TP02.py

```python
import random, time, matplotlib.pyplot as plt, numpy as np

########0######
# Exercice 01 #
###############

def suite_u(n):
    if n == 0:
        return 1
    return n * suite_u(n - 1)

def suite_v(n):
    if n == 0:
        return 3
    return 2 * suite_v(n - 1) + 1

def suite_w(n):
    if n == 0:
        return 5
    return 2 * (n - 1) * suite_w(n - 1) + 3

def suite_t(n):
    if n == 0:
        return 0
    return 1 / n**2 + suite_t(n - 1)

########0######
# Exercice 02 #
###############

def expo(x, n):
    if n == 0:
        return 1
    return x * expo(x, n - 1)

def expo_rapide(x, n):
    if n == 0:
        return 1
    if n % 2 == 0:
        return expo_rapide(x * x, n // 2)
    return x * expo_rapide(x, n - 1)

########0######
# Exercice 03 #
###############

def chiffres(x, b):
    if x == 0:
        return []
    q, r = x // b, x % b
    return chiffres(q, b) + [r]

def valeur(liste, b):
    if liste == []:
        return 0
    return b * valeur(liste[: - 1], b) + liste[-1]

def convertit(liste, b1, b2):
    return chiffres(valeur(liste, b1), b2)
```

```python
########0######
# Exercice 04 #
###############

def fusionne(s, t):
    u = []
    i = j = 0
    while i < len(s) and j < len(t):
        if s[i] <= t[j]:
            u.append(s[i])
            i += 1
        else:
            u.append(t[j])
            j += 1
    for k in range(i, len(s)):
        u.append(s[k])
    for k in range(j, len(t)):
        u.append(t[k])
    return u

# On peut éviter les deux boucles for en faisant
# return u + s[i:] + t[j:]

def tri_fusion(s):
    n = len(s)
    if n < 2:
        return s
    ga, dr = s[ : n // 2], s[n // 2 : ]
    return fusionne(tri_fusion(ga), tri_fusion(dr))

def tri_selection(s):
    for k in range(len(s) - 1):
        i_min = k
        for i in range(k, len(s)):
            if s[i] < s[i_min]:
                i_min = i
        s[k], s[i_min] = s[i_min], s[k]

def temps(f, expo_max):
    N = []
    T = []
    for k in range(3, expo_max + 1):
        s = [random.random() for _ in range(2**k)]
        t0 = time.clock()
        f(s)
        dt = time.clock() - t0
        N.append(2**k)
        T.append(dt)
    return N, T

def trace_graphe(f1, n1, f2, n2):
    N1, T1 = temps(f1, n1)
    N2, T2 = temps(f2, n2)
    plt.loglog(N1, T1, linestyle = "--", label = f1.__name__)
    plt.loglog(N2, T2, label = f2.__name__)
    plt.xlabel("n")
    plt.ylabel("t (en secondes)")
    plt.legend(loc = 'upper left')
    plt.grid()
    plt.savefig("tris.png")
```

TP02.py

```
########0######
# Exercice 05 #
###############

def evalue(expr):
    t = expr.split()
    pile = []
    for x in t:
        if x == '+':
            a = pile.pop()
            b = pile.pop()
            pile.append(a + b)
        elif x == '*':
            a = pile.pop()
            b = pile.pop()
            pile.append(a * b)
        elif x == '-':
            a = pile.pop()
            b = pile.pop()
            pile.append(b - a)
        else:
            pile.append(int(x))
    resultat = pile.pop()
    if pile == []:
        return resultat
    else:
        return "Erreur : expression mal formée"


########0######
# Exercice 07 #
###############

def interpole(x, t, deb, fin):
    if deb >= fin:
        return -1
    if t[deb] == t[fin - 1]:
        if t[deb] == x:
            return deb
        return -1
    coupure = deb + (x - t[deb]) // (t[fin - 1] - t[deb]) * (fin - 1 - deb)
    if t[coupure] == x:
        return coupure
    if t[coupure] < x:
        return interpole(x, t, coupure + 1, fin)
    if t[coupure] > x:
        return interpole(x, t, deb, coupure)

def cherche_interpole(x, t):
    return interpole(x, t, 0, len(t))

def compte_interpole(x, t, deb, fin):
    if deb >= fin or x < t[deb] or x > t[fin - 1]:
        return (-1, 0)
    if t[deb] == t[fin - 1]:
        if t[deb] == x:
            return (deb, 0)
        return (-1, 0)
    coupure = deb + ((x - t[deb]) * (fin - 1 - deb)) // (t[fin - 1] - t[deb])
    if t[coupure] == x:
        return (coupure, 0)
    if t[coupure] < x:
        indice, compteur = compte_interpole(x, t, coupure + 1, fin)
        return (indice, compteur + 1)
    if t[coupure] > x:
        indice, compteur = compte_interpole(x, t, deb, coupure)
        return (indice, compteur + 1)


def compte_dicho(x, t, deb, fin):
    if deb >= fin:
        return (-1, 0)
    mil = (deb + fin) // 2
    if t[mil] == x:
        return (mil, 0)
    if t[mil] < x:
        indice, compteur = compte_dicho(x, t, mil + 1, fin)
        return (indice, compteur + 1)
    if t[mil] > x:
        indice, compteur = compte_dicho(x, t, deb, mil)
        return (indice, compteur + 1)


def compare_methodes(t):
    n = max(t) + 1 - min(t)
    max_d = moy_d = 0
    max_i = moy_i = 0
    for x in range(min(t), max(t) + 1):
        i_d, c_d = compte_dicho(x, t, 0, len(t))
        i_i, c_i = compte_interpole(x, t, 0, len(t))
        if (i_d == -1 and i_i != -1) or (i_d != -1 and i_i == -1) \
        or (i_d != -1 and i_i != -1 and t[i_d] != t[i_i]):
            print(i_d, i_i, t[i_d], t[i_i])
            return "Erreur pour " + str(x)
        max_d = max(max_d, c_d)
        max_i = max(max_i, c_i)
        moy_d += c_d / n
        moy_i += c_i / n
    print("Pour la dichotomie simple : moyenne {}, max {}".format(moy_d, max_d))
    print("Pour l'interpolation : moyenne {}, max {}".format(moy_i, max_i))

def compare(n_elts, borne_inf, borne_sup):
    t = [random.randint(borne_inf, borne_sup) for k in range(n_elts)]
    t.sort()
    compare_methodes(t)

# In [37]: compare(5 * 10**4, 10**6, 2 * 10**6)
# Pour la dichotomie simple : moyenne 15.592511033007591, max 16
# Pour l'interpolation : moyenne 3.39020211991738, max 12


########0######
# Exercice 08 #
###############

def hanoi_aux(n, init, final, autre):
    if n > 0:
        hanoi_aux(n - 1, init, autre, final)
        print("Déplacer un disque de {} vers {}".format(init, final))
        hanoi_aux(n - 1, autre, final, init)

def hanoi(n):
    hanoi_aux(n, 1, 3, 2)
```