PRÁCTICA 1 – Esquema Divide-and-Conquer

Torneo de Tenis

Objetivos

- Construir soluciones a un problema utilizando el **método algorítmico divide-and-conquer** (o divide y vencerás). Comparar los algoritmos implementados tanto cualitativa como cuantitativamente.
- Realizar el **análisis de la eficiencia** de las soluciones aportadas, y una comparativa tanto desde el punto de vista teórico como práctico.

Requerimientos para superar la práctica

- Conocer cómo implementar el **esquema Divide y Vencerás (DyV)** (divide-and-conquer) para responder a una necesidad concreta (en este caso, resolver un problema relativo a encontrar los emparejamientos de jugadores en un torneo de tenis de todos contra todos (liga)).
- **Evaluar los algoritmos implementados**.

Enunciado del problema

Como ya sabemos, **theBestSoft** es la mejor empresa de desarrollo de software de **EDAland** (un país en continuo crecimiento gracias al extraordinario uso que se hace de las estructuras de datos y algoritmos para resolver los problemas que allí se plantean). Esta empresa, debido la gran demanda de proyectos software que tiene, decidió crear un nuevo departamento (**EDASoft**) que se encarga de la resolución de problemas. Este departamento recibe peticiones por parte de otros departamentos de la empresa para que resuelva los problemas que se le plantean a la misma, siempre de la forma **más eficiente posible**.

En esta ocasión el exclusivo club de tenis de EDAland, **EDAland-Gran-Club-de-Tenis** ha encargado a *theBestSoft* un importante proyecto, pues debe organizar un torneo de tenis de Gran Slam (**EDAland-Garros**) cuyo principal objetivo es desarrollar el cuadro de cruces del torneo/liga (round-robin tournament) de todos contra todos, con *n* participantes en donde cada jugador ha de jugar exactamente una vez contra cada uno de sus posibles *n – 1* competidores, y además ha de jugar un partido cada día, teniendo a *lo sumo un día de descanso* en todo el torneo. Por ejemplo, las siguientes tablas muestran posibles cuadros de emparejamientos para torneos con $n = 5$ y $n = 6$ jugadores, respectivamente.

Jug/día	**d1**	**d2**	**d3**	**d4**	**d5**
1	2	3	4	5	--
2	1	5	3	--	4
3	--	1	2	4	5
4	5	--	1	3	2
5	4	2	--	1	3

Jug/día	**d1**	**d2**	**d3**	**d4**	**d5**
1	2	3	4	5	6
2	1	5	3	6	4
3	6	1	2	4	5
4	5	6	1	3	2
5	4	2	6	1	3
6	3	4	5	2	1

Teniendo en cuenta los ejemplos anteriores, para resolver este problema procederemos por partes, considerando los dos siguientes casos:

1. Si *n* es **potencia de 2** ($n = 2^k$), implementaremos un algoritmo para construir un cuadro de partidos del torneo que permita terminarlo en $n - 1$ días.
2. Dado cualquier $n > 1$, implementaremos un algoritmo para construir un cuadro de partidos del torneo que permita terminarlo en $n - 1$ días si *n* es **par**, o en *n* días si *n* es **impar**.

Caso 1: *n* es potencia de dos ($n = 2^k$)

El caso más simple se produce cuando sólo tenemos dos jugadores $(n = 2)$, cuya solución es fácil pues basta enfrentar uno contra el otro.

