Escuela de Ingeniería Informática
Universidad de Oviedo

Guion de Prácticas

Estructuras de Datos [ED]

Curso 2025 – 2026

Proyecto 1: Recursividad, algorítmica y medición de tiempos

Durante la unidad, el alumno creará un proyecto llamado ***Algoritmia*** compuesto por las clases que se indican a continuación.

Clase Algorithms

Esta clase contendrá los siguientes algoritmos:

- ***boolean esPar(int n***): Método recursivo que retorna si un nº es par o no.
- ***Int factorialIter(int n)***: Retorna el factorial de un número n (v. iterativa).
- ***Int factorialRec(int n)***: Retorna el factorial de un número n (v. recursiva).
- ***Int fibonacciIter(int n)***: Versión iterativa que devuelve el valor del termino d la serie de Fibonacci.
- ***Int fibonacciRec(int n)***: Versión recursiva que devuelve el valor del termino d la serie de Fibonacci.
- ***Boolean esSimétrico(int[][] matrix)***: Método recursivo que recibe una matriz y retorna si es simétrica o no.
- ***int resto(int dividendo, int divisor)***: Método recursivo que recibe dividendo y divisor y retorna el resto de la división.
- ***int division(int dividendo, int divisor)***: Método recursivo que recibe dividendo y divisor y retorna el resultado de la división.
- ***powIter(int n)***: Método iterativo que calcula la potencia de 2 para un exponente dado.
- ***powRec(int n)***: Método recursivo que calcula la potencia de 2 para un exponente dado, con las siguientes versiones:
  - *1 llamada que decrece el exponente linealmente (n--).*
  - *1 llamada que decrece el exponente más rápidamente.*
  - *2 llamadas que decrecen el exponente linealmente (n--).*
  - *2 llamadas que decrecen el exponente más rápidamente.*
- Métodos de prueba para la medición de complejidades temporales. Hacen uso del método **doNothing.**
  - ***void lineal (long n)***
  - ***void cuadratica (long n)***

- ***void cubica (long n)***
- ***void logaritmica (long n)***

Todos estos métodos deben incluir comentarios que indiquen y justifiquen brevemente la **complejidad** de la implementación realizada.

Clase AlgorithmsBenchmark

Esta clase contendrá los métodos capaces de evaluar el rendimiento de cualquier método que utilice un único parámetro entero (int) en cualquier clase Java. Utilizará reflectividad computacional para invocar el método y medir su tiempo de ejecución para una carga de trabajo entera **n** pasada como parámetro. Los resultados se almacenarán en un fichero CSV.

Se explicará un método llamado ***test*** cuyos parámetros irá indicando el profesor durante la clase. La última versión del dicho método recibirá los siguientes parámetros:

- ***output***. Nombre del fichero CSV de salida.
- ***startN***: Carga de trabajo mínima n sobre la que realizará la medición de tiempos.
- ***endN***: Carga de trabajo máxima n sobre la que realizará la medición de tiempos. Se ejecutará el algoritmo desde una n igual a startN hasta una n igual a endN.
- ***times***. Número de veces que se ejecutará el algoritmo bajo prueba. Se medirá el tiempo de ejecución de cada una, almacenando el promedio de tiempos de todas ellas.
- ***nombreClase***: nombre de la clase en la que se encuentra el método a evaluar.
- ***nombreMetodo***: nombre del método a evaluar.

Ejemplo de uso del método **testFinal**:

testFinal*("lineal.csv", 0, 1000, 5, "Algorithms", "lineal");*

Gráficos de Análisis Empírico de Eficiencia

Utilizando la clase **AlgorithmsBenchmark**, se deben medir los tiempos y generar los siguientes gráficos de rendimiento en Excel:

- **Hoja A**: Incluirá la representación gráfica de las complejidades temporales obtenidas tras la ejecución de los algoritmos lineales, cuadráticos, cúbicos y logarítmicos.
- **Hoja B:** Incluirá la representación gráfica de las complejidades temporales obtenidas tras la ejecución de las 5 versiones diferentes de la función *pow*.

Se deben incluir pruebas unitarias (utilizando **JUnit**) para verificar la correcta implementación de cada una de las funciones de la primera sección. Se deben lanzar las excepciones adecuadas para evitar que esas funciones se ejecuten con argumentos incorrectos.

Consideraciones sobre las mediciones

1. Las mediciones deben realizarse en la misma máquina para que los tiempos obtenidos para cada implementación sean coherentes.
2. Para evitar mediciones imprecisas que siempre devuelven 0 o números cercanos a 0, debe asegurarse de que se llama a ***doNothing*** en la implementación de la función, como se ve en la clase.
3. Debe asegurarse de que el argumento ***endN*** (del método *TestBench.test*) sea lo suficientemente grande como para que las ejecuciones consuman un tiempo que pueda medirse con precisión.
4. El argumento ***endN*** debe reducirse en lo posible, pero sin llegar al caso del punto anterior, para evitar ejecuciones que no terminan o emplean demasiado tiempo.
5. Los métodos que se miden con *TestBench.test* deben tener un único parámetro y este debe ser de tipo *long*.
6. Los gráficos resultantes deben ser coherentes con las complejidades calculadas. Es decir, hay que apreciar la diferencia entre los gráficos de implementaciones con diferente complejidad. Si dos implementaciones que teóricamente deberían tener complejidades diferentes tienen gráficos similares, es muy posible que:
    a. Se deba a ejecuciones demasiado rápidas de ambas implementaciones, que no permiten medir los tiempos con precisión. En estos casos, se recomienda aumentar el parámetro *endN* hasta que se aprecien las diferencias.
    b. Una implementación se haya estado ejecutando al mismo tiempo que otros procesos en la misma máquina. Por ejemplo, si ves transmisiones en directo o juegas a videojuegos mientras se realizan las mediciones, esto puede hacer que algunas mediciones tarden mucho más tiempo, lo que estropearía el resultado final.

Consideraciones sobre la entrega

Una vez completado el proyecto, deberá cargarse en el Campus Virtual antes de la siguiente sesión, en la tarea habilitada para tal fin. Transcurrido este plazo, no se aceptará la entrega del proyecto bajo ningún concepto.