

# PREDA - UNED

Prácticas de curso – 2019/20

# Histórico

Curso	Nº práctica	Esquema	Tema
<b>2011/12</b>	1	DyV	Skyline
	2	RyP	Viajante de comercio
<b>2012/13</b>	1	DyV	Organización calendario liga deportiva
	2	VA	Problema del caballo
<b>2013/14</b>	1	Voraz	Mensajería urgente
	2	PD	Devolución de cambio en monedas
<b>2014/15</b>	1	PD	Multiplicación asociativa de matrices
	2	VA	Coloreado de grafos
<b>2015/16</b>	1	Voraz	Minimización del tiempo en el sistema
	2	PD	Coeficientes binomiales
<b>2016/17</b>	1	DyV	Multiplicación de grandes números
	2	VA	N reinas
<b>2017/18</b>	1	Voraz	Robot en un circuito
	2	PD	Mochila no fraccionable
<b>2018/19</b>	1	VA	Subconjuntos de suma dada
	2	PD	Distancia de edición

# 2019-20

- **Práctica 1:**
  - **Cálculo del elemento mayoritario de un vector (DyV)**
- **Práctica 2:**
  - **Reparto equitativo de activos (VA)**

# Normas

- Los estudiantes que tengan aprobadas ambas prácticas en el **curso anterior** (sólo el anterior) no es necesario que vuelvan a realizarlas en este curso.
- Para que el examen sea calificado el alumno deberá:
  - haber asistido a las **sesiones presenciales** de prácticas
  - haber **entregado y aprobado las prácticas** obligatorias
    - Si se entregan y aprueban las prácticas en **septiembre**, hay que presentarse necesariamente al examen de septiembre

# Entrega

- Fechas
  - Convocatoria de Febrero:
    - Domingo 19 de enero de 2020
  - Convocatoria de Septiembre:
    - Domingo 23 de agosto de 2020
- Doble entrega necesaria
  - Por email: [fenros@us.es](mailto:fenros@us.es)
  - En la plataforma ALF
    - A través del enlace habilitado en el apartado *“Entrega de trabajos”*

# Entrega

- **¿Todos inscritos al grupo de prácticas del centro asociado de Sevilla?**
- Plataforma ALF
- Entrega de trabajos -> Apuntarse a grupo de prácticas

**Lista de alumnos del tutor de Programación y Estructuras de Datos Avanzadas  
Fernando Enríquez de Salamanca en el centro Sevilla**

Introduzca su DNI/Pasaporte para apuntarse a la lista de alumnos de este tutor

# Evaluación de las prácticas

- Imprescindible: que el **programa completo compile y funcione**
- Sobresaliente: **excelente documentación y calidad del código**, que además cumplan los requisitos imprescindibles de correcto funcionamiento y aplicación de la estructura de datos o esquema adecuados
- Evaluación:

ASPECTO DE LA PRÁCTICA	NOTA SOBRE 10
Utilización óptima de la estructura de datos o el esquema	2.5
Estilo de programación y calidad del código	2.5
Documentación presentada	2
Eficiencia del algoritmo	2
Posibles mejoras introducidas por el alumno a los requisitos básicos de la práctica	1

# Material a entregar (formato ZIP)

- Archivo .pdf con la siguiente información:
  - Datos de la asignatura
    - Nombre y Código
    - Título de la práctica
    - Centro Asociado
  - Datos del alumno
    - Nombre, apellidos, NIF, teléfono y correo electrónico
  - Respuestas a los apartados:
    - Descripción del esquema utilizado y su adecuación al problema (Seguir las directrices del libro)
    - Análisis del coste computacional
    - Estudio de otras alternativas y comparativa
    - Descripción de los casos de prueba realizados y sus resultados
- Código fuente Java (diseño orientado a objetos)
  - Adecuadamente documentado
  - Ejecutable (.jar) y directorio con fuentes (.java)

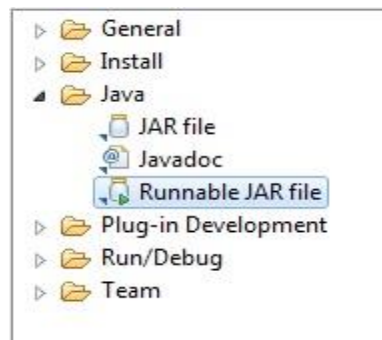


# Crear .jar ejecutable

- BlueJ: Project -> Create Jar File -> Main class



- Eclipse: File -> Export -> Java -> runnable Jar file



# Errores a evitar en la entrega

- Código:
  - No compila
  - No está desarrollado en Java
  - No se corresponde con el libro
  - No es original, está copiado
  - No sigue un diseño OO encapsulado o modular
- Ejecutable:
  - No termina
  - Se queda sin memoria con ejemplos pequeños
  - Aborta sin justificación
  - No lee los ficheros previstos en el formato adecuado
  - No trata los argumentos
  - No se ajusta a las especificaciones
- Documentación:
  - No se presenta como ha indicado el tutor
  - Está incompleta
- Soporte:
  - No se puede leer
  - Contiene virus (Suspenso)

# ¿Dudas?

- Foros de debate de la asignatura en ALF
  - Foro de la práctica
  - Grupo de tutoría de Sevilla
- Correo electrónico al tutor
  - Fernando Enríquez de Salamanca Ros: [fenros@us.es](mailto:fenros@us.es)
- Sesiones de monitorización (obligatorias)
  - Se resolverán colectivamente las dudas más comunes e individualmente las dudas particulares de cada alumno

