LEA DETENIDAMENTE LAS NORMAS DE LA PRUEBA

Cualquier incumplimiento de las normas significará una respuesta nula al examen y por tanto una valoración de 0 puntos.

1. **La resolución del ejercicio propuesto se entregará en un único archivo formato ZIP que contendrá EXCLUSIVAMENTE:**
    1. **El proyecto que se proporciona en la actividad con la codificación de la solución planteada por el alumno en el fichero "Solucion.java". Cada línea de código debe ser comentada indicando la finalidad de la misma (use comentarios en línea, para ello antes de cada comentario escriba //).**
    2. **Un documento en formato PDF donde se indicarán las respuestas a cada una de las cinco cuestiones planteadas en el presente documento.**
2. **El nombre del fichero ZIP tendrá un formato especifico dictado por el nombre de cada alumno. Por ejemplo, para un alumno llamado "José María Núñez Pérez" el fichero se nombrará como NunyezPerezJM.zip. Obsérvese que las tildes son ignoradas y las eñes sustituidas.**
3. **En caso de detectarse soluciones copiadas, la nota de dichas soluciones 0 y se requerirán explicaciones ante los profesores de la asignatura para evaluar si se permite poder continuar con la evaluación continua.**
4. **El fichero ZIP se subirá utilizando la correspondiente tarea en el Aula Virtual.**

IMPORTANTE: Cualquier envío que no respete el formato de compresión o el nombre adecuado será ignorado y, por tanto, valorado con cero puntos.

Realice la implementación recursiva del siguiente problema planteado. Debe cumplir con la fórmula que indique en el apartado 2.a

Problema planteado:

Dadas dos matrices de enteros, retorne un booleano indicando si son iguales, es decir, si tienen los mismos valores en las mismas posiciones.

Para obtener su solución, debe deben implementar los siguientes métodos (usando recursividad en ambos):

Método sonMatricesIguales: Recibe dos matrices y la fila por la que comenzar a comparar (por defecto se recibirá siempre la última fila, de forma que se debe ir decrementando hasta llegar al caso base). Por cada fila a comparar se usará el siguiente método sonVectoresIguales (definido a continuación) para comparar vector a vector (es decir, fila a fila). Finalmente devolverá un valor booleano indicando si las matrices son iguales (true) o no (false). La declaración de este método será:

```
public boolean sonMatricesIguales(int [][] matriz1, int [][] matriz2, int fila)
```

Método sonVectoresIguales: Recibe dos vectores y devuelve un valor booleano indicando si ambos vectores son iguales (true) o no (false). La declaración de este método será:

```
public boolean sonIguales(int [] vector 1, int [] vector2)
```

Solo puede usar los parámetros que se indican en los métodos descritos. Puede crear todos los métodos auxiliares que estime oportuno.

Consejos:

Las matrices deben tener el mismo número de elementos para ser iguales, este debe ser uno de los casos base.

Use el argumento fila para detectar en qué fila de la matriz se encuentra, y vaya decrementándola hasta llegar a su caso base.

Para recorrer los vectores, debe dividirlos en trozos (de este modo no necesitará un índice para la columna a tratar). Para trocear un vector use el siguiente método:

```
Arrays.copyOfRange(<VECTOR>, <INICIO>, <FIN>)
```

Donde:

- `<VECTOR>` es el vector del que queremos extraer un subvector.
- `<INICIO>` inicio del rango de los valores a obtener en el nuevo vector.
- `<FIN>` fin del rango de los valores a obtener en el nuevo vector.

A la hora de calcular los tiempos de ejecución de su código, recuerde usar el método visto en clase para la obtención de los nanosegundos: System.nanoTime();

PROYECTO REALIZADO POR:

- JESUS CIVICO LOBATO: 49137352Z
- IGNACIO PUIG MARGALEF: 49133074Z

Por favor conteste a las cuestiones planteadas con sus palabras, haga uso de ejemplos para completar sus explicaciones:

a. **Indique a continuación la fórmula que usará para la implementación de su solución recursiva. Indique su caso base y su caso general. Explique por qué considera que su fórmula es la adecuada para el problema planteado:**

METODO sonMatricesIguales

```
public static boolean sonMatricesIguales(int[][] matriz1, int[][] matriz2, int fila) {
    if (matriz1.length == matriz2.length && matriz1[0].length == matriz2[0].length) {
        boolean iguales = true;
        if (fila > -1) {


            iguales = sonMatricesIguales(matriz1, matriz2, fila - 1);
            if (iguales) {
                iguales = sonIguales(matriz1[fila], matriz2[fila]);
            }
        }
        return iguales;
    } else {
        return false;
    }

}
```

