Tri par insertion

1. Présentation

```
def tri_debut(A, i):
    V = A[i]
    while V < A[i-1] and i-1 >=0:
        A[i] = A[i-1]
        i-=1
    A[i] = V
```

Pour des raisons de praticité, on décide de créer une classe qui aura comme méthode publique triInsertion et comme méthode privée triDebut.

```
import numpy as np
import matplotlib.pylab as plt
import time


class Liste(list):

    def __init__(self):
        self.A = []

    def generateRadomList(self, n):
        """n: lenght of list to generate"""
        self.A = [np.random.randint(n) for k in range(n)]

    def _triDebut(self, i):
        V = self.A[i]
        while V < self.A[i-1] and i-1 >= 0:
            self.A[i] = self.A[i-1]
            i -= 1
        self.A[i] = V

    def trier(self):
        for i in range(1, len(self.A)):
            self._triDebut(i)

    def time(self, n, show=False):
        X, T = [], []
        for i in range(1, n):
            self.generateRadomList(i)
            start = time.time()
            self.trier()
            end = time.time()

            X.append(i)
```

```
            T.append((end - start) / i ** 2)
        if (show):
            plt.plot(X, T, label='temps / $n^2')
            plt.show()
```

2. Tri insertion

```
def triInsertion(A, i):
   for i in range(1, len(self.A)):
            self._triDebut(i)
```

3. Complexité

La complexité et c = $O(n^2)$

4. Complexité

```
from Ressources.list import Liste
A = Liste()
A.time(1000, True)
```

5. Remplissage d'ellipse

Ressources/Ellipse.py

```
import numpy as np
import matplotlib.pylab as plt
from Ressources.liste import Liste
import cmath


class Ellipse():
    def __init__(self):
        self.X = []
        self.Y = []
        self.Filled = []
        self.conditionInitales = (0, 0)
        self.Eulerized = False

    def Eulerize(self, n):
        h = 2*np.pi/n
        self.X = [self.conditionInitales[0]]
        self.Y = [self.conditionInitales[1]]

        for i in range(n):
            self.X.append(self.X[i] + h * (2*self.X[i]-5*self.Y[i]))
```

```
            self.Y.append(self.Y[i] + h * (self.X[i]-2*self.Y[i]))
        self.Eulerized = True

    def Tracer(self):
        if (self.Eulerized):
            l = len(self.Filled)
            if (l > 0):
                colonne = int(np.sqrt(l))
                ligne = l // colonne
                ligne = ligne if l % colonne == 0 else ligne + 1
                for i in range(l):
                    plt.subplot(colonne, ligne, i+1)
                    plt.plot(self.Filled[i][0], self.Filled[i][1])
                    plt.plot(self.X, self.Y)
            plt.show()
        else:
            raise NameError('Veuillez d\'abord lancer la procédeure d\'Euler')

    def GetListOfPoint(self, n, inverted=False, radial=False):
        A = []
        l = len(self.X)
        for i in range(0, l, l//n):
            if inverted:
                A.append([self.Y[i], self.X[i]])
            else:
                A.append([self.X[i], self.Y[i]])
        return A

    def Fill(self, pas, methode):
        if (methode == 0):
            X, Y = [0], [0]
        elif (methode == 1):
            A = Liste(self.GetListOfPoint(pas))
            A.trier()
            X, Y = A.separate()
        elif (methode == 2):
            A = Liste(self.GetListOfPoint(pas, True))
            A.trier()
            Y, X = A.separate()
        elif (methode == 3):
            A = Liste()
            for i in range(0, len(self.X), pas):
                r, tetha = cmath.polar(complex(self.X[i], self.Y[i]))
                A.append([tetha, r])
            A.trier()
            X, Y = [], []
            for a in A:
                X.append(a[1] * np.cos(a[0]))
                Y.append(a[1] * np.sin(a[0]))
                X.append(0)
                Y.append(0)
        self.Filled.append([X, Y])
```

main.py

```
from Ressources.ellipse import Ellipse
import matplotlib.pylab as plt

E = Ellipse()

E.conditionInitales = (1, 2)

E.Eulerize(1000)

E.Fill(25, methode=0)
E.Fill(25, methode=1)
E.Fill(25, methode=2)
E.Fill(25, methode=3)

E.Tracer()
```

6. Rotation

On rajoute deux méthodes à la classe Elipse :

Ressources/Ellipse.py

```
    def GetPolar(self):
        rMax = 0
        phiMax = 0
        for i in range(len(self.X)):
            r, phi = cmath.polar(complex(self.X[i], self.Y[i]))
            if r > rMax:
                rMax = r
                phiMax = phi
        return (rMax, phiMax)

    def Rotate(self, t):
        R = np.array([
            [np.cos(t), -np.sin(t)],
            [np.sin(t), np.cos(t)]
        ])
        X, Y = [], []
        for i in range(len(self.X)):
            u = np.array([self.X[i], self.Y[i]])
            v = np.dot(R, u)
            X.append(v[0])
            Y.append(v[1])
        self.Filled.append([X, Y])
```

main.py

```
from Ressources.ellipse import Ellipse

E = Ellipse()

E.conditionInitales = (1, 2)

E.Eulerize(1000)

r, t = E.GetPolar()

E.Rotate(-t)
E.Tracer()
```

7. Aire

Ressources/Ellipse.py

```
    def GetGrandRayon(self):
        X = np.array(self.X)
        Y = np.array(self.Y)
        return(max(np.sqrt(X**2 + Y**2)))

    def GetPetitRayon(self):
        X = np.array(self.X)
        Y = np.array(self.Y)
        return(min(np.sqrt(X**2 + Y**2)))

    def GetAire(self, t):
        A = []
        aire = 0
        R = np.array([
            [np.cos(t), -np.sin(t)],
            [np.sin(t), np.cos(t)]
        ])
        for i in range(len(self.X)):
            u = np.array([self.X[i], self.Y[i]])
            v = np.dot(R, u)
            if(v[0] > 0 and v[1] > 0):
                A.append(v)

        for i in range(len(A)-1):
            aire += A[i+1][1] * abs(A[i+1][0] - A[i][0])
        return aire * 4
```

main.py

```
from Ressources.ellipse import Ellipse
import numpy as np
E = Ellipse()

E.conditionInitales = (1, 2)

E.Eulerize(1000)

a = E.GetGrandRayon()
b = E.GetPetitRayon()

r, t = E.GetPolar()
A1 = E.GetAire(-t)
A2 = np.pi*a*b

print(A1)
print(A2)
print(abs(A2-A1)/A2)
```