En el caso de que $n > 2$, aplicaremos la técnica de Divide y Vencerás para construir la tabla requerida suponiendo que tenemos calculada ya una solución para la mitad de los jugadores. Esto es, tenemos relleno el cuadrante superior izquierdo de la tabla. En este caso los otros tres cuadrantes no son difíciles de rellenar, como puede observarse en la siguiente figura, donde se han tenido en cuenta las siguientes restricciones:

1. El cuadrante inferior izquierdo debe enfrentar a los jugadores de número superior entre ellos, por lo que se obtiene sumando *n/2* a los valores del cuadrante superior izquierdo.
2. El cuadrante superior derecho enfrenta a los jugadores con menores y mayores números, y se puede obtener enfrentando a los jugadores numerados de *1 a n/2* contra los numerados de *(n/2)+1 a n* respectivamente en el día *n/2*, y después rotando los valores de *(n/2)+1 a n* cada día.
3. Análogamente, el cuadrante inferior derecho enfrenta a los jugadores de mayor número contra los de menor número, y se puede obtener enfrentando a los jugadores de *(n/2)+1 a n* contra los jugadores de *1 a n/2* respectivamente en el día *n/2*, y después rotando los valores de *1 a n* cada día, pero en sentido contrario a como lo hemos hecho para el cuadrante superior derecho.

	d1
j1	2
j2	1

	d1	**d2**	**d3**
j1	2	3	4
j2	1	4	3
j3	4	1	2
j4	3	2	1

	d1	**d2**	**d3**	**d4**	**d5**	**d6**	**d7**
j1	2	3	4	5	6	7	8
j2	1	4	3	6	7	8	5
j3	4	1	2	7	8	5	6
j4	3	2	1	8	5	6	7
j5	6	7	8	1	4	3	2
j6	5	8	7	2	1	4	3
j7	8	5	6	3	2	1	4
j8	7	6	5	4	3	2	1

Ejemplos de ejecución serían los siguientes (las **filas** indican los *jugadores* y las **columnas** los *días*):

```
2  3  4  5  6  7  8
1  4  3  6  7  8  5
4  1  2  7  8  5  6
3  2  1  8  5  6  7
6  7  8  1  4  3  2
5  8  7  2  1  4  3
8  5  6  3  2  1  4
7  6  5  4  3  2  1
```

