

Algoritmia  
Grado en Ingeniería Informática del Software  
Escuela de Ingeniería Informática – Universidad de Oviedo

# Divide y Vencerás

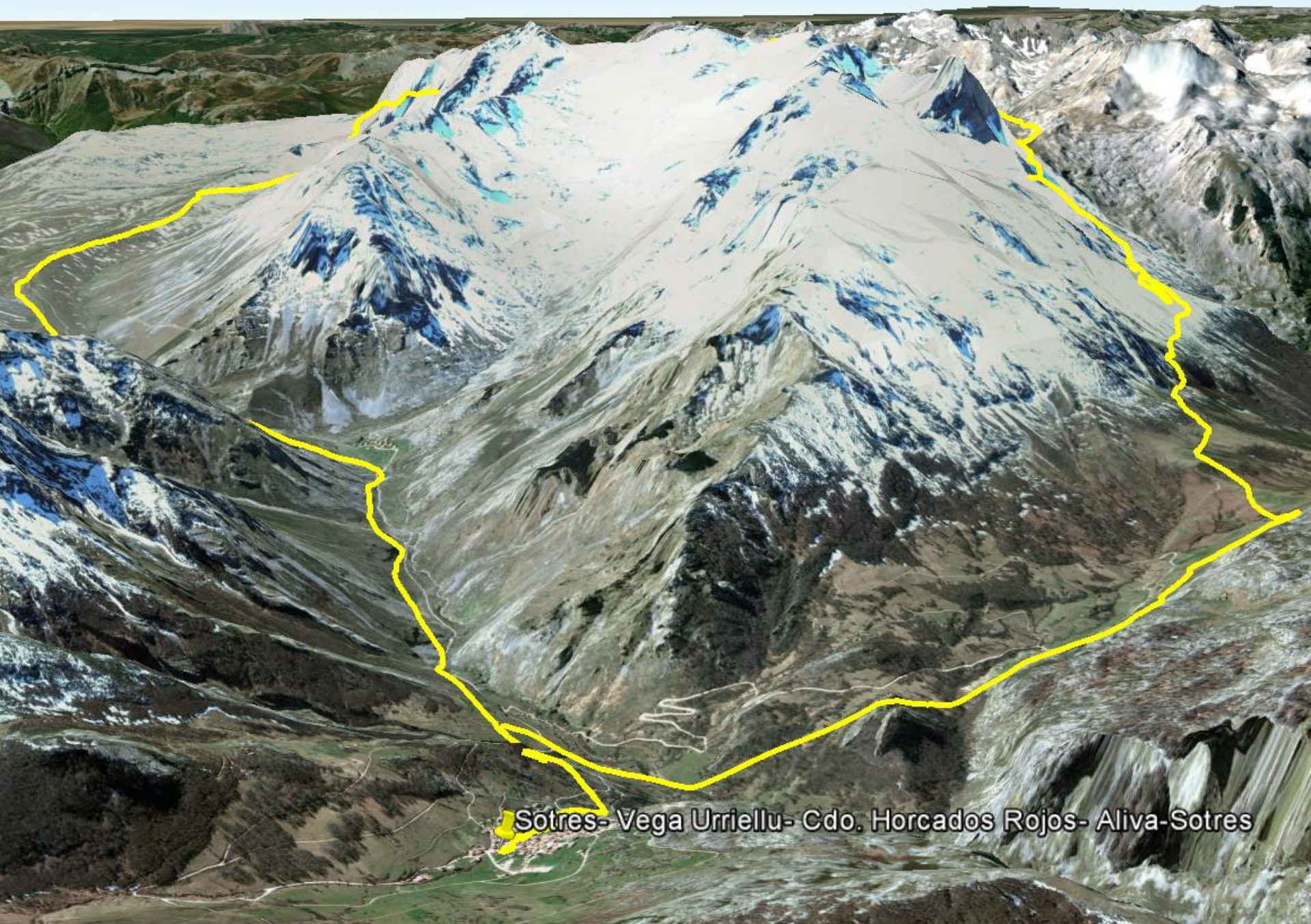
Juan Ramón Pérez Pérez

[jrpp@uniovi.es](mailto:jrpp@uniovi.es)



Una superficie 3D  
¿Cómo calculamos la cota máxima?