- Esta la solución recursiva del método sonMatricesIguales que hemos implementado.
- Nuestro caso base se dará cuando el valor de fila llegue a -1. Esto es debido a que, cuando estemos en fila = 0 (no sería este nuestro caso base, ya que queremos quedarnos este valor en la pila para pasárselo también a la función sonIguales y que nos compruebe también la fila 0), llamaremos a la función, pero esta vez con el valor -1 para fila. En este punto ya habremos hecho las llamadas a todas las filas y, además, fila = -1 no existe (nos lanzaría una excepción del tipo arrayIndexOutOfBounds), por lo tanto saldríamos del if (salir del if por primera vez se considera nuestro caso base) y haríamos el primer return (inicializado en true para que pueda hacer la primera llamada a sonIguales, ya que en caso de ser false no lo haría).
- Nuestro caso general se dará mientras no lleguemos al caso base.
- Consideramos que nuestra formula del método sonMatricesIguales es adecuada porque:

En primer lugar, cumple el objetivo primero: recorre todas las filas de las matrices de manera recursiva. Este recorrido lo hará de manera descendente (como se indicaba en el enunciado). Es por ello que, en cada llamada a la función sonMatricesIguales, a parte de pasar las matrices pasamos fila – 1.

Además, por cada llamada recursiva de nuestra función, estamos haciendo la llamada respectiva al método sonIguales, al cual le pasamos los vectores correspondientes a las filas de las matrices. Esto nos comprobara la igualdad entre filas y, en caso de ser false la comparación, dejara de entrar en el if (que tiene como condición iguales == true), y devolverá false hasta llegar al main.
En último lugar, destacar que antes de nada hacemos la comprobación entre el tamaño tanto de filas como de columnas entre ambas matrices ya que es la condición número 1 de comparación.

METODO sonMatricesIguales

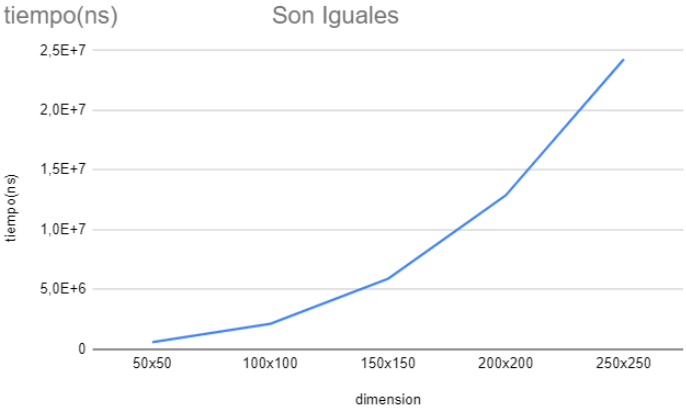
```java
public static boolean sonIguales(int[] vector1, int[] vector2) {
    boolean iguales = true;
    if (vector1.length > 0){
        iguales = sonIguales(Arrays.copyOfRange(vector1, 0, vector1.length - 1), Arrays.copyOfRange(vector2, 0, vector2.length - 1));
        if (vector1[vector1.length - 1] != vector2[vector2.length - 1] && iguales) {
            iguales = false;
        }
        return iguales;
    }
    return iguales;

}
```

-Esta es la solución del método sonIguales que hemos implementado
-Nuestro caso base se dará siempre que la longitud de los vectores sea mayor que 0 ya que un vector de longitud cero no existe, y se irán realizando las llamadas recursivas siempre que se cumpla esta condición, obtendremos cada vez vectores más pequeños que iremos comparado su última posición
-Nuestro caso general nos ira devolviendo la variable igual así si en algún momento si no lo son nos ahorramos realizar pasos extras
Esta forma de realizar el método e forma recursiva pensamos que es la mejor porque es lo más corto que hemos podido obtener y además en el caso en que alguna posición no sean iguales te evitas realizar pasos de mas

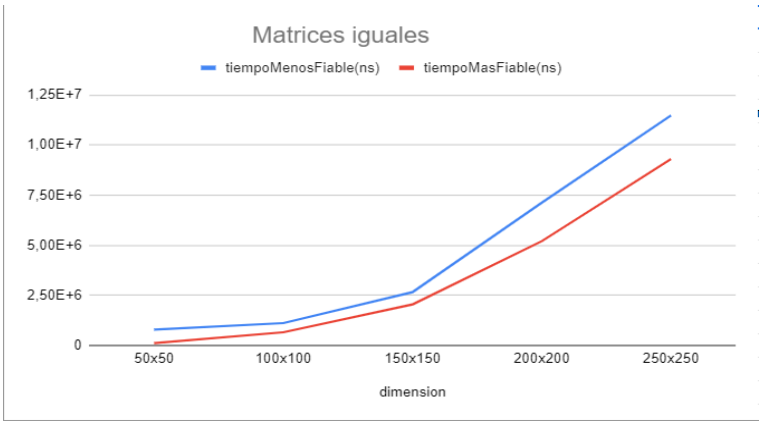
b. **Realice un análisis del algoritmo desarrollado.**