# Práctica 1

Cálculo del elemento mayoritario de un vector

# Enunciado

- Dado un vector  $v[1..n]$  de número naturales, se quiere averiguar si existe un elemento mayoritario, es decir, que aparezca al menos  $n/2+1$  veces en el vector ( $n/2$  es división entera)
- Se pide diseñar un algoritmo, siguiendo el esquema de **divide y vencerás**, que resuelva este problema
- Ejemplos:
  - $[4,2,1,3,3,3,2,3] \rightarrow$  no tendría elemento mayoritario (tiene 8 elementos y ninguno aparece 5 veces o más)
  - $[3,1,4,5,3,3,3,2,3] \rightarrow$  el 3 es el elemento mayoritario (tiene 9 elementos y el 3 aparece 5 veces)

# Estrategia – Divide y Vencerás (C4)

1. Descomposición del problema en subproblemas de su mismo tipo o naturaleza
  - Disminuir la complejidad y ¿paralelizar?
  - Para casos sencillos se aporta solución trivial (caso base de la recursión)
2. Resolución recursiva de los subproblemas
3. Combinación, si procede, de las soluciones de los subproblemas

# Elementos

- Trivial y solución-trivial:
  - ¿Cuándo se convierte el problema en **trivial**?  
Cuando una función pueda resolverlo sin descomponerlo
- Descomponer:
  - **Dividir** el problema en subproblemas más pequeños que el inicial
  - El tipo de sucesión formada por los sucesivos tamaños del problema y el nº de subproblemas determinan la **complejidad**
- Combinar:
  - Aplicando el principio de inducción, se dan los subproblemas por resueltos y lo que queda es cómo **combinar** las soluciones de los subproblemas para obtener la solución del problema final
  - Si no hay que realizar combinación de soluciones el tipo de problema se conoce como **reducción**

# Esquema general

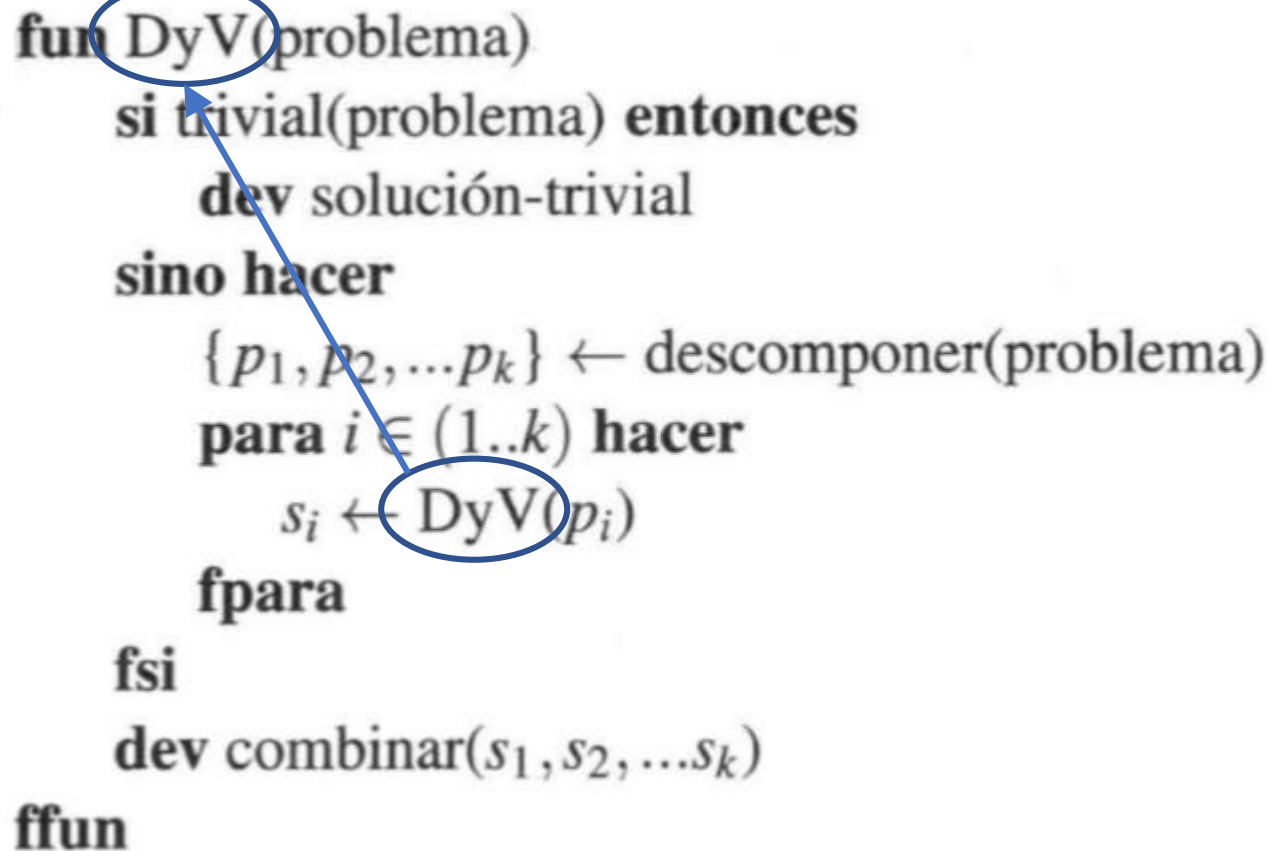
```
fun DyV(problema)
  si trivial(problema) entonces
    dev solución-trivial
  sino hacer
     $\{p_1, p_2, \dots, p_k\} \leftarrow \text{descomponer}(\text{problema})$ 
    para  $i \in (1..k)$  hacer
       $s_i \leftarrow \text{DyV}(p_i)$ 
    fpara
  fsi
  dev combinar( $s_1, s_2, \dots, s_k$ )
ffun
```