n = 8

```
 2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
 1  4  3  6  7  8  5 10 11 12 13 14 15 16  9
 4  1  2  7  8  5  6 11 12 13 14 15 16  9 10
 3  2  1  8  5  6  7 12 13 14 15 16  9 10 11
 6  7  8  1  4  3  2 13 14 15 16  9 10 11 12
 5  8  7  2  1  4  3 14 15 16  9 10 11 12 13
 8  5  6  3  2  1  4 15 16  9 10 11 12 13 14
 7  6  5  4  3  2  1 16  9 10 11 12 13 14 15
10 11 12 13 14 15 16  1  8  7  6  5  4  3  2
 9 12 11 14 15 16 13  2  1  8  7  6  5  4  3
12  9 10 15 16 13 14  3  2  1  8  7  6  5  4
11 10  9 16 13 14 15  4  3  2  1  8  7  6  5
14 15 16  9 12 11 10  5  4  3  2  1  8  7  6
13 16 15 10  9 12 11  6  5  4  3  2  1  8  7
16 13 14 11 10  9 12  7  6  5  4  3  2  1  8
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

n = 16

Caso 2. Dado cualquier $n > 1$, para este caso debemos considerar a su vez los casos de que n sea **par** y termine el torneo en $n - 1$ días, y que n sea **impar** y termine el torneo en n días.

Caso 2.1: *n* es impar, solución al problema en *n* días

Supongamos ahora que el número de jugadores n es **impar** y que sabemos resolver el problema para un número *par* de jugadores. En este caso, existe una solución al problema en n días, que se construye a partir de la solución al problema para $n+1$ jugadores. Si n es **impar** entonces $n+1$ es *par*, y sea Sol[1..n+1][1..n] el cuadro solución de emparejamientos para $n+1$ jugadores. Entonces podemos obtener el cuado solución para n jugadores Tabla[1..n][1..n] como:

$$Tabla[jug, dia] = \begin{cases} Sol[jug, dia] & si \quad Sol[jug, dia] \neq n+1 \\ 0 & si \quad Sol[jug, dia] = n+1 \end{cases}$$

Es decir, utilizamos la convención de que un *0* en la posición *[i, j]* de la tabla indica que el jugador i descansa (no se enfrenta a nadie) el día j, y aprovechamos este hecho para construir el cuadro solución pedido. Por ejemplo, para el caso de $n = 3$ nos apoyamos en la tabla construida para $n = 4$ jugadores, eliminando la última fila y sustituyendo las apariciones del jugador número 4 por ceros (0):

	d1	**d2**	**d3**
j1	2	3	0
j2	1	0	3
j3	0	1	2

Ejemplos de ejecución serían los siguientes (las **filas** indican los *jugadores* y las **columnas** los *días*):

2	3	4	5	6	7	8	9	0
1	5	3	7	4	8	9	0	6
8	1	2	4	5	9	0	6	7
5	9	1	3	2	0	6	7	8
4	2	0	1	3	6	7	8	9
7	8	9	0	1	5	4	3	2
6	0	8	2	9	1	5	4	3
3	6	7	9	0	2	1	5	4
0	4	6	8	7	3	2	1	5

n = 9

2	3	4	5	6	7	8	9	10	11	12	13	0
1	4	3	6	7	9	5	10	11	12	13	0	8
4	1	2	7	10	5	6	11	12	13	0	8	9
3	2	1	11	5	6	7	12	13	0	8	9	10
6	7	12	1	4	3	2	13	0	8	9	10	11
5	13	7	2	1	4	3	0	8	9	10	11	12
0	5	6	3	2	1	4	8	9	10	11	12	13
9	10	11	12	13	0	1	7	6	5	4	3	2
8	11	10	13	0	2	12	1	7	6	5	4	3
11	8	9	0	3	12	13	2	1	7	6	5	4
10	9	8	4	12	13	0	3	2	1	7	6	5
13	0	5	8	11	10	9	4	3	2	1	7	6
12	6	0	9	8	11	10	5	4	3	2	1	7