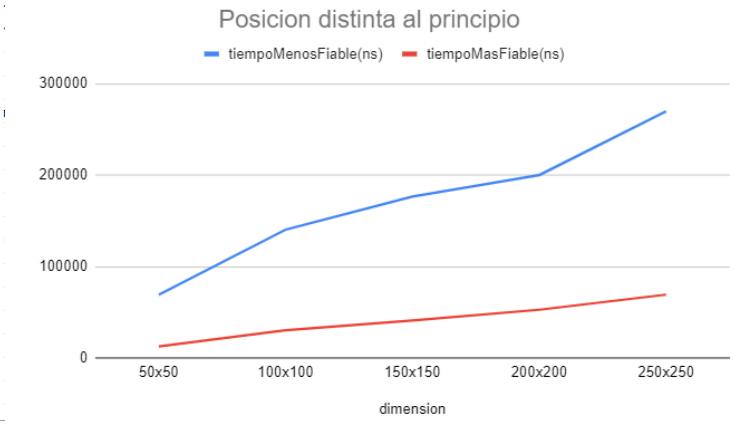
i. **Calcule el tiempo de ejecución de su código, tal y como se ha hecho en clase. Muestre los tiempos de cada ejecución por consola, cree una gráfica con los tiempos obtenidos y muéstrela a continuación:**

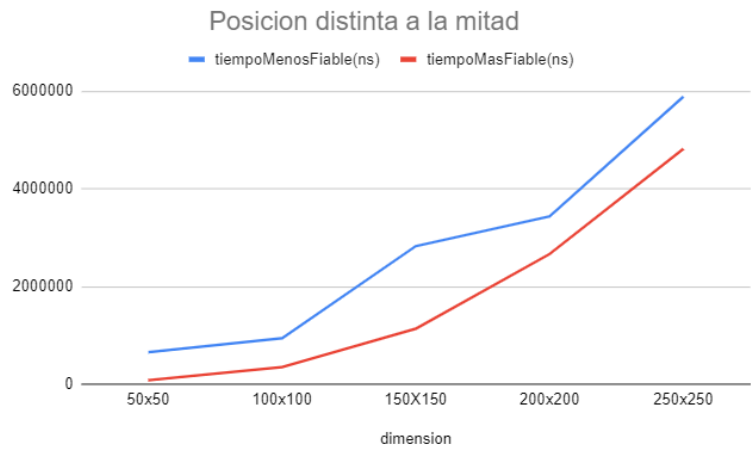


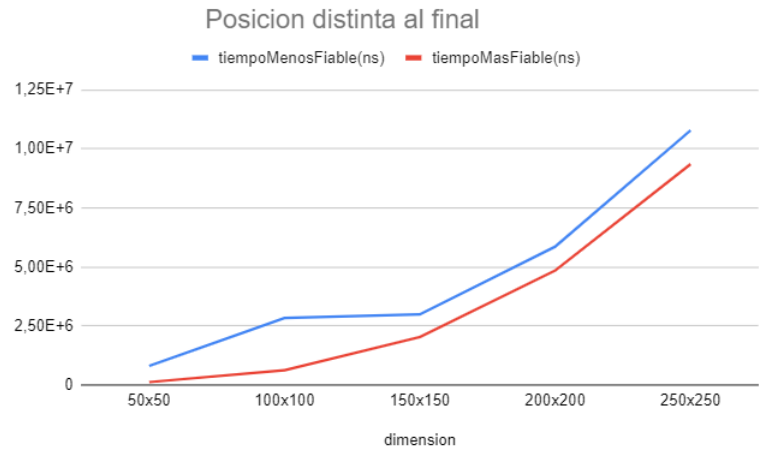
ii. **¿Qué mecanismo debe usar para obtener mayor fiabilidad de los tiempos obtenidos? Coméntelo a continuación (y aplíquelo en su código)**