# Esquema general

Función recursiva

```
fun DyV(problema)
  si trivial(problema) entonces
    dev solución-trivial
  sino hacer
     $\{p_1, p_2, \dots, p_k\} \leftarrow \text{descomponer}(\text{problema})$ 
    para  $i \in (1..k)$  hacer
       $s_i \leftarrow \text{DyV}(p_i)$ 
    fpara
    fsi
    dev combinar( $s_1, s_2, \dots, s_k$ )
ffun
```



The diagram illustrates the recursive nature of the DyV function. A blue circle highlights the function name 'DyV' in the first line. A blue arrow points from this circle to a yellow box labeled 'Función recursiva'. Another blue circle highlights the recursive call 'DyV(p\_i)' in the line 's\_i ← DyV(p\_i)'. A blue arrow points from this circle back to the 'DyV' in the first line, indicating a recursive call.

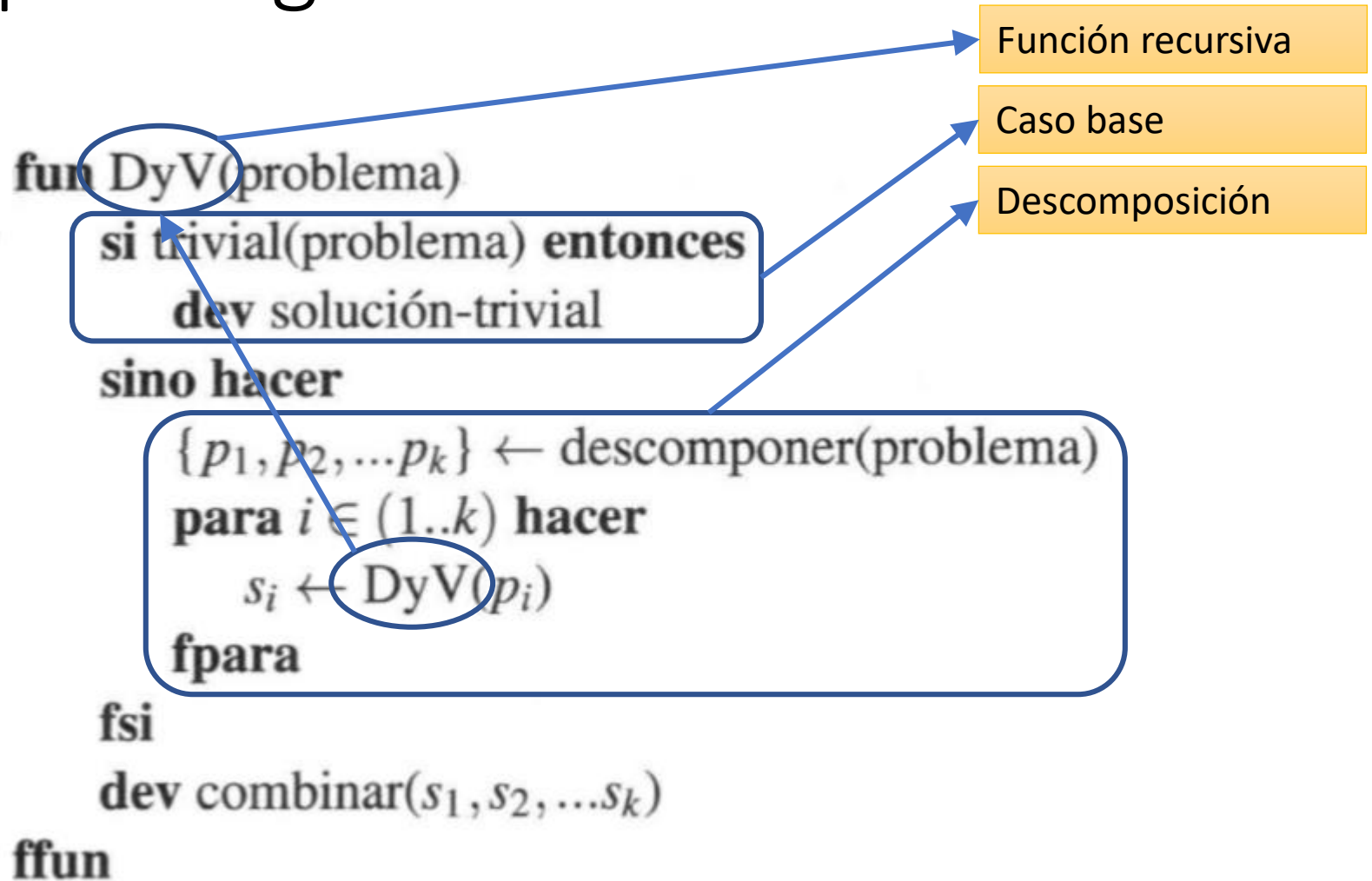
# Esquema general

```
fun DyV(problema)
  si trivial(problema) entonces
    dev solución-trivial
  sino hacer
     $\{p_1, p_2, \dots, p_k\} \leftarrow \text{descomponer}(\text{problema})$ 
    para  $i \in (1..k)$  hacer
       $s_i \leftarrow \text{DyV}(p_i)$ 
    fpara
    fsi
    dev combinar( $s_1, s_2, \dots, s_k$ )
ffun
```