n = 13

Caso 2.2: *n* es par, solución al problema en *n - 1* días

Ahora nos queda resolver el caso en que *n* es **par**, y para ello llamaremos ***m*** = *n / 2*. Utilizando la técnica de Divide y Vencerás se debe de encontrar una forma de resolver el problema para *n* jugadores suponiendo que lo tenemos resuelto para *m*, por lo que distinguiremos dos casos para ***m***:

1. Si *m* es **par**, sabemos que existe una solución para enfrentar a esos *m* jugadores entre sí en *m – 1* días. Éste va a constituir el cuadrante superior izquierdo de la solución para *n*. Los otros tres cuadrantes se van a construir de igual forma al caso en el que *n* es potencia de 2.
2. Si *m* es **impar**, su solución necesita *m* días. Esto va a volver a constituir el cuadrante superior izquierdo de la solución para el caso de *n* jugadores, pero ahora nos encontramos con que tiene algunos ceros (días que el jugador descansa), que necesitaremos rellenar convenientemente.

Para ello, tenemos en cuenta las siguientes restricciones para la generación de los correspondientes cuadrantes de la tabla o cuadro de emparejamientos:

1. El cuadrante inferior izquierdo va a construirse como anteriormente, es decir, va a enfrentar a los jugadores de número superior entre ellos, por lo que se obtiene sumando *n/2* a los valores del cuadrante superior que no sean cero.
2. El cuadrante superior derecho enfrenta a los jugadores con menores y mayores números y se va a obtener de forma similar a como lo construíamos antes, solo que no va a enfrentar a los jugadores de 1 a *n/2* contra *(n/2)+1* a *n* **en el día *n/2***, sino ocupando cada una de las posiciones vacías del cuadrante superior izquierdo. Los demás días del cuadrante superior derecho sí se van a obtener rotando esos valores cada día.
3. De forma similar, el cuadrante inferior derecho enfrenta a los jugadores de mayor número contra los de menor número, y se va a obtener enfrentando a los jugadores *(n/2)+1* a *n* contra *1* a *n/2* respectivamente, pero **no en el día *n/2***, sino ocupando las posiciones vacías del cuadrante inferior izquierdo. Los valores restantes sí se obtendrán como antes, rotando los valores *1* a *n* cada día, pero en sentido contrario a como se ha hecho para el cuadrante superior.

Este proceso se ilustra en la siguiente figura para *n = 6*:

	d1	**d2**	**d3**
j1	2	3	0
j2	1	0	3
j3	0	1	2

m = 3

	d1	**d2**	**d3**	**d4**	**d5**
j1	2	3	0		
j2	1	0	3		
j3	0	1	2		
j4	5	6	0		
j5	4	0	6		
j6	0	4	5		

cuadrantes 1º y 2º

	d1	**d2**	**d3**	**d4**	**d5**
j1	2	3	4	5	6
j2	1	5	3	6	4
j3	6	1	2	4	5
j4	5	6	1	3	2
j5	4	2	6	1	3
j6	3	4	5	2	1

cuadrantes 3º y 4º

Ejemplos de ejecución serían los siguientes (las **filas** indican los *jugadores* y las **columnas** los *días*):

