Convocatoria ordinaria – 18 de mayo de 2023 (Parte 1)

Apellidos, nombre _____ NIF: _

Pregunta 1 (2 p.)

Responde a las siguientes preguntas (Nota: no utilices calculadora y responde con un valor numérico):

a) (1 p.) Tenemos un método con una complejidad $O(4^n)$. Si para n=5 tarda 2 segundos, ¿cuánto tarda para n=9?

b) (1 p.) Tenemos un método con una complejidad $O(4^n)$. Si para n=5 tarda 2 segundos, que tamaño del problema podemos resolver si tenemos 128 segundos en el mismo ordenador.

Pregunta 2 (2 p.)

Teniendo en cuenta los siguientes métodos, indica sus complejidades y explica cómo las has calculado (aprovecha el espacio para contestarlo en esta misma hoja):

a) (0,5 p.)

```
public static void method2(int n) {
    int p = 0;
    for (int i = 2*n; i >= 0; i -= 3)
        for(int j = 1; j <= n*n*n*n; j *= 5) {
            System.out.println("Hello");
            p++;
            for (int k = 0; k < i; k++) {
                System.out.println(p);
            }
        }
}
```

b) (0,5 p.)

```java
public static void method1(int n) {
    if (n <= 0) System.out.println("hello");
    else {
        int p = 10;
        method1(n-2);
        method1(n-2);
        method1(n-2);
        for (int i = 0; i<=n-1; i++)
            for (int j=4; j<=n; j++) {
                p++;
                System.out.println(p);
            }
    }
}
```

c) (0,5 p.)

```java
public static void method1(int n){
    int c = 0;
    for (int i=2; i<=n/4; i++)
        for (int j=10; j<=n*n*n; j*=2)
            for (int p=4*n; p>=n/3; p-=1)
                c++;
}
```

d) (0,5 p.)

```java
public static void method2(int n){
    if (n > 5) {
        for (int i=n; i>0; i--)
            System.out.println("hello");
        method2(n/3);
        method2(n/3);
        method2(n/3);
        method2(n/3);
        method2(n/3);
        method2(n/3);
    }
}
```

Pregunta 3 (3 p.)

a) (1 p.) Realiza la traza del algoritmo de ordenación Quicksort (usando la mediana de 3 para seleccionar el pivote) para ordenar los siguientes números en orden ascendente 6, 3, 4, 7, 2, 1, 5. Los pasos seguidos deben verse claramente para que el ejercicio sea considerado válido: rodea con un círculo el pivote elegido en cada caso y subraya cada partición obtenida.

b) (1 p.) Completa la traza del algoritmo de ordenación Mergesort para ordenar los siguientes números en orden ascendente 6, 3, 4, 7, 2, 1, 5. Los pasos seguidos deben verse claramente para que el ejercicio se considere válido.

c) (1 p.) Rellena la siguiente tabla con las complejidades de los diferentes algoritmos de ordenación:

	Caso peor	**Caso medio**	**Caso mejor**
Burbuja			
Inserción			
Selección			

Quicksort (con un buen pivote)			
Mergesort (Mezcla)			

Pregunta 4 (3 p.)

Partiendo del algoritmo de ordenación de Mezcla (Mergesort), resuelve las siguientes preguntas:

a) (0,5 p.) Cuál es la altura del árbol de llamadas si tenemos n elementos a ordenar.

b) (1,5 p.) Considerando que tenemos una clase en Java denominada Mezcla con la siguiente forma:

```
public class Mezcla {
    private int []elementos;

    public Mezcla(int[]elementos) {
            this.elementos = elementos;
            mezcla(0, elementos.length-1);
    }

    private void mezcla(int izq, int der) {
            //TODO
    }

    /* Proceso que combina dos secuencias ordenadas
     * Usa dos vectores auxiliaries de entreda y lo deja en array elementos[] */
    private void combinar(int x1, int x2, int y1, int y2) {
            //…
    }

}
```

Sabiendo que tenemos el método **combinar()** implementado, escribe el código del método recursivo **mezcla()**, necesario para realizar la mezcla.

c) (1 p.) Justifica por qué es posible paralelizar este algoritmo y si supondrá ganancia de tiempo si tenemos un procesador de varios núcleos.