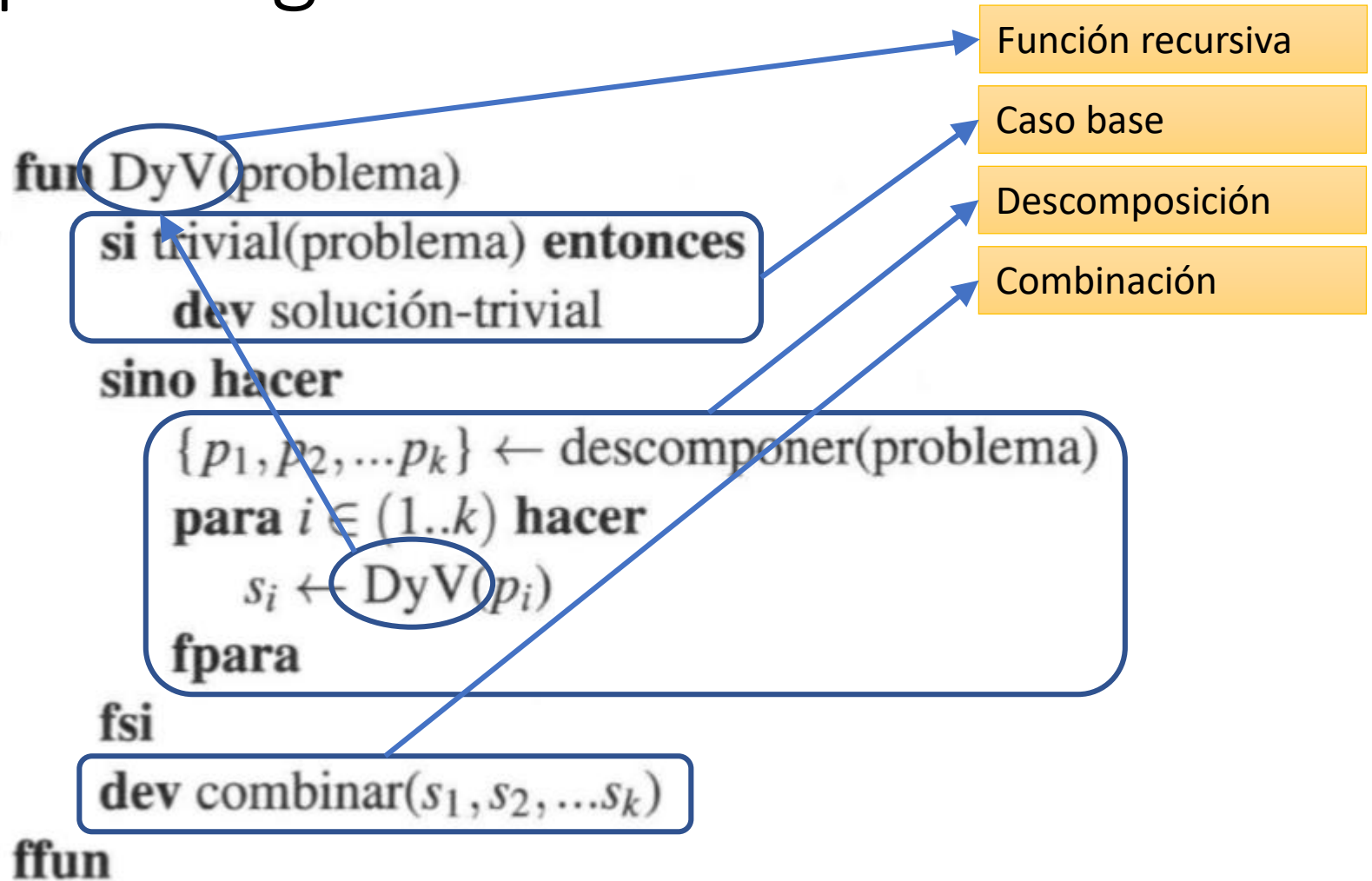
Función recursiva

Caso base

# Esquema general



# Esquema general



# Cálculo del elemento mayoritario

- Dado un vector  $v[1..n]$  de números naturales, se quiere averiguar si existe un elemento mayoritario (que aparezca al menos  $n/2 + 1$  veces en el vector)

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

Conteos de cada número:

- 5: 6 ← MAYORITARIO ( $n/2 + 1 = 6$ )
- 9: 2
- 2: 1
- 7: 1

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
  var
    m:natural
    s1,s2:entero
  fvar
    si  $i = j$  entonces
      dev  $v[i]$ 
    sino
       $m \leftarrow (i + j) \div 2$ 
       $s_1 \leftarrow \text{Mayoritario}(i,m,v)$ 
       $s_2 \leftarrow \text{Mayoritario}(m+1,j,v)$ 
      dev Combinar( $s_1, s_2$ )
ffun
```

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1, s2:entero
```

```
  fvar
```

```
  si i = j entonces
```

```
    dev v[i]
```

```
  sino
```

```
    m ← (i + j) ÷ 2
```

```
    s1 ← Mayoritario(i, m, v)
```

```
    s2 ← Mayoritario(m + 1, j, v)
```

```
    dev Combinar(s1, s2, v)
```

```
ffun
```

Función recursiva

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1,s2:entero
```

```
  fvar
```

```
    si i = j entonces
```

```
      dev v[i]
```

```
    sino
```

```
      m ← (i + j) ÷ 2
```

```
      s1 ← Mayoritario(i,m,v)
```

```
      s2 ← Mayoritario(m+1,j,v)
```

```
      dev Combinar(s1,s2,v)
```

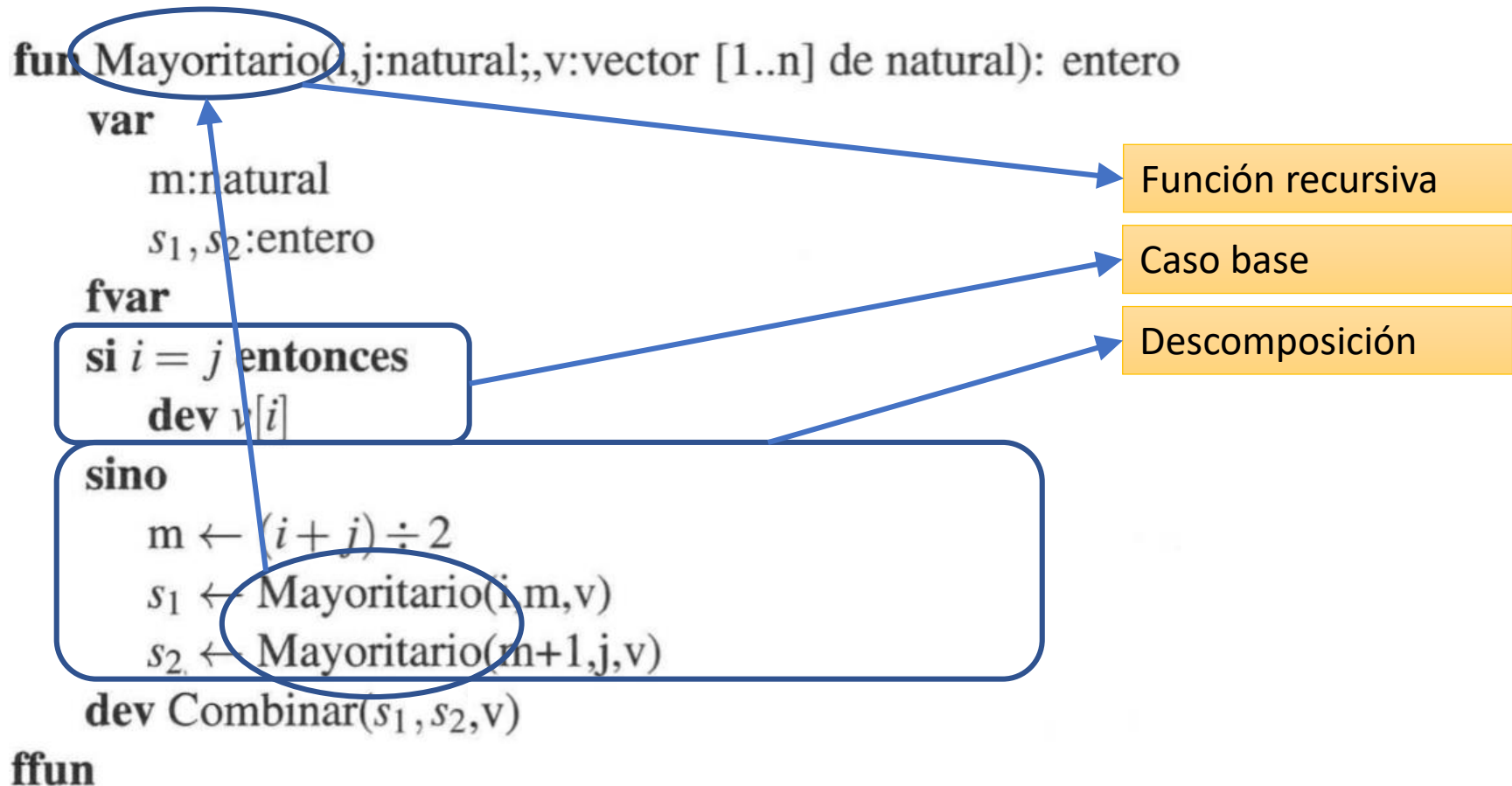
```
ffun
```

Función recursiva

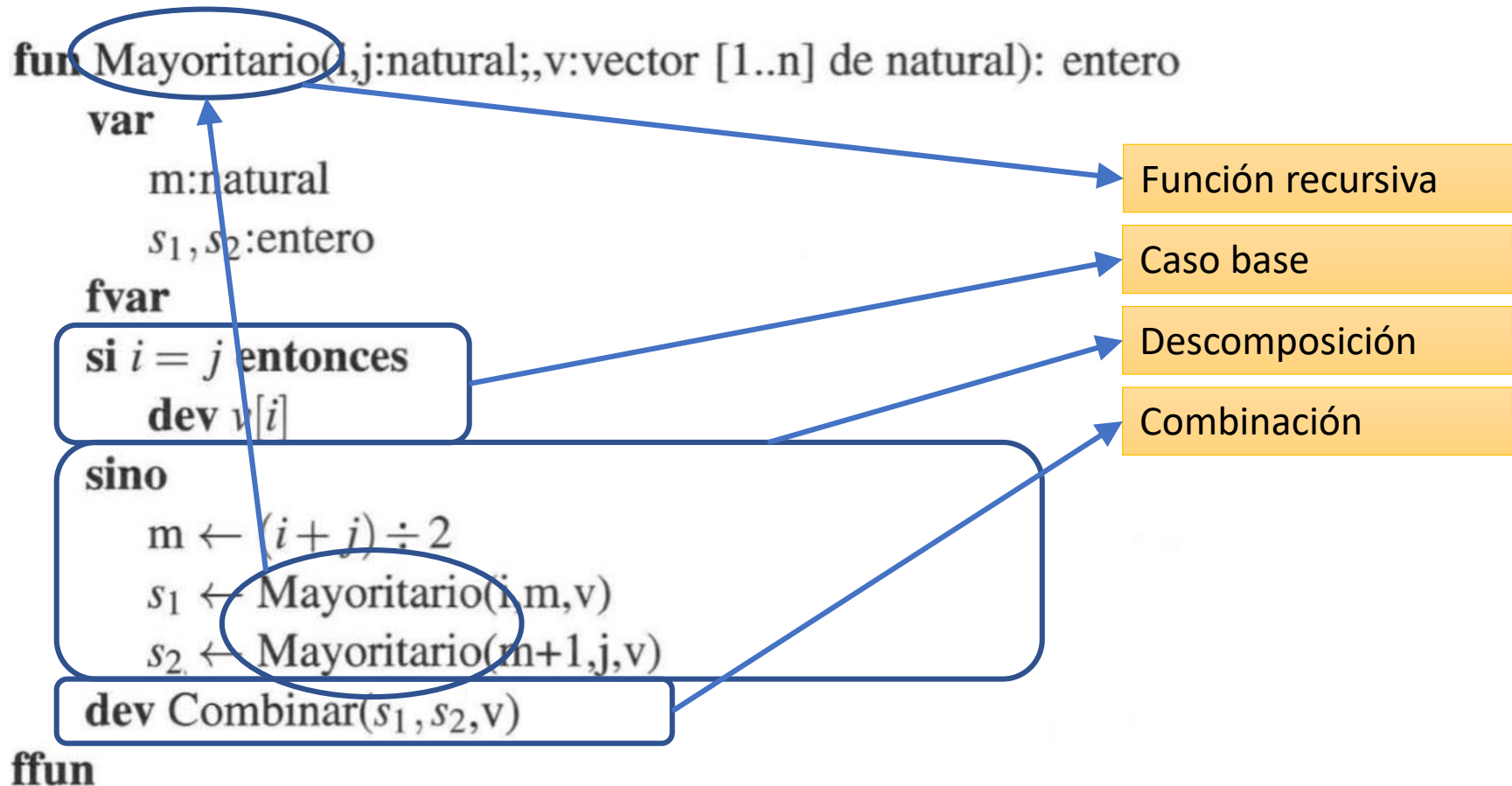
Caso base



# Cálculo del elemento mayoritario



# Cálculo del elemento mayoritario



# Cálculo del elemento mayoritario

**fun** Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero

**var**

m:natural

s<sub>1</sub>, s<sub>2</sub>:entero