Sotres- Vega Urriellu- Cdo. Horcados Rojos- Aliva-Sotres





**Simplificamos** el problema:

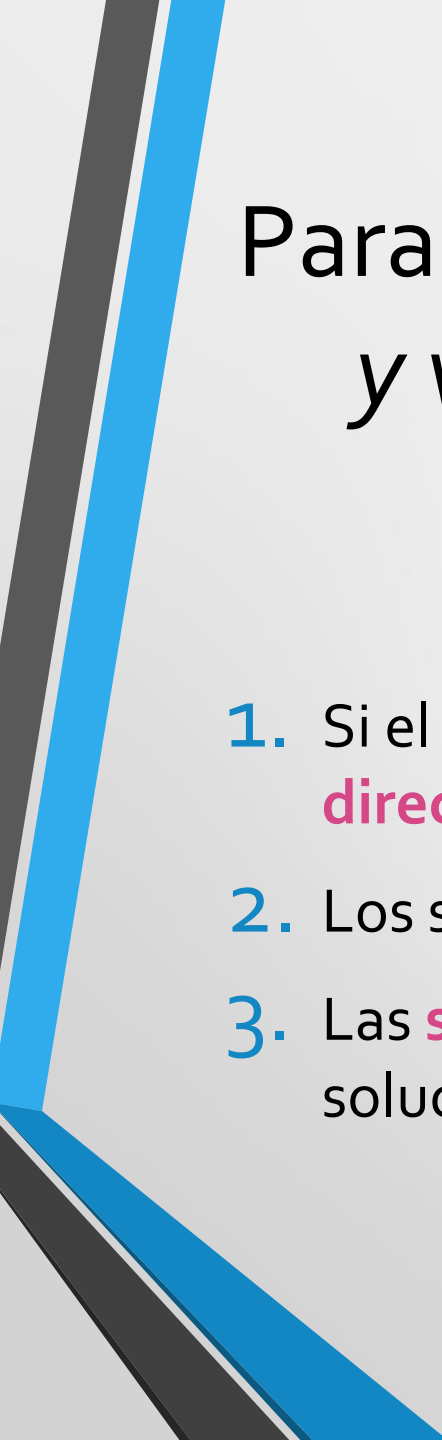
Calcular la cota máxima sobre el perfil de la ruta

# Qué hace divide y vencerás

- **Descomponer** el problema de tamaño  $n$  en subproblemas idénticos al anterior pero de tamaño más pequeño.
- **Resolver** los subproblemas.
- **Componer** las soluciones de los subproblemas, para obtener la solución al problema original.

# La cuestión clave en el planteamiento

- Los problemas resultado de la división son idénticos al original
- Se pueden descomponer aplicando de nuevo la misma técnica. Es decir, de manera **recursiva**.
- Si son suficientemente pequeños, se resuelven aplicando métodos directos.



# Para desarrollar un algoritmo *divide y vencerás* seguiremos 3 pasos

1. Si el **tamaño** del problema es pequeño, resolución **directa**; en caso contrario se **divide en subproblemas**.
2. Los subproblemas se resuelven de manera **recursiva**.
3. Las **soluciones** de los subproblemas se **fusionan** en la solución al problema original.

