n):
    "terme d'indice n dans la méthode de Newton appliquée à f"
    if n == 0:
        return x0
    return newton(f, df, x0 - f(x0) / df(x0), n - 1)

# Complexités spatiale et temporelle linéaires

# Version erreur

def newton2(f, df, x0, epsilon):
    "terme dans la méthode de Newton appliquée à f à epsilon près du terme précédent"
    x1 = x0 - f(x0) / df(x0)
    if abs(x1 - x0) < epsilon:
        return x1
    return newton2(f, df, x1, epsilon)

# Problème de terminaison si ça ne converge pas.

```

```
116|
117| ## Dichotomie
118|
119| # Version nombre de termes
120|
121| def dicho(f, a, b, n):
122|     "terme d'indice n dans la recherche dichotomique d'un zéro de f"
123|     assert f(a) * f(b) <= 0, "Hypothèses non vérifiées"
124|     c = (a + b) / 2
125|     if n == 0:
126|         return c
127|     if f(a) * f(c) > 0:
128|         return dicho(f, c, b, n - 1)
129|     return dicho(f, a, c, n - 1)
130|     # /!\ cas traité également dans le cas où f(c) = 0.
131|
132|
133|
134|
135| # Version erreur
136|
137| def dicho2(f, a, b, epsilon):
138|     "terme d'indice n dans la recherche dichotomique d'un zéro de f"
139|     assert f(a) * f(b) <= 0, "Hypothèses non vérifiées"
140|     c = (a + b) / 2
141|     if abs(b - a) < 2 * epsilon:
142|         return c
143|     if f(a) * f(c) > 0:
144|         return dicho2(f, c, b, epsilon)
145|     return dicho2(f, a, c, epsilon)
146|     # /!\ cas traité également dans le cas où f(c) = 0.
147|
148|
149|
150|
151| ## Recherche dichotomique
152|
153| # Première version
154|
155| def rech_dicho(T, x):
156|     """position de x dans le tableau trié T par recherche dichotomique,
157| -1 si x n'est pas dedans"""
158|     if T == []:
159|         return -1 # Cas d'un tableau vide à traiter à part
160|
161|     def rech_aux(T, x, debut, fin):
162|         "recherche dichotomique de x dans T[debut: fin] (convention
python)"
163|         if debut == fin - 1:
164|             if T[debut] == x:
165|                 return debut
166|             return -1
167|         milieu = (debut + fin) // 2
168|         if T[milieu] <= x:
169|             return rech_aux(T, x, milieu, fin)
170|         return rech_aux(T, x, debut, milieu)
171|
172|     return rech_aux(T, x, 0, len(T))
173|
174|
```

```python

# Deuxième version

def rech_dicho2(T, x):
    """position de x dans le tableau trié T par recherche dichotomique,
-1 si x n'est pas dedans"""
    if T == []:
        return -1 # Cas d'un tableau vide à traiter à part

    def rech_aux(T, x, debut, fin):
        "recherche dichotomique de x dans T[debut], ... , T[fin]"
        if debut > fin:
            return -1
        milieu = (debut + fin) // 2
        if T[milieu] == x:
            return milieu
        if T[milieu] < x:
            return rech_aux(T, x, milieu + 1, fin)
        return rech_aux(T, x, debut, milieu - 1)

    return rech_aux(T, x, 0, len(T) - 1)

## Numérotation des couples

# On numérote par diagonale :
# y
#   ▲
#   |
# 4 |  14
# 3 |   9  13
# 2 |   5   8  12
# 1 |   2   4   7  11  ..
# 0 |   0   1   3   6  10  15
#   +-------------------------> x
#     0   1   2   3   4   5
#
# Si (x, y) n'est pas au pied d'une diagonale (y != 0),
# num(x, y) = num (x + 1, y - 1) + 1
# Sinon num(x, 0) = 0 si x = 0 et num(x - 1, 0) + x sinon

def num(x, y):
    "umérotation par diagonales de IN²"
    if y == 0:
        if x == 0:
            return 0
        return num(x - 1, 0) + x
    return num(x + 1, y - 1) + 1

##

# Pour la suite, voir dans les fichiers joints.
# https://prepa-carnot.fr/mpstar/index.php/69
```