**fvar**

**si** i = j **entonces**

**dev** v[i]

**sino**

m ← (i + j) ÷ 2

s<sub>1</sub> ← Mayoritario(i,m,v)

s<sub>2</sub> ← Mayoritario(m+1,j,v)

**dev** Combinar(s<sub>1</sub>, s<sub>2</sub>, v)

**ffun**

**fun** Combinar(a,b:entero;v:vector [1..n] de natural):entero

**si** a = -1 ∧ b = -1 **entonces dev** -1 **fsi**

**si** a = -1 ∧ b ≠ -1 **entonces dev** ComprobarMayoritario(b,v) **fsi**

**si** a ≠ -1 ∧ b = -1 **entonces dev** ComprobarMayoritario(a,v) **fsi**

**si** a ≠ -1 ∧ b ≠ -1 **entonces**

**si** ComprobarMayoritario(a,v) = a **entonces dev** a

**sino si** ComprobarMayoritario(b,v) = b **entonces dev** b **fsi**

**fsi**

**fsi**

**fun**

Añadir parámetros i y j:  
ComprobarMayoritario  
busca en v[i..j] no en v

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1, s2:entero
```

```
  fvar
```

```
  si i = j entonces
```

```
    dev v[i]
```

```
  sino
```

```
    m ← (i + j) ÷ 2
```

```
    s1 ← Mayoritario(i,m,v)
```

```
    s2 ← Mayoritario(m+1,j,v)
```

```
  dev Combinar(s1, s2)
```

```
ffun
```

```
fun Combinar(a,b:entero;v:vector [1..n] de natural):entero
```

```
  fun ComprobarMayoritario (x:natural;v:vector [1..n] de natural):entero
```

```
    var
```

```
      c:natural
```

```
    fvar
```

```
      c ← 1
```

```
    para k ← 1 hasta n hacer
```

```
      si v[k]=x entonces c ← c + 1 fsi
```

```
    fpara
```

```
    si c > (j - i + 1) / 2 entonces dev c sino dev -1 fsi
```

```
  ffun
```

