Procesador:	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.8GHz
Memoria:	16 GB

Cuestiones:
Complejidad del algoritmo usando programación dinámica:

- La función rellenaTabla posee una complejidad cuadrática, O(n^2).
- Si la condición es verdadera, la función encontrarMSC posee complejidad O(n).
- La función máximo posee complejidad constante.

Complejidad del algoritmo usando recursividad:

- La función encontrarMSC posee complejidad exponencial, O(3^n), pues {a=3, b=1, k=0}, por substracción.
- La función máxima posee complejidad constante.

La complejidad del algoritmo con implementación recursiva es exponencial, cuanto más largas sean las cadenas más repeticiones innecesarias se realizarán en el algoritmo, y como consecuencia, más tiempo tardará en hallar la solución al problema.
Con programación dinámica se acelera este proceso, mejora la eficiencia del algoritmo. Evita calcular dos veces (o más) lo mismo.

Mi algoritmo recursivo no está completado.

Tablas de tiempos:

nTimes = 1000000000

n	progdin
	milisegundos
100	4.0E-9
200	2.71E-7
400	2.31E-7
800	2.33E-7
1600	2.66E-7
3200	2.36E-7
6400	2.47E-7
Java heap space	

Puede haber más de una MSC, p. e. GCCCTAGCG y GCGCAATG tiene dos GCGCG y GCCAG. La sección de código que determina qué subsecuencia es elegida se encuentra en la función encontrarMSC. Muestro a continuación la salida por consola y las implementaciones de la función para cada caso.

```
Problems  Javadoc  Declaration  Console  Git Staging  Coverage
<terminated> MscPrueba [Java Application] C:\Users\mayte\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_15.0.1.v202(
PROGRAMACIÓN DINÁMICA:
String1: *GCCCTAGCG
String2: *GCGCAATG
Inicializando la tabla...
Rellenando la tabla...
Imprimiendo la tabla...
          *          *          G          C          C          C          T          A          G          C          G
          *   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)   0( 0, 0)
          G   0( 0, 0)   1( 0, 0)   1( 1, 1)   1( 2, 1)   1( 3, 1)   1( 4, 1)   1( 5, 1)   1( 6, 0)   1( 7, 1)   1( 8, 0)
          C   0( 0, 0)   1( 1, 1)   2( 1, 1)   2( 2, 1)   2( 3, 1)   2( 4, 2)   2( 5, 2)   2( 6, 2)   2( 7, 1)   2( 8, 2)
          G   0( 0, 0)   1( 1, 2)   2( 2, 2)   2( 3, 2)   2( 4, 2)   2( 5, 2)   2( 6, 2)   3( 6, 2)   3( 7, 3)   3( 8, 2)
          C   0( 0, 0)   1( 1, 3)   2( 2, 3)   3( 2, 3)   3( 3, 3)   3( 4, 4)   3( 5, 4)   3( 7, 3)   4( 7, 3)   4( 8, 4)
          A   0( 0, 0)   1( 1, 4)   2( 2, 4)   3( 3, 4)   3( 4, 4)   3( 5, 4)   4( 5, 4)   4( 6, 5)   4( 8, 4)   4( 9, 4)
          A   0( 0, 0)   1( 1, 5)   2( 2, 5)   3( 3, 5)   3( 4, 5)   3( 5, 5)   4( 6, 5)   4( 7, 5)   4( 8, 5)   4( 9, 5)
          T   0( 0, 0)   1( 1, 6)   2( 2, 6)   3( 3, 6)   3( 4, 6)   4( 4, 6)   4( 6, 6)   4( 7, 6)   4( 8, 6)   4( 9, 6)
          G   0( 0, 0)   1( 1, 7)   2( 2, 7)   3( 3, 7)   3( 4, 7)   4( 5, 7)   4( 6, 7)   5( 6, 7)   5( 7, 8)   5( 8, 7)
Buscando la MSC...
           5( 8, 7)->G
           4( 6, 5)->A
           3( 3, 3)->C
           2( 1, 1)->C
           1( 0, 0)->G
Imprimiendo la MSC...
GCCAG
           1( 0, 0)->G
           2( 1, 1)->C
           3( 6, 2)->G
           4( 7, 3)->C
           5( 6, 7)->G
GCGCG
```

```java
/**
 * Encuentra la MSC a partir de los valores de la tabla
 * @param v si verdadero se propocionan mensajes que muestran los pasos seguidos
 */
public String encontarMSC(boolean v) {
    // TODO: después de rellenas la tabla, reconstruye la MSC empezando por el
    // último elemento
    if (v) {
        int i = size1-1, j = size2-1;
        while (i > j) {
            if (str1.charAt(i) == str2.charAt(j)) {
                result = str1.charAt(i) + result;
                System.out.printf("\t%4d(%2d,%2d)->%c\n", table[i][j].value, table[i][j].iPrev, table[i][j].jPrev, str1.cha
                i--;
            }
        }
        for (int k = size2-1; k > 0; k--) {
            if (str1.charAt(k) == str2.charAt(k)) {
                result = str1.charAt(k) + result;
                System.out.printf("\t%4d(%2d,%2d)->%c\n", table[k][k].value, table[k][k].iPrev, table[k][k].jPrev, str1.cha
            }
        }
        return result;
    }
    return null;
}
```

```java
public String encontar() {
    String r = "";
    int pivote = 0;
    for (int j = 1; j < size2; j++) {
        for (int i = 1; i < size1; i++) {
            if (str1.charAt(i) == str2.charAt(j) && pivote < table[i][j].value) {
                pivote = table[i][j].value;
                r = str1.charAt(i) + r;
                System.out.printf("\t%4d(%2d,%2d)->%c\n", table[i][j].value, table[i][j].iPrev, table[i][j].jPrev, str1.cha
                break;
            }
        }
    }
    return r;
}
```