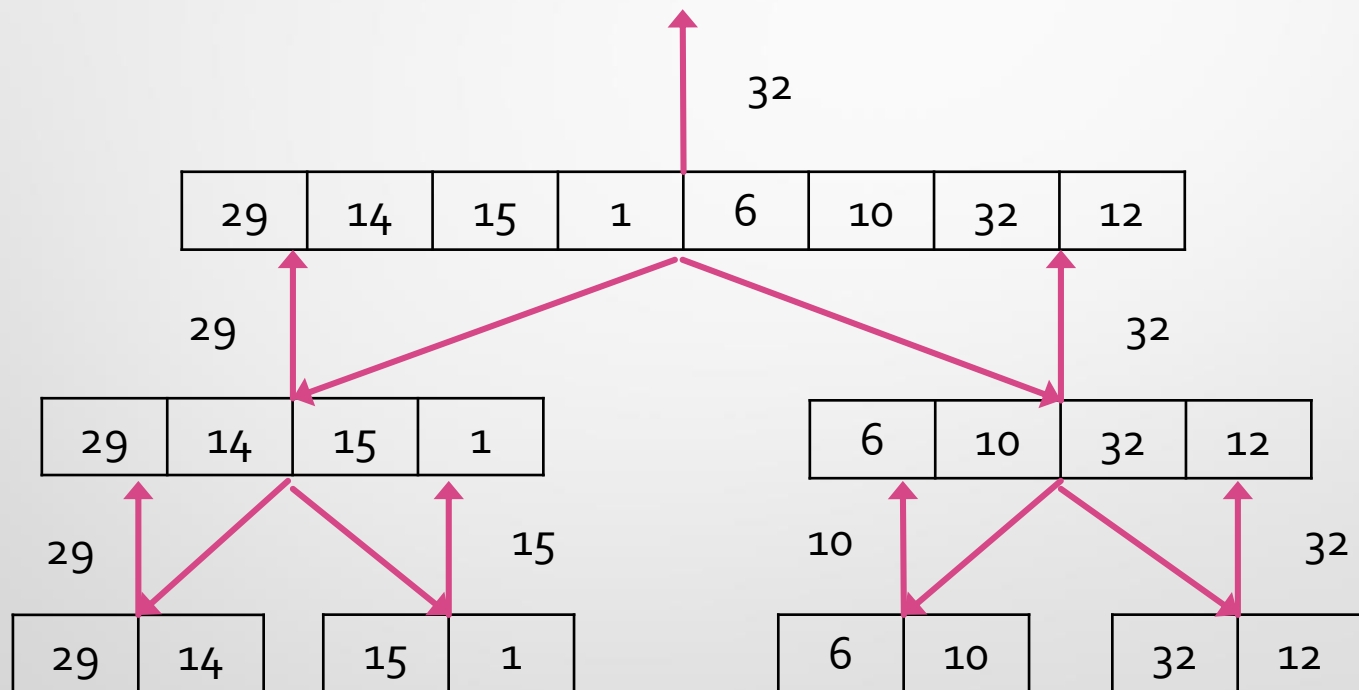
# Cómo aplicamos esto al problema de la cota máxima de una ruta

Nuestra estructura de datos será un array de  $n$  enteros que recoge las cotas de la ruta cada  $x$  metros.

1. Problema *pequeño* es cuando tenemos un array de dos elementos: para obtener el máximo simplemente devolvemos el mayor de los dos.
2. Si no es *pequeño*, dividimos el array en dos partes iguales ( $n/2$ ).
3. Aplicamos recursivamente el mismo planteamiento a cada parte.
4. Tendremos un máximo por cada parte del array, para *fusionarlos* simplemente devolvemos el mayor de los dos.



# Estrategia Divide y vencerás para búsqueda de la cota máxima



# Código del problema de la máxima cota

```
public static int calcularMax(int[] v, int izq, int der)
{
    if ((der-izq)<=1) // tamaño pequeño -> resolución directa
        return (v[izq]>=v[der])?v[izq]:v[der];
    // mayor dos elementos
    else
    {
        int m= (izq+der)/2;
        int maxIzq= calcularMax(v,izq,m-1);
        int maxDer= calcularMax(v,m,der);
        return (maxIzq>=maxDer)?maxIzq:maxDer;
    }
}
```

Código: Maximo.java

# Caso general *Divide y Vencerás*

- El enfoque Divide y vencerás se puede aplicar a otros muchos problemas.
- Requisitos para aplicar esta técnica a un problema:
  - Encontrar un esquema recursivo que vaya reduciendo el problema inicial al caso básico.
  - Tener un algoritmo sencillo capaz de resolver los casos básicos. Eficiente en sus casos pequeños.
  - Disponer de un método para combinar los resultados de los subproblemas.