Añadir parámetros i y j

Ojo a las erratas  
en el libro

Ejemplo: Mayoritario(7,12,v)

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

s<sub>1</sub>=2

s<sub>2</sub>=3

ComprobarMayoritario(2,3,7,12,v) → 2

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1, s2:entero
```

```
  fvar
```

```
  si i = j entonces
```

```
    dev v[i]
```

```
  sino
```

```
    m ← (i + j) ÷ 2
```

```
    s1 ← Mayoritario(i,m,v)
```

```
    s2 ← Mayoritario(m+1,j,v)
```

```
  dev Combinar(s1, s2, v)
```

```
ffun
```

¿Coste?

```
fun Combinar (a,b:entero;v:vector [1..n] de natural):entero
```

```
  si a = -1 ∧ b = -1 entonces dev -1 fsi
```

```
  si a = -1 ∧ b ≠ -1 entonces dev ComprobarMayoritario(b,v) fsi
```

```
  si a ≠ -1 ∧ b = -1 entonces dev ComprobarMayoritario(a,v) fsi
```

```
  si a ≠ -1 ∧ b ≠ -1 entonces
```

```
    si ComprobarMayoritario(a,v) = a entonces dev a
```

```
    sino si ComprobarMayoritario(b,v) = b entonces dev b fsi
```

```
  fsi
```

```
  fsi
```

```
fun
```

```
fun ComprobarMayoritario (x:natural;v:vector [1..n] de natural):entero
```

```
  var
```

```
    c:natural
```

```
  fvar
```

```
  c ← 1
```

```
  para k ← 1 hasta n hacer
```

```
    si v[k]=x entonces c ← c + 1 fsi
```

```
  fpara
```

```
  si c > (j - i + 1) / 2 entonces dev c sino dev -1 fsi
```

```
ffun
```

Añadir parámetros i y j

Ojo a las erratas  
en el libro

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1, s2:entero
```

```
  fvar
```

```
  si i = j entonces
```

```
    dev v[i]
```

```
  sino
```

```
    m ← (i + j) ÷ 2
```

```
    s1 ← Mayoritario(i,m,v)
```

```
    s2 ← Mayoritario(m+1,j,v)
```

```
  dev Combinar(s1, s2, v)
```

```
ffun
```

¿Coste?

```
fun Combinar (a,b:entero;v:vector [1..n] de natural):entero
```

```
  si a = -1 ∧ b = -1 entonces dev -1 fsi
```

```
  si a = -1 ∧ b ≠ -1 entonces dev ComprobarMayoritario(b,v) fsi
```

```
  si a ≠ -1 ∧ b = -1 entonces dev ComprobarMayoritario(a,v) fsi
```

```
  si a ≠ -1 ∧ b ≠ -1 entonces
```

```
    si ComprobarMayoritario(a,v) = a entonces dev a
```

```
    sino si ComprobarMayoritario(b,v) = b entonces dev b fsi
```

```
  fsi
```

```
  fsi
```

```
fun
```

```
fun ComprobarMayoritario (x:natural;v:vector [1..n] de natural):entero
```

```
  var
```

```
    c:natural
```

```
  fvar
```

```
  c ← 1
```

```
  para k ← 1 hasta n hacer
```

```
    si v[k]=x entonces c ← c + 1 fsi
```

```
  fpara
```

```
  si c > (j - i + 1) / 2 entonces dev c sino dev -1 fsi
```

```
ffun
```

Añadir parámetros i y j

Ojo a las erratas  
en el libro

# Cálculo del elemento mayoritario

```
fun Mayoritario(i,j:natural;;v:vector [1..n] de natural): entero
```

```
  var
```

```
    m:natural
```

```
    s1,s2:entero
```

```
  fvar
```

```
    si i = j entonces
```

```
  fun Combinar(a,b:entero;v:vector [1..n] de natural):entero
```

```
    si a = -1 ∧ b = -1 entonces dev -1 fsi
```

```
    si a = -1 ∧ b ≠ -1 entonces dev ComprobarMayoritario(b,v) fsi
```

```
    si a ≠ -1 ∧ b = -1 entonces dev ComprobarMayoritario(a,v) fsi
```

```
    si a ≠ -1 ∧ b ≠ -1 entonces
```

Progresión  
geométrica

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n/b) + cn^k & , \text{ si } n \geq b \end{cases}$$



$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < b^k \\ \Theta(n^k \log n) & , \text{ si } a = b^k \\ \Theta(n^{\log_b a}) & , \text{ si } a > b^k \end{cases}$$