```
 2  3  4  5  6  7  8  9 10
 1  5  3  7  4  8  9 10  6
 8  1  2  4  5  9 10  6  7
 5  9  1  3  2 10  6  7  8
 4  2 10  1  3  6  7  8  9
 7  8  9 10  1  5  4  3  2
 6 10  8  2  9  1  5  4  3
 3  6  7  9 10  2  1  5  4
10  4  6  8  7  3  2  1  5
 9  7  5  6  8  4  3  2  1
```

n = 10

```
 2  3  4  5  6  7  8  9 10 11 12 13 14
 1  4  3  6  7  9  5 10 11 12 13 14  8
 4  1  2  7 10  5  6 11 12 13 14  8  9
 3  2  1 11  5  6  7 12 13 14  8  9 10
 6  7 12  1  4  3  2 13 14  8  9 10 11
 5 13  7  2  1  4  3 14  8  9 10 11 12
14  5  6  3  2  1  4  8  9 10 11 12 13
 9 10 11 12 13 14  1  7  6  5  4  3  2
 8 11 10 13 14  2 12  1  7  6  5  4  3
11  8  9 14  3 12 13  2  1  7  6  5  4
10  9  8  4 12 13 14  3  2  1  7  6  5
13 14  5  8 11 10  9  4  3  2  1  7  6
12  6 14  9  8 11 10  5  4  3  2  1  7
 7 12 13 10  9  8 11  6  5  4  3  2  1
```

n = 14

Se debe de destacar que, éstos son ejemplos de algoritmos Divide y Vencerás de **simplificación** (como hemos visto en clase de teoría, estos algoritmos solo contienen *una llamada recursiva*), es decir, en donde el problema se reduce en cada paso a un solo problema del mismo tipo y de tamaño más pequeño, mientras que la fase de combinación puede ser más costosa dependiendo de la naturaleza del problema que pretendemos resolver.

Además de los algoritmos DyV anteriores, se va a **implementar otro algoritmo DyV para el caso 1** (*n* es potencia de dos ($n = 2^k$), y cada participante jugará un partido por día durante *n-1* días). Para ello, utilizaremos la técnica algorítmica de Divide y Vencerás (**no** siendo un algoritmo de simplificación como los casos anteriores) de la siguiente manera. Primero dividimos a los jugadores en dos grupos iguales y que juegan dentro de los grupos durante los primeros *n/2 – 1* días. Luego, se programarán los partidos entre los grupos para los otros *n/2* días.

La solución debe ir expresada en una matriz bidimensional denominada **`tabla`** de *n-1* días (**filas**) por *n* jugadores (**columnas**) donde `tabla[d, i]` expresa contra quien se enfrenta el jugador *i* el día *d*. El algoritmo **DyV(tabla, i, j)** debe ser recursivo para generar un cuadro del torneo de tenis en `tabla` para los *jugadores* de ***i*** a ***j*** durante los *días* de *1* a *j-i*.

Para implementar el algoritmo recursivo, debemos considerar los siguientes casos:

1. **Caso base**: DyV(tabla, **i, i+1**) para 2 jugadores (**j == i +1**). Se programa para que jueguen entre ellos el primer día.
2. **Caso recursivo**: DyV(tabla, **i, j**) para **j > i+1**. Debemos tener en cuenta que estamos suponiendo que el número de jugadores **n = (j-i+1)** es una potencia de 2. Sea $k = i + n/2$ (es decir, **k = i + ((j - i + 1) / 2)**). Luego llamamos a **DyV(A, i, k-1)** y **DyV(A, k, j)**. Con lo que se genera un cuadro de torneo de todos contra todos entre la primera mitad de jugadores y la segunda mitad de los mismos en los días *1 a n/2-1*.
3. Finalmente, necesitamos **emparejar** a cada jugador de la primera mitad con todos los jugadores en la segunda mitad y para ello lo realizaremos de la siguiente manera. Para **m** = 0 a n/2, emparejamos al jugador *i* contra el jugador *k+m* el día *n/2 + m*. Emparejamos al jugador *i+1* contra el jugador *k+(l+m)%(n/2)* el día *n/2 + m*. En particular, el día *n-1*, el jugador *i+1* juega contra el jugador *k*. Continuamos así para todos los jugadores en la primera mitad. En particular, emparejamos al jugador *i+l* contra el jugador *k+(l+m)%(n/2)* el día *n/2 + m*.

La forma de llamar al algoritmo implementado sería como **DyV(tabla, 1, n)** y luego se debe de mostrar la matriz asociada columna por columna (recordemos que en este caso la matriz (`tabla`) bidimensional, las filas representan los *días* y la columnas los *jugadores*).

Ejemplos de ejecución serían los siguientes (recordad que para este caso las **filas** indican los *días* y las **columnas** los *jugadores*):