Para obtener mejor fiabilidad de los tiempos obtenidos, hemos empleado en primera instancia el mecanismo de la media aritmética.
Esto es debido a que al calcular los tiempos y ejecutar varias veces en cualquiera de los ejemplos (posición distinta de las matrices al principio, a mitad, a final y matrices iguales) y con cualquier tamaño de matrices (50, 100, 200, 250…), nos damos cuenta de que los valores del tiempo son muy oscilantes. Es por ello que, considerando una media aritmética (en nuestro caso, entre 100 tiempos de ejecución), estimamos que la fiabilidad de los tiempos va a ser mayor que en el caso de hacerlo único.
Vamos a ver una comparación de los tiempos menos fiable con más fiable:



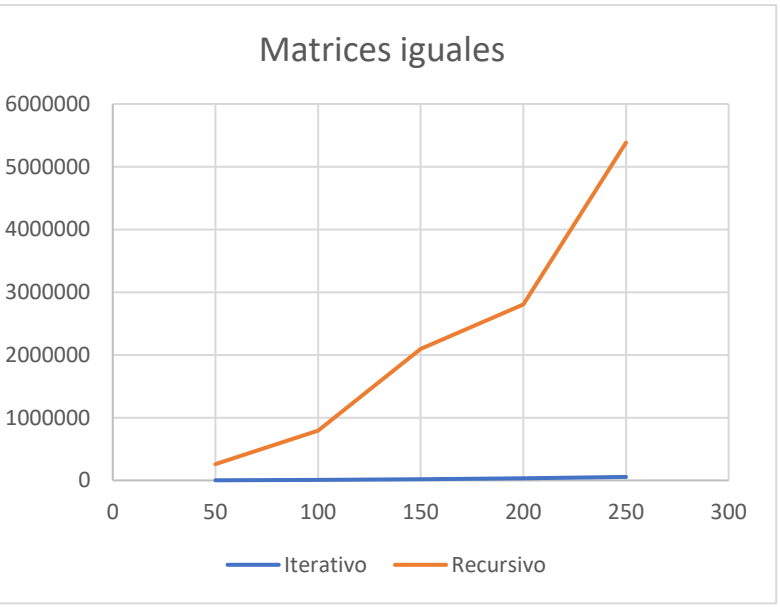


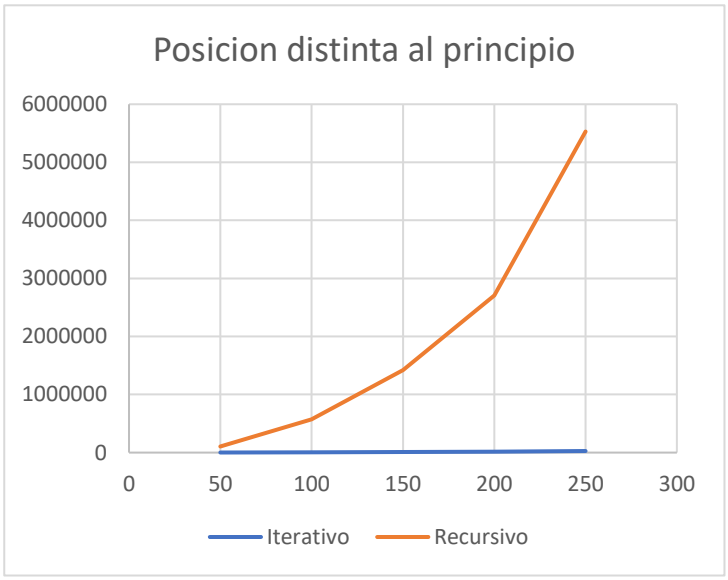




c. **¿Cree que una implementación iterativa sería más eficiente en tiempo? ¿Por qué?**

Sí, porque el algoritmo iterativo va a tener una variable de escape en caso de que el contenido de dos posiciones iguales de las matrices sea diferente. En cambio, el algoritmo recursivo precisa de hacer todas las llamadas hasta el caso base, no pudiendo salir antes de esto (no podemos usar un salto de pila). Vamos a compararlo gráficamente con los tiempos más fiables:





d. **Y en memoria consumida ¿sería más eficiente la implementación iterativa? ¿Por qué?**

Debido al funcionamiento de la recursividad, en cuanto a memoria, podemos afirmar que el algoritmo recursivo va a consumir más recursos que el iterativo. Esto será debido a que necesitamos apilar el contenido de las variables en cada llamada hasta llegar al caso base para poder recuperarlos a la vuelta. Esto nos va a generar un coste que no genera el iterativo, llegando incluso a producirse las famosas excepciones stackOverflowException, las cuales vienen dadas por "desbordamientos de pila", al exceder la cantidad de apilamientos que podemos llevar a cabo. Esto no nos ocurrirá con el algoritmo iterativo (a igual magnitud). Por lo tanto, podemos afirmar que la implementación iterativa es más eficiente que la recursiva.