Guion 5:

Actividad 1:

A) Analizar la Complejidad Temporal del algoritmo.

El algoritmo tiene una complejidad temporal de O(n^2).

B) Implementar el algoritmo de Programación Dinámica, probando su buen funcionamiento para otros ejemplos.

```java
/**
 * Constructor
 * @param cad1
 * @param cad2
 */
public Levenshtein(String cad1, String cad2) {
    this.cad1=cad1;
    this.cad2=cad2;
    distanciaLevenshtein(cad1, cad2);
    showResult();
}

/**
 * Método que calcula la distancia de Levenshtein
 * @param cad1
 * @param cad2
 */
private void distanciaLevenshtein(String cad1, String cad2) {
    matrix = new int[cad2.length()+1][cad1.length()+1];
    prepareMatrix();
    for (int i=1; i<matrix.length; i++) {
        for (int j=1; j<matrix[i].length; j++) {
            if (cad2.charAt(i-1) == cad1.charAt(j-1)) {
                matrix[i][j] = matrix[i-1][j-1];
            } else if (cad2.charAt(i-1) != cad1.charAt(j-1)) {
                matrix[i][j] = 1 + min(matrix[i-1][j-1], matrix[i][j-1], matrix[i-1][j]);
            }
        }
    }
    result = matrix[cad2.length()][cad1.length()];
}

/**
 * Método que calcula el minimo
 * @param a
 * @param b
 * @param c
 * @return minimo
 */
private int min(int a, int b, int c) {
    return Math.min(Math.min(a, b), c);
}

/**
 * Método que prepara la matriz
 */
private void prepareMatrix() {
    for (int i=0; i < matrix.length; i++) {
        matrix[i][0]=i;
    }
    for (int j=0; j<matrix[0].length; j++) {
        matrix[0][j]=j;
    }
}
```

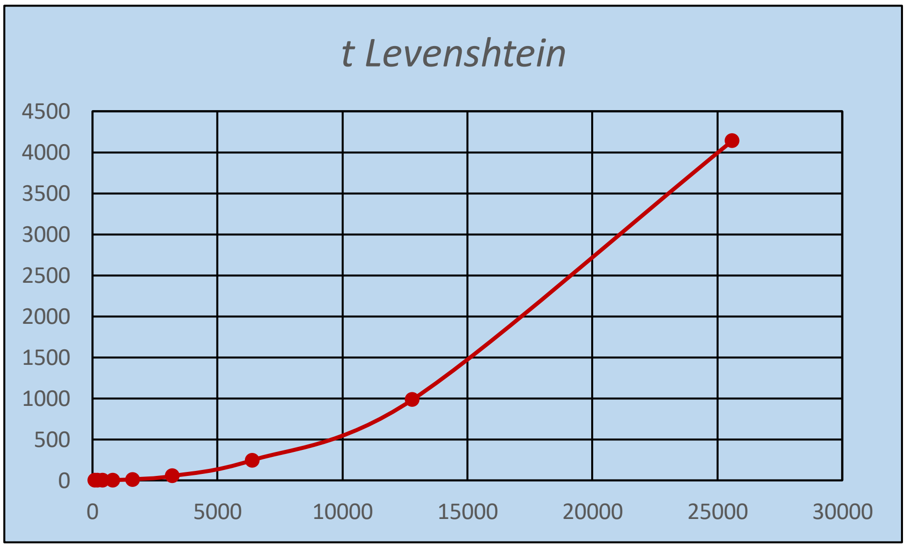
```
Distancia de Levenshtein:

       B   A   R   C   A   Z   A   S
   0   1   2   3   4   5   6   7   8
A  1   1   1   2   3   4   5   6   7
B  2   1   2   2   3   4   5   6   7
R  3   2   2   2   3   4   5   6   7
A  4   3   2   3   3   3   4   5   6
C  5   4   3   3   3   4   4   5   6
A  6   5   4   4   4   3   4   4   5
D  7   6   5   5   5   4   4   5   5
A  8   7   6   6   6   5   5   4   5
B  9   8   7   7   7   6   6   5   5
R  10  9   8   7   8   7   7   6   6
A  11  10  9   8   8   8   8   7   7

El resultado es: 7
```

C) Medir tiempos poniendo ambas longitudes iguales (luego n=m), dando valor a las cadenas de forma aleatoria y creciendo el tamaño de problema así (n= 100, 200, 400, 800, 1600,… etc.). Hay que rellenar la tabla de tiempos correspondiente y razonar si cumple o no la Complejidad hallada.

N	*t Levenshtein*
100	0,1030
200	0,2640
400	0,8520
800	3,3760
1600	13,451
3200	53,649
6400	246,40
12800	983,80
25600	4139,5
Complejidad	***O(n²)***



Tomamos $n_1 = 12800$ y $n_2 = 25600$, con $t_1 = 983{,}80$, para demostrar que la complejidad del algoritmo es $O(n^2)$ aplicamos la siguiente formula:

$$t_2 = \frac{f(n_2)}{f(n_1)} * t_1 = \frac{{n_2}^2}{{n_1}^2} * t_1 = \frac{25600^2}{12800^2} * 983{,}80 => \boldsymbol{t_2 = 3935{,}2\ s}$$

Como el tiempo resultante es aproximado al que se ha obtenido en la gráfica queda demostrado que la complejidad del algoritmo es $O(n^2)$.