```
2  1  4  3  6  5  8  7
3  4  1  2  7  8  5  6
4  3  2  1  8  7  6  5
5  6  7  8  1  2  3  4
6  7  8  5  4  1  2  3
7  8  5  6  3  4  1  2
8  5  6  7  2  3  4  1
```

n = 8

```
 2  1  4  3  6  5  8  7 10  9 12 11 14 13 16 15
 3  4  1  2  7  8  5  6 11 12  9 10 15 16 13 14
 4  3  2  1  8  7  6  5 12 11 10  9 16 15 14 13
 5  6  7  8  1  2  3  4 13 14 15 16  9 10 11 12
 6  7  8  5  4  1  2  3 14 15 16 13 12  9 10 11
 7  8  5  6  3  4  1  2 15 16 13 14 11 12  9 10
 8  5  6  7  2  3  4  1 16 13 14 15 10 11 12  9
 9 10 11 12 13 14 15 16  1  2  3  4  5  6  7  8
10 11 12 13 14 15 16  9  8  1  2  3  4  5  6  7
11 12 13 14 15 16  9 10  7  8  1  2  3  4  5  6
12 13 14 15 16  9 10 11  6  7  8  1  2  3  4  5
13 14 15 16  9 10 11 12  5  6  7  8  1  2  3  4
14 15 16  9 10 11 12 13  4  5  6  7  8  1  2  3
15 16  9 10 11 12 13 14  3  4  5  6  7  8  1  2
16  9 10 11 12 13 14 15  2  3  4  5  6  7  8  1
```

n = 16

Una vez comprobado el correcto funcionamiento de todos los algoritmos requeridos sobre los datos según los ejemplos planteados. Los ejecutaremos sobre los diferentes números de participantes (*n*) que se espera en el torneo de tenis.

Trabajo a desarrollar

Como ya sabemos, el método **divide-and-conquer** (divide y vencerás) consiste en descomponer el problema que hay que resolver en una serie de subproblemas, resolver estos subproblemas y después combinar los resultados para obtener la solución del problema original. Lo importante en este método es que los subproblemas deben de ser del mismo tipo que el problema original, pero de menor tamaño, y se resuelven utilizando la misma técnica. El número de subproblemas debe ser pequeño e independiente de una entrada determinada. En el caso particular de los algoritmos Divide y Vencerás que contienen sólo una llamada recursiva, tenemos los algoritmos de **simplificación**. La ventaja de estos algoritmos es que consiguen reducir el tamaño del problema en cada paso, por lo que sus tiempos de ejecución suelen ser relativamente buenos.

Para resolver el problema anteriormente planteado (generación del cuadro de un torneo de tenis de todos contra todos (liga)) hay que realizar lo siguiente:

1. Implementar en Java el algoritmo de simplificación recursivo para resolver el **caso 1**, ilustrando y explicando su ejecución para algún valor de *n*. Calcular su complejidad (espacial y temporal). Probar para distintos valores de *n* (potencia de 2) y justificar su comportamiento.
2. Implementar en Java el algoritmo de simplificación recursivo para resolver el **caso 2**, ilustrando y explicando su ejecución para algún valor de *n* (par o impar). Calcular su complejidad (espacial y temporal). Probar para distintos valores de *n* (*n* par y *n* impar) y justificar su comportamiento.
3. Implementar en Java algoritmo Divide y Vencerás recursivo para resolver el **caso 1** (*n* potencia de 2), siguiendo las restricciones impuestas, e ilustrando y explicando su ejecución para algún valor de *n*. Calcular su complejidad (espacial y temporal). Probar para distintos valores de *n* (potencia de 2) y justificar su comportamiento.

Para ello deberá realizar los siguientes apartados:

- **Estudio de la implementación**: Explicar los detalles más importantes de las implementaciones de los algoritmos requeridos. El código debe de estar razonablemente bien documentado (JavaDoc).
- **Estudio teórico**: Estudiar los tiempos de ejecución de los algoritmos implementados, en función del número de elementos que componen la entrada, *n* (número de jugadores). Se debe tener en cuenta los valores de *n* según los casos (*n* potencia de dos, *n* par o impar)
- **Estudio experimental**: Validación de los algoritmos implementados sobre los distinto valores de *n* (participantes en **EDAland-Garros**). Se contrastarán los resultados teóricos y los experimentales, comprobando si los experimentales confirman los teóricos previamente analizados. Se justificarán los experimentos realizados, y en caso de discrepancia entre la teoría y los experimentos se debe intentar buscar una explicación razonada. Además, se recomienda generar resultados según **diferentes valores de *n*** (suficientemente grandes, $\mathbf{n = 2^k}$ y valores pares e impares próximos a esos valores de $\mathbf{2^k}$) para comprobar tanto el *tiempo de respuesta* (a nivel de milisegundos) que tardan los algoritmos implementados como la *memoria principal* (a nivel de MB) que ocupa la tabla resultante.

Entregas

Se ha de entregar, en fecha, en el repositorio GitHub (mismo repositorio para todas las prácticas de EDA II) con toda la documentación y código fuente requerido en la práctica:

- En dicho repositorio en la carpeta de fuentes `src/main/java`, crear un nuevo paquete llamado `org.eda2.practica01`, para el **código fuente** de la práctica.
- Al mismo nivel de `src` crear una carpeta "normal", denominada `docs`, para la **documentación**. En ella hay que crear una subcarpeta denominada `practica01` para guardar toda la documentación (documento en `pdf` y los fuentes utilizados para su creación (por ejemplo, `.docx`).
- **Memoria** que explique todo lo que habéis realizado/requerido en la práctica. La memoria deberá tener el formato que se indica a continuación. Si se desea, también se podrá realizar una presentación de la práctica.
- **Código fuente** de la aplicación, desarrollada en JAVA o en C++, que resuelva correctamente todo lo planteado en la práctica.
- **Juegos de prueba** que consideréis oportunos incluir para justificar que todo funciona correctamente (verificando los casos planteados y sus correspondientes figuras). En este caso los juegos de prueba deben estar en una nueva carpeta de fuentes denominada `src/test/java` y dentro de ella en un paquete denominado `org.eda2.practica01`.

La **memoria de la práctica** a entregar debe ser breve, clara y estar bien escrita, adaptándose en lo posible a la norma UNE 157001:2014 "Criterios generales para la elaboración de proyectos". Ésta debe incluir las siguientes secciones:

- Un apartado **Objetivo** con un estudio teórico del método algorítmico utilizado en esta práctica (divide-and-conquer). Además, en esta sección deberá exponerse claramente ***qué miembro*** **del grupo ha actuado como** ***líder*** **y** ***qué tareas ha realizado cada miembro del grupo***. Es muy importante que todos los miembros dominen la práctica en su conjunto. Se puede incorporar la hoja de datos indicada para tal fin (`PR1_Tareas_a_repartir.xlsx`).
- Un apartado **Antecedentes** en el que se explique el motivo que lleva a realizar esta memoria y se enumeran los aspectos necesarios, en su caso, de las alternativas contempladas y la solución final adoptada.
- Una sección para cada uno de **apartados propuestos** a desarrollar en esta práctica (estudio de la implementación, estudio teórico y estudio experimental). Para el **estudio experimental**, el correcto funcionamiento de los algoritmos debe de justificarse con la correspondiente captura de pantalla en la que se pueda apreciar la salida correcta. Hemos de remarcar que deben incluirse los apartados en el mismo orden en el que se han expuesto. El apartado **estudio experimental** de la memoria recogerá los resultados finales de todo lo realizado.
- Se incluirá también un **anexo** con el diseño del código implementado, con diagramas de clases y cualquier otro diagrama que estiméis necesario incluir, **no introducir ningún código en este anexo**. Además, añadir en esta sección una lista de los archivos fuente y una breve descripción del contenido de cada uno.
- Un **anexo** con los cálculos realizados en el **estudio experimental**, con la explicación del método seguido para la toma de datos y gráficos con el resultado obtenido (datos de muestras y ecuación seleccionada). Cuando se elija una curva para explicar el orden de un algoritmo (por ejemplo, $O(n^2)$), se deberá añadir el estadístico de la varianza explicada R^2, de forma que este valor (que varía entre 0 y 1) debería ser superior a 0.95 para que se acepte la solución propuesta. Valores por debajo 0.90 indican que el proceso de toma de muestras o el de elaboración de la hipótesis, ha sido incorrecto y debería llevar a su revisión. Cada muestra debería ser una media de varias medidas (al

menos 10) de tiempo de ejecución y el tiempo medido debería ser, al menos, cercano al segundo para evitar interferencias del sistema operativo.

- Es importante incluir siempre las **fuentes bibliográficas** utilizadas (web, libros, artículos, etc.) y hacer referencia a ellas en el documento.

Evaluación

Cada apartado se evaluará independientemente, aunque es condición necesaria para aprobar la práctica que los programas implementados funcionen correctamente.

- La implementación junto con la documentación del código se valorará sobre un 40%
- El estudio de la implementación se valorará sobre un 10%
- El estudio teórico se valorará sobre un 15%
- El estudio experimental se valorará sobre un 35%

Se penalizará de forma importante no entregar el apartado de introducción teórico o una mala presentación de la memoria. Debemos insistir en lo importante que es la redacción de una buena memoria para poder optar a la máxima nota en la práctica.

Se podrá requerir la defensa del código y de la memoria por parte del profesor.

Fecha de entrega: 31 de Marzo de 2024