Progresión  
aritmética

$$T(n) = \begin{cases} cn^k & , \text{ si } 1 \leq n < b \\ aT(n-b) + cn^k & , \text{ si } n \geq b \end{cases}$$



$$T(n) \in \begin{cases} \Theta(n^k) & , \text{ si } a < 1 \\ \Theta(n^{k+1}) & , \text{ si } a = 1 \\ \Theta(a^{n/b}) & , \text{ si } a > 1 \end{cases}$$

$$T(n) = 2T(n/2) + cn$$

$$a = b = 2 \text{ y } k = 1$$

$$t(n) \in \Theta(n \log n)$$

```
  ivar
```

```
    c ← 1
```

```
  para k ← 1 hasta n hacer
```

```
    si v[k]=x entonces c ← c + 1 fsi
```

```
  fpara
```

```
    si c > (j - i + 1) / 2 entonces dev c sino dev -1 fsi
```

```
ffun
```

Ojo a las erratas  
en el libro

# Argumentos y parámetros

- Sintaxis:
  - `java mayoritario [-t][-h] [fichero_entrada] [fichero_salida]`
  - `java -jar mayoritario.jar [-t][-h] [fichero_entrada] [fichero_salida]`
- Argumentos:
  - `-t`: traza cada paso de manera que se describa la aplicación del algoritmo utilizado
  - `-h`: muestra una ayuda y la sintaxis del comando

Ejemplo:

```
$ java mayoritario -h <ENTER>
SINTAXIS:
mayoritario [-t][-h][fichero_entrada][fichero_salida]
-t          Traza las llamadas recursivas
-h          Muestra esta ayuda
fichero_entrada  Nombre del fichero de entrada
fichero_salida   Nombre del fichero de salida
```

- `fichero_entrada`: nombre del fichero del que se leen los datos de entrada
  - Contiene el número de elementos en el vector de enteros y el propio vector
  - Si la entrada no es correcta, el programa debe indicarlo
  - Si no existe el fichero, se utilizará la entrada estándar
- `fichero_salida`: es el nombre del fichero que se creará para almacenar la salida
  - Si el fichero ya existe, el comando dará un error
  - Si falta este argumento, el programa muestra el resultado por pantalla



# Argumentos y parámetros

¿Qué ocurre si no se indica ningún parámetro?

- Sintaxis:
  - `java mayoritario [-t][-h] [fichero_entrada] [fichero_salida]`
  - `java -jar mayoritario.jar [-t][-h] [fichero_entrada] [fichero_salida]`
- Argumentos:
  - `-t`: traza cada paso de manera que se describa la aplicación del algoritmo utilizado
  - `-h`: muestra una ayuda y la sintaxis del comando

Ejemplo:

```
$ java mayoritario -h <ENTER>
SINTAXIS:
mayoritario [-t][-h][fichero_entrada][fichero_salida]
-t           Traza las llamadas recursivas
-h           Muestra esta ayuda
fichero_entrada  Nombre del fichero de entrada
fichero_salida   Nombre del fichero de salida
```

- `fichero_entrada`: nombre del fichero del que se leen los datos de entrada
  - Contiene el número de elementos en el vector de enteros y el propio vector
  - Si la entrada no es correcta, el programa debe indicarlo
  - Si no existe el fichero, se utilizará la entrada estándar
- `fichero_salida`: es el nombre del fichero que se creará para almacenar la salida
  - Si el fichero ya existe, el comando dará un error
  - Si falta este argumento, el programa muestra el resultado por pantalla

# Argumentos y parámetros

- Sintaxis:

```
public class Mayoritario{  
    ...  
    public static void main(String[] args){  
        switch (args.length){  
            case 0: //no hay argumentos  
                break;  
            case 1: //args[0] puede ser -t, -h, fich_ent o fich_sal  
                break;  
            case 2: //args[0] y args[1]  
                break;  
            case 3: //args[0], args[1], args[2]  
                break;  
            case 4: //args[0], args[1], args[2], args[3]  
                break;  
            default: //demasiados argumentos  
                break;  
        }  
    }  
    ...  
}
```

ado

# Entrada / Salida

- Entrada

- El fichero de datos de entrada consta de 2 líneas con:
  - El valor del parámetro n, número de elementos en el vector
  - Los elementos del vector de enteros separados por espacios

- Ejemplo:

9

3 1 4 5 3 3 3 2 3

- Salida

- La salida es una línea con
  - El valor del elemento mayoritario si existe
  - Si no existe se indicará con un carácter especial, N
- Ejemplo (correspondiente al ejemplo de entrada):

3

# Código auxiliar

Fragmentos de código Java que pueden ser útiles

# Lectura de ficheros en java (I)

- Lectura de un fichero de texto línea a línea:

```
BufferedReader bf = new BufferedReader(new FileReader(nomFich));
```

```
String linea = bf.readLine();
```

```
// Si no quedan datos en el fichero, linea será null
```

```
// Para dividir una línea:
```

```
String[] datos = linea.split(" ")
```

```
// Para convertir a entero un String:
```

```
volumen = new Integer(datos[0])
```

```
// Hay que cerrar el fichero una vez leído
```

```
bf.close();
```

# Lectura de ficheros en java (II)

- Scanner

```
Scanner sc = new Scanner(new File(nomFich));  
while(sc.hasNextLine()){  
    String linea = sc.nextLine();  
    //procesar elemento  
}  
sc.close();
```

# Escritura de ficheros en java

- `PrintStream`

```
PrintStream ps = new PrintStream(new File(nomFich));
```

```
...
```

```
ps.println(elem); //ps.print(elem + "\n");
```

```
...
```

```
ps.close();
```