Examen final – 22 de junio de 2017

Apellidos, nombre _____ NIF: _____

Pregunta 1 (1,5 p.)

Responde a las siguientes preguntas (Nota: Resuelve las operaciones intermedias):

a) (1 punto) Se ha medido tiempos empíricamente para un algoritmo y el resultado ha sido la siguiente tabla:

n	Tiempo (msegundos)
1024	1326
2048	5274
4096	21077

No disponemos del código fuente del algoritmo, pero sabemos la complejidad analítica es o bien $O(n^2)$ o bien O(nlogn). ¿A cuál de estas complejidades se ajustan mejor los tiempos medidos? Proporcione una estimación para n=2048 y para n= 4096. Explique el criterio de elección.

b) (0,5 puntos) Considere un algoritmo de complejidad O(logn) Si para t=4 segundos puede resolver un problema de tamaño n=32 ¿cuál sería el tamaño del problema resuelto si dispusiéramos de un tiempo de 12 segundos?

Pregunta 2 (1 p.)

Dados los siguientes métodos, indica su complejidad y explica cómo lo has calculado claramente (asumimos que las instrucciones básicas en Java tienen una complejidad O(1)) (contesta directamente en esta hoja junto a cada apartado):

a) (0.5 p.)

```
public void complexity1(int n) {
    int x = n;
    while (x > 0) {
        int y = x;
        while (y > 0) {
            y = y / 2;
        }
        x = x - 1;
    }
}
```

b) (0.5 p.)

```
public void complexity2(int n) {
    int x = n;
    while (x < Integer.MAX_VALUE) {
        int y = Integer.MAX_VALUE;
        while (y > 0) {
            y = y - 2;
        }
        x = x + 2;
    }
    complexity2(n-2);
    complexity2(n-2);
    complexity2(n-2);
}
```

Pregunta 3 (1,5 p.)

Convertir la implementación del algoritmo ordenación por mezcla o mergesort DV recursiva a una implementación que utilice hilos para ejecutar cada una de las llamadas de forma paralela.

```
public class MezclaParalelo
{
	static int []v;

	public static void mezcla(int[]v)
	{
		mezclarec (v,0,v.length-1);
	}

	private static void mezclarec(int[] v, int iz,int de)
	{
		if (de>iz)
		{
			int m=(iz+de)/2;
			mezclarec(v,iz,m);
			mezclarec(v,m+1,de);
			combina(v,iz,m,m+1,de);
		}
	}

	public static void main (String arg [] )
	{
		int n=10;
		v = generarVector(n);
		mezcla(v);
		imprimirVector(v);
	}
...
}
```

Pregunta 4 (2 p.)

Dadas dos cadenas de texto **str1** y **str2** y las operaciones (indicadas debajo), que pueden ser realizadas sobre **str1**, encuentra el mínimo número de ediciones (operaciones) necesarias para convertir **str1** en **str2**.

1. Insertar
2. Eliminar
3. Reemplazar

Todas las operaciones tienen el mismo coste. Algunos ejemplos:

```
Entrada:   str1 = "geek", str2 = "gesek"
Salida:  1
Podemos convertir str1 en str2 insertando una 's'.

Entrada:   str1 = "cat", str2 = "cut"
Salida:  1
Podemos convertir str1 en str2 re'a' with 'u'.
```

Apellidos, nombre _____ NIF: _____

```
Entrada:   str1 = "sunday", str2 = "saturday"
Salida:  3
Los últimos tres caracteres y el primero son los mismos. Básicamente, necesitamos
covertir "un" a "atur".  Podemos remplazar 'n' por 'r', insertar 't', insertar 'a'
```

a) (1 p.) Crea un método llamado **EditDistance** (utilizando pseudocódigo) para indicar cómo implementarías una solución a este problema utilizando programación dinámica. El método recibirá como parámetros las dos cadenas de texto (una por cada palabra). Ten en cuenta la tabla proporcionada en el siguiente apartado.

b) (1 p.). Completa la tabla correspondiente, teniendo en cuenta los valores proporcionados, que resuelve el problema para las palabras (**Sunday** y **Saturday**), indicando claramente el resultado final.

		S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
	0	1	2	3	4	5	6	7	8
S	1								
u	2								
n	3								
d	4								
a	5								
y	6								

Pregunta 5 (2 p.)

Necesitamos que un autobús haga un recorrido desde la estación de autobuses (a través de **n** ciudades diferentes) minimizando la cantidad de kilómetros que realiza hasta que vuelva a la estación de autobuses. Asumiendo que hay una carretera entre cada par de ciudades y que, como somos expertos en Algoritmia, sabemos que la solución tiene que ser un ciclo simple, completa el siguiente código para encontrar todas las posibles soluciones al problema:

```
import java.util.Random;
public class Problem {
	static int n; //número de ciudades
	static String []nodes; //ciudades
	static int[][]weights; //distancias entre ciudades (-1 no hay camino)
	static int source; //ciudad inicial
	static boolean []mark; //para marcar ciudades ya visitadas
	static int[] path;  //almacena un camino posible
	static int cost; //coste de un camino possible
	static int length; //tamaño (nivel o longitude) de un camino
```

```java
	static int nsol; //número total de soluciones

	public static void main (String arg[]) {
		//Consideramos las variable correctamente inicializadas
		backtracking(               );
		if (nsol==0)
			System.out.println("THERE ARE NO CYCLES");
		else System.out.println("NUMBER OF CYCLES = " + nsol);
	}

	static void backtracking(                ) {
		if (current==source && length>0) {  //estado solución
			if (               ) {

				for (                       )
					System.out.print(                    ); /*
nombre de las ciudades en orden */
				System.out.println("ITS COST IS = "+cost);
			}
		}
		else
			for (                       )
				if(                                     ) {



					backtracking(                    );



				}
	} //backtracking
}
```

Pregunta 6 (2 p.)

Consideremos la técnica de ramificación y poda. Responder de forma breve pero razonada las siguientes cuestiones:

a) Escribir el código para el método principal de la técnica:

```java
public void ramificaYPoda(Nodo nodoRaiz)
```

b) Una de las condiciones que manejamos en el esquema es la cota de poda. Qué efecto tendría si al encontrar una solución cambiásemos esta cota de poda a 0.

c) Si la clase cola guardase los elementos en una pila LIFO en vez de una cola de prioridad.

d) Si el método `getHeuristico()` devolviera un valor constante para todos los estados del problema.