# Pseudocódigo caso general *Divide y Vencerás*

```
TipoSolucion divideVenceras (int n)
{
    TipoProblema [] subproblemas;
    TipoSolucion [] subsoluciones;

    if (n es suficientemente pequeño)
        return Resolver caso trivial;
    else
    {
        subproblemas= descomponer(n);
        for (int i : subproblemas)
            subsoluciones[i]= divideVenceras(nuevoTam);

        return combinar(subsoluciones);
    }
}
```

# Comentarios al caso general

- Para conseguir un algoritmo eficiente  $\rightarrow$  el número de subproblemas  $a$ , debe de ser pequeño.
- Si  $a=1 \rightarrow$  recibe el nombre de *reducción*.
- El diseño recursivo es más claro; pero podemos abordar lo mismo, sobre todo en la reducción, mediante un bucle iterativo.

# Análisis de la complejidad de un algoritmo *Divide y Vencerás*

$$T(n) = \begin{cases} aT\left(\frac{n}{b}\right) + S(n) + M(n), & n \geq c \\ D(n), & n < c \end{cases}$$

$S(n)$  denota los pasos temporales necesarios para dividir el problema en subproblemas,

$M(n)$  denota los pasos temporales necesarios para fusionar las dos subsoluciones

$D(n)$  son los pasos temporales necesarios para resolver, de forma directa el problema elemental.



# Análisis de la complejidad del problema de la máxima cota

Este problema genera 2 subproblemas de la mitad de tamaño por cada problema original

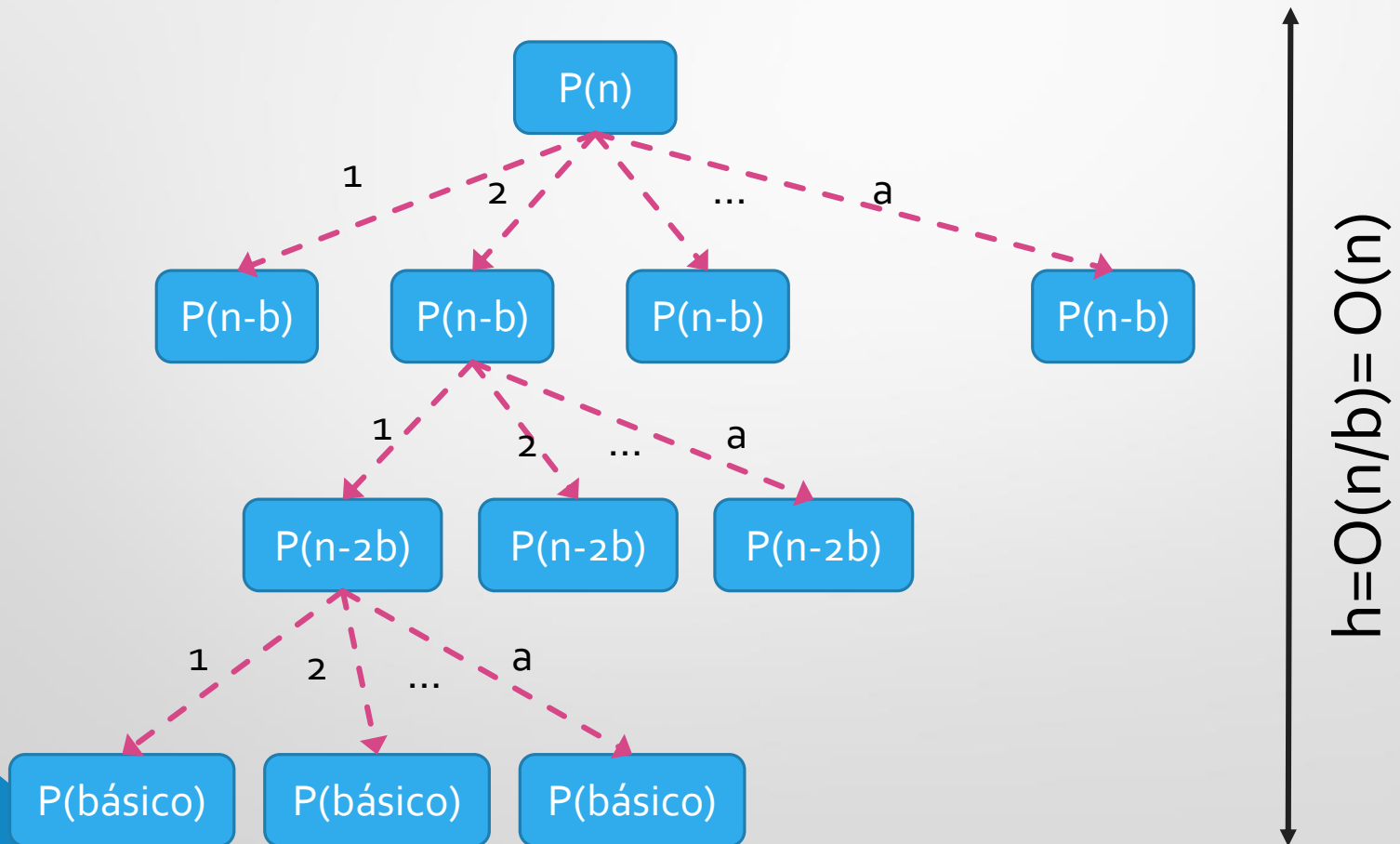
Tanto la división en subproblemas como la fusión no dependen del tamaño del problema

