```python
# TP12-c.py

import random as rd
import matplotlib.pyplot as plt

# Première partie : Le module Random

def fonction1(n):
    L=[]
    for i in range (0,n):
        L.append(rd.randint(1,n))
    return(L)

def fonction2(n):
    L=[]
    for i in range (0,n):
        L.append(rd.randint(1,n))
    L.sort()
    return(L)

def fonction3(n):
    L=[]
    for i in range (0,n):
        L.append(rd.uniform(-10,10))
    L.sort()
    return(L)

# Deuxième partie : Complexité quadratique vs complexité linéaire

# 2.1. Évaluation naïve d'un polynôme

def evaluation1_modif(A,x):
    n = len(A)
    s = 0.0
    compteur = 0
    for k in range(n):
        p = 1.0
        for i in range(k):
            p = p * x
            compteur+=1
        s = s + A[k] * p
        compteur+=2
    return(s,compteur)

def fonction4(n):
    d=rd.randint(0,(n-1))
    A=[]
    for i in range (0,d):
        A.append(rd.uniform(-5,5))
    x=rd.uniform(-2,2)
    a,b=evaluation1_modif(A,x)
    return(b)

def fonction5(N):
    Y=[]
    X=[]
    for n in range(1,N+1):
        X.append([n])
        A=[]
        for i in range (0,n):
```

```
059|             A.append(rd.uniform(-5,5))
060|         x=rd.uniform(-2,2)
061|         a,b=evaluation1_modif(A,x)
062|         Y.append([b])
063|     plt.plot(X,Y)
064|     plt.show()
065|
066| # 2.2. Amélioration de l'algorithme
067|
068| def evaluation2_modif(A,x):
069|     n = len(A)
070|     s = 0.0
071|     p = 1.0
072|     compteur=0
073|     for k in range(n):
074|         s = s + A[k] * p
075|         p = p * x
076|         compteur+=3
077|     return(s,compteur)
078|
079| def fonction6(n):
080|     d=rd.randint(0,(n-1))
081|     A=[]
082|     for i in range (0,d):
083|         A.append(rd.uniform(-5,5))
084|     x=rd.uniform(-2,2)
085|     a,b=evaluation2_modif(A,x)
086|     return(b)
087|
088| def fonction7(N):
089|     Y=[]
090|     X=[]
091|     for n in range(1,N+1):
092|         X.append([n])
093|         A=[]
094|         for i in range (0,n):
095|             A.append(rd.uniform(-5,5))
096|         x=rd.uniform(-2,2)
097|         a,b=evaluation2_modif(A,x)
098|         Y.append([b])
099|     plt.plot(X,Y)
100|     plt.show()
101|
102| # 2.3. Algorithme de Hörner
103| def evaluation3_modif(A,x):
104|     n = len(A)
105|     s = 0.0
106|     compteur=0
107|     for k in range(n-1,-1,-1):
108|         s = s * x + A[k]
109|         compteur+=2
110|     return(s,compteur)
111|
112| def fonction8(n):
113|     d=rd.randint(0,(n-1))
114|     A=[]
115|     for i in range (0,d):
116|         A.append(rd.uniform(-5,5))
117|     x=rd.uniform(-2,2)
118|     a,b=evaluation3_modif(A,x)
```

```
    return(b)

def fonction9(N):
    Y=[]
    X=[]
    for n in range(1,N+1):
        X.append([n])
        A=[]
        for i in range (0,n):
            A.append(rd.uniform(-5,5))
        x=rd.uniform(-2,2)
        a,b=evaluation3_modif(A,x)
        Y.append([b])
    plt.plot(X,Y)
    plt.show()

# Troisième partie : Complexité linéaire vs complexité logarithmique

# Schéma linéaire

def recherche_lineaire(L,x):
    i=0
    while i<len(L):
        if L[i]==x:
            return(i)
        else:
            i+=1
    return(i)

def fonction10(n):
    L=fonction1(n)
    return(recherche_lineaire(L,0))

def fonction11(N):
    X=[]
    Y=[]
    for n in range(1,N+1):
        X.append(n)
        Y.append(fonction10(n))
    plt.plot(X,Y)
    plt.show()

def fonction12(n,NB):
    som=0
    for i in range(1,NB):
        L=fonction1(n)
        x=rd.randint(0,n)
        som+=recherche_lineaire(L,x)
    return(som/NB)

def fonction13(N,NB):
    X=[]
    Y=[]
    for n in range(1,N+1):
        X.append(n)
        Y.append(fonction12(n,NB))
    plt.plot(X,Y)
    plt.show()

# Schéma logarithmique
```

```
179|
180|
181| def recherche_dichotomique(L,x):
182|     deb=0
183|     fin=len(L)-1
184|     compteur=0
185|     while deb<fin:
186|         m=(deb+fin)//2
187|         if x==L[m]:
188|             compteur+=1
189|             return(True,compteur)
190|         elif x<L[m]:
191|             compteur+=2
192|             fin=m-1
193|         else:
194|             compteur+=2
195|             deb=m+1
196|     if x==L[deb]:
197|         compteur+=1
198|         return(True,compteur)
199|     else:
200|         compteur+=1
201|         return(False,compteur)
202|
203| def fonction14(n):
204|     L=fonction2(n)
205|     a,b=recherche_dichotomique(L,0)
206|     return(b)
207|
208| def fonction15(N):
209|     X=[]
210|     Y=[]
211|     for n in range(1,N+1):
212|         X.append(n)
213|         Y.append(fonction14(n))
214|     plt.plot(X,Y)
215|     plt.show()
```