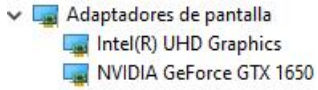
PREGUNTAS SECCIÓN 5

Sesión práctica 8

Jonathan Arias Busto
UO283586
71780982-Y
Escuela de Ingeniería Informática - EII

INFORMACIÓN SOBRE OREDENADOR USADO PARA MEDIR TIEMPOS

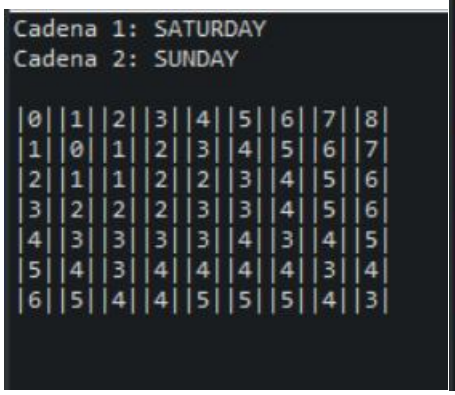


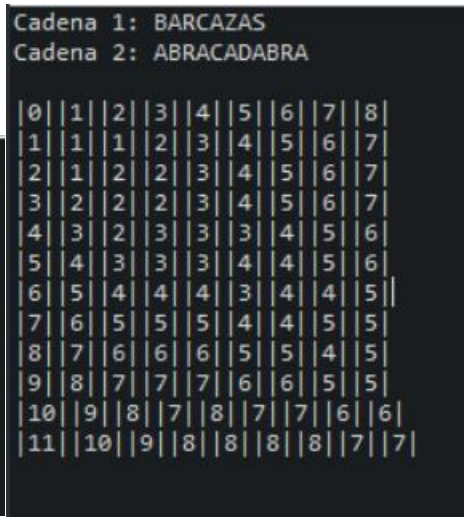
Se trata de un portátil con el modo de energía en “High performance”.

RESOLUCIÓN DEL PROBLEMA

Primero de todo vamos a comprobar la veracidad de la implementación que resuelve el problema de la *Distancia de Levenshtein.*

Tras completar dicha implementación cromprobamos que funcione para varios ejemplos. Por ejemplo:





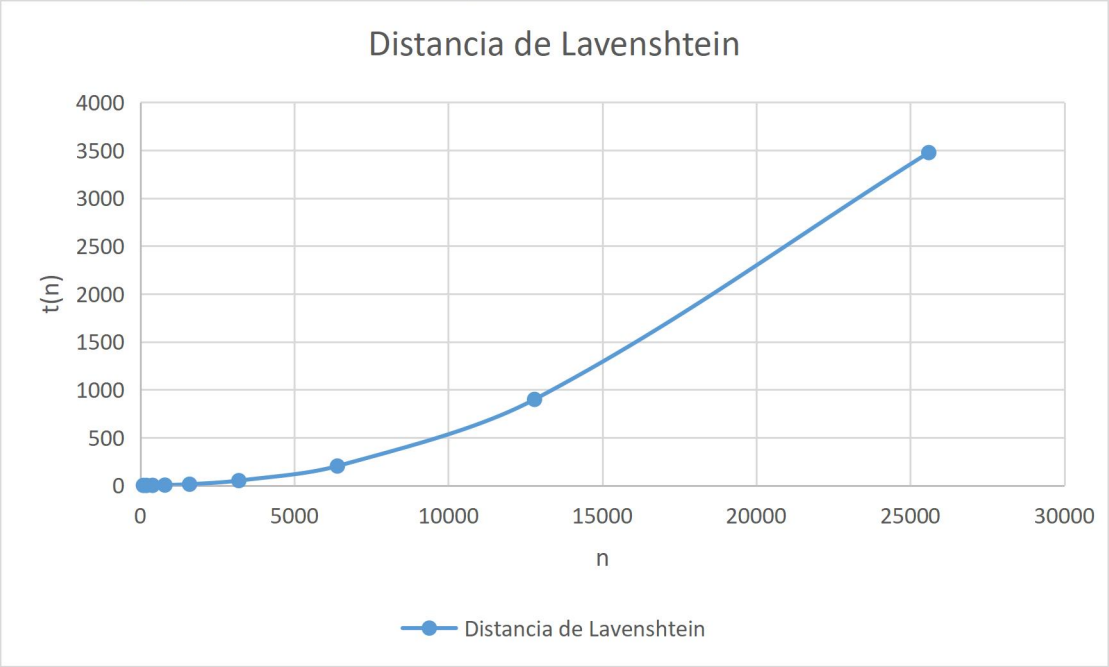
Como podemos ver funciona para ambos casos que son los ejemplos que se dieron en clase.

TOMA DE TIEMPOS

Ahora medimos tiempos para poder estudiar la complejidad temporal del algoritmo. Dichos tiempos se toman con cadenas aleatorias de tamaño (100, 200, 400, 800, 1600, ...). Los tiempos son los siguientes:

n	t	t (s)
100	35	0,35
200	29	0,29
400	95	0,95
800	350	3,5
1600	1263	12,63
3200	5035	50,35
6400	20245	202,45
12800	89837	898,37
25600	347607	3476,07
ERROR MEMORIA	n. veces = 100	

Con estos datos construimos un gráfico de dispersión para hacernos a la idea de que tendencia, en cuanto a complejidad se refiere, tiene el algoritmo implementado.



Como se puede apreciar el gráfico tiene una tendencia cuadrática.

Ahora con los tiempos calculados podemos comprobar teóricamente que complejidad tiene dicho algoritmo.

Tenemos:

--> n1 = 400 | t1 = 95
--> n2 = 800 | t2 = 350

Ahora podemos utilizar la fórmula de que usa K para comprobar si el tiempo es más o menos el mismo.

--> K = n1/n2 = 2
--> t2 (aprox.) = K^2 * t1 = 4 * 95 = 380

Como se puede ver el algoritmo tiene una complejidad temporal cuadrática para esta implementación. Esto tiene sentido debido a que el algoritmo tiene dos bucles for anidados, por lo que O(n) * O(n) = O(n^2).

```
public int distancia() {
    int[][] array = initialize();
    for (int i=1;i<convertir.length()+1;i++) {
        for (int j=1; j<cadena1.length()+1;j++) {
            if (convertir.charAt(i-1) == cadena1.charAt(j-1)) {
                array[i][j] = array[i-1][j-1];
            } else {
                array[i][j] = 1 + Math.min(Math.min(array[i-1][j-1], array[i-1][j]), array[i][j-1]);
            }
        }
    }
    //print(array);
    return array[convertir.length()][cadena1.length()];
}
```

En la imagen se puede apreciar dichos bucles for.
En la práctica la complejidad es un poco distinta a O(n^2) puesto que hay un if/else que afecta a dicha complejidad, pero de una forma aproximada podemos determinar que la complejidad será cuadrática.