$$T(n) = \begin{cases} 2T\left(\frac{n}{2}\right) + 1, & n \geq 2 \\ 1, & n < 2 \end{cases}$$

Si resolvemos mediante ecuaciones de recurrencias:

$$T(n) = n - 1 \rightarrow O(n)$$

# Divide y vencerás con sustracción



# Divide y vencerás con sustracción

- Parámetros:
  - $a$ , indica el número de subproblemas o llamadas recursivas para resolver un problema no trivial.
  - $b$ , todos los subproblemas tienen el mismo tamaño ( $n - b$ , donde  $b$  es constante).
  - $K$ , excluyendo las llamadas recursivas, el algoritmo es de tipo polinómico  $O(n^K)$ , este parámetro es el exponente máximo de esta complejidad polinómica.



# Análisis de *Divide y vencerás* con sustracción

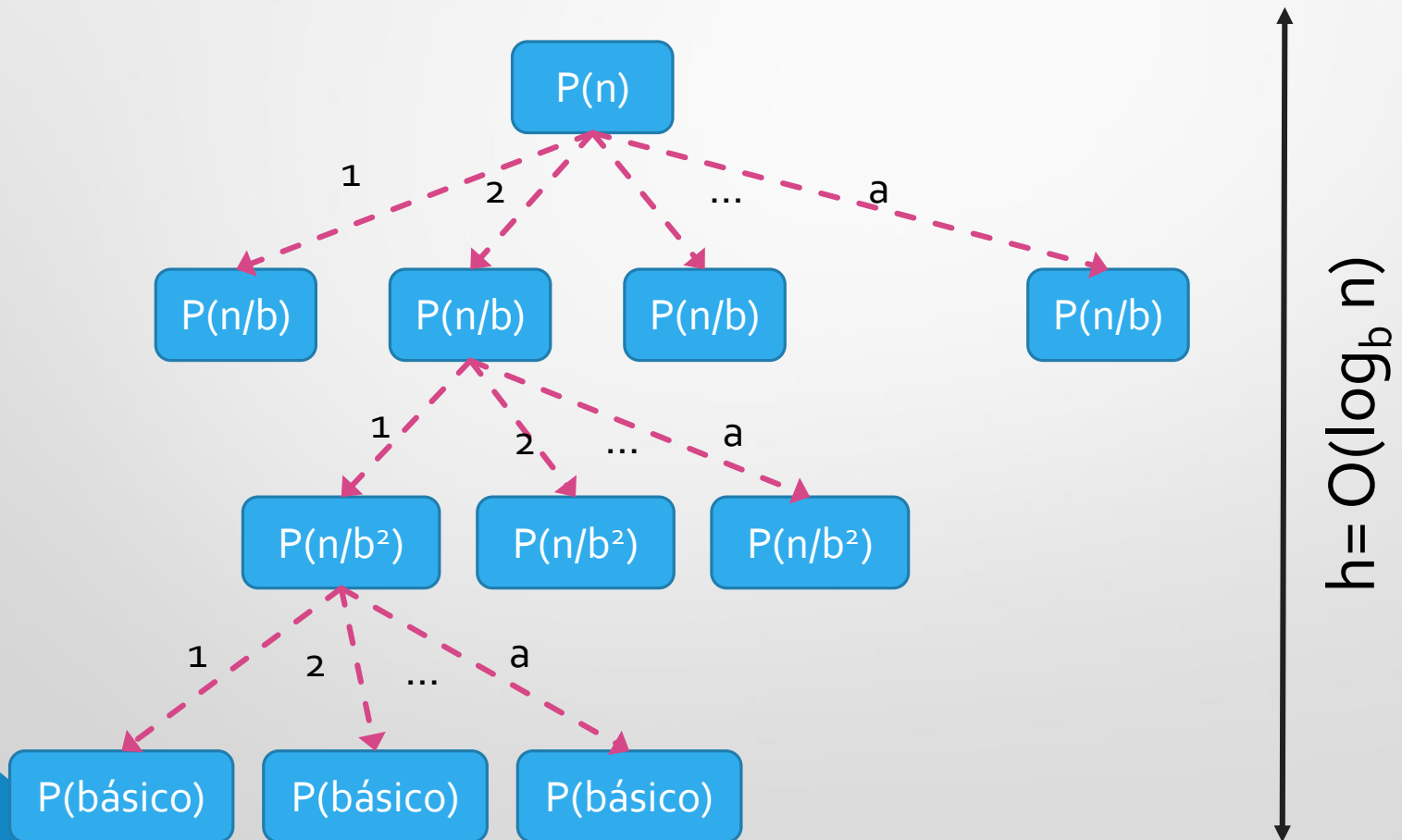
Luego:

$$T(n) = \begin{cases} aT(n - b) + cn^K, & n \geq c \\ cn^K, & n < c \end{cases}$$

Complejidad:

- $O(n^{K+1})$  si  $a=1$
- $O(a^{n/b})$  si  $a>1$

# Divide y vencerás con división



# Divide y vencerás con división

- Parámetros
  - $a$ , indica el número de subproblemas.
  - $b$ , todos los subproblemas tienen el mismo tamaño ( $n / b$ , donde  $b$  es constante).
  - $K$ , excluyendo las llamadas recursivas, el algoritmo es de tipo polinómico  $O(n^K)$ , este parámetro es el exponente máximo de esta complejidad polinómica.



# Análisis de *Divide y vencerás* con división

Luego

$$T(n) = \begin{cases} aT(n/b) + cn^K, & n \geq c \\ cn^K, & n < c \end{cases}$$

Complejidad:

- $O(n^K)$       si  $a < b^K$
- $O(n^K \cdot \log n)$       si  $a = b^K$
- $O(n^{\log_b a})$       si  $a > b^K$



# Ejemplos de *Divide y Vencerás*

# Factorial recursivo

$$n! = \begin{cases} 1 & \text{si } n = 0 \\ n(n-1)! & \text{si } n > 0 \end{cases}$$

Análisis:

- Caso base: tamaño, cómo se resuelve
- Descomposición en subproblemas
- Composición de resultados parciales
- Parámetros
- Complejidad



```
public static int fact (int n)
{
    if (n==0) return 1;
    else return n*fact(n-1);
}
```

