



Algoritmos y Estructuras de Datos

Cursada 2018

Prof. Alejandra Schiavoni (ales@info.unlp.edu.ar)

Prof. Catalina Mostaccio (catty@lifa.info.unlp.edu.ar)

Prof. Laura Fava (lfava@info.unlp.edu.ar)

Prof. Pablo Iuliano (piuliano@info.unlp.edu.ar)

Agenda

- ❖ Temas de la materia
- ❖ Objetivos de la materia
- ❖ Introducción al Análisis de Algoritmos
- ❖ Repaso de Recursión

Temas del curso

- Árboles
- Cola de Prioridades
- Análisis de Algoritmos
- Grafos

Objetivos de la materia

- Analizar Algoritmos y evaluar su eficiencia
- Estudiar estructuras de datos avanzadas: su implementación y aplicaciones

¿De qué se trata el curso?

Estudiar formas **inteligentes** de organizar la información, de forma tal de obtener Algoritmos **eficientes**.

Listas, Pilas, Colas

Árboles Binarios

Árboles AVL

Árboles Generales

Heaps

Grafos

Estructuras de Datos

Insertar

Borrar

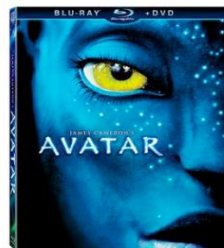
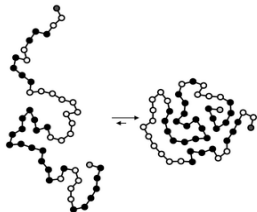
Buscar

Caminos mínimos