Código: Factorial.java

# Búsqueda binaria o dicotómica

- Recibe un vector ordenado v y un entero a buscar x.
- Devuelve el índice del elemento buscado, -1 si no existe.
- ¿Cuántas llamadas recursivas?
- Complejidad:



```
private static int busqrec(int iz,int de)
{
    if (iz>de) return -1; // no existe x
    else {
        int m=(iz+de)/2;
        if (v[m]==x) return m;
        else if (v[m]>x)
            return busqrec(iz,m-1);
        else return busqrec(m+1,de);
    }
}
```

Código: BusqBin.java

# Algoritmos conocidos. Técnica

## *Divide y vencerás* (II)

- Ordenación rápida (Quicksort):
  - Caso mejor:
    - *¿Cuándo se da este caso?*
    - *¿Qué complejidad tiene (parámetros)?*
  - Caso peor: cuando el vector está ya ordenado y el pivote es el primer elemento.
    - *¿Cuándo se da este caso?*
    - *¿Qué complejidad tiene (parámetros)?*

Código: Rapido.java



# Fibonacci recursivo

$$fib(n) = \begin{cases} 0 & \text{si } n = 1 \\ 1 & \text{si } n = 2 \\ f(n-1) + f(n-2) & \text{si } n > 2 \end{cases}$$

Análisis:

- Caso base: tamaño, cómo se resuelve
- Descomposición en subproblemas
- Composición de resultados parciales
- Complejidad

```
public static int fib(int n)
{
    if (n<=1) return n;
    return fib(n-1) + fib(n-2);
}
```

Código: Fibonacci.java

# Análisis de la complejidad de fibonacci

- Parece un esquema por sustracción ( $a=2$ ,  $K=0$ ) pero  $b$  es diferente en los dos subproblemas.
- Si la parte recursiva de la función fuese  $f = f(n-1) + f(n-1)$ 
  - $a=2$ ,  $b=1$ ,  $K=0 \rightarrow O(2^n)$
- Si la parte recursiva de la función fuese  $f = f(n-2) + f(n-2)$ 
  - $a=2$ ,  $b=2$ ,  $K=0 \rightarrow O(2^{n \div 2})$
- Podríamos concluir:
  - $O(2^{n \div 2}) \leq O(\text{Fibonacci}) \leq O(2^n)$
- Complejidad de forma más precisa  $\rightarrow$  hay que plantear y resolver la ecuación de recurrencias.

# Más ejemplos Divide y vencerás

Núm.	Problema	Versiones	Fichero ejemplo
2	Cálculo de Fibonacci	5 algoritmos diferentes	Fibonacci.java
3	Sumatorio de un vector	3 algoritmos	SumaVector.java
4	Búsqueda secuencial	2 algoritmos	BusqSec.java
7	Ordenación por mezcla	Algoritmo recursivo	Mezcla.java
8	Elemento mayoritario x aparece más de $n/2$ veces.	3 algoritmos diferentes	Mayoritario.java
9	Moda de un vector	2 algoritmos diferentes	Moda.java
10	Mediana	2 algoritmos diferentes	
11	Subsecuencia de suma máxima	3 algoritmos diferentes	SumaMaxima.java

# Más ejemplos Divide y vencerás

Problema	Versiónes
Multiplicación de enteros muy grandes	Multiplicar enteros fuera del rango de long
Torres de Hanoi	Algoritmo que intercambia discos entre tres lugares para reordenarlos
Determinante por adjuntos	Cálculo clásico del determinante
Producto de matrices cuadradas	Aplicando un algoritmo recursivo

# Ejercicios propuestos

- Existe otro método de ordenación que utiliza la técnica divide y vencerás, se llama: mergesort
1. Busca la implementación de este método entre los ejemplos (Mezcla.java) y comprende el código.
  2. Realiza un análisis de complejidad del método
  3. Compáralo con el método quicksort: ventajas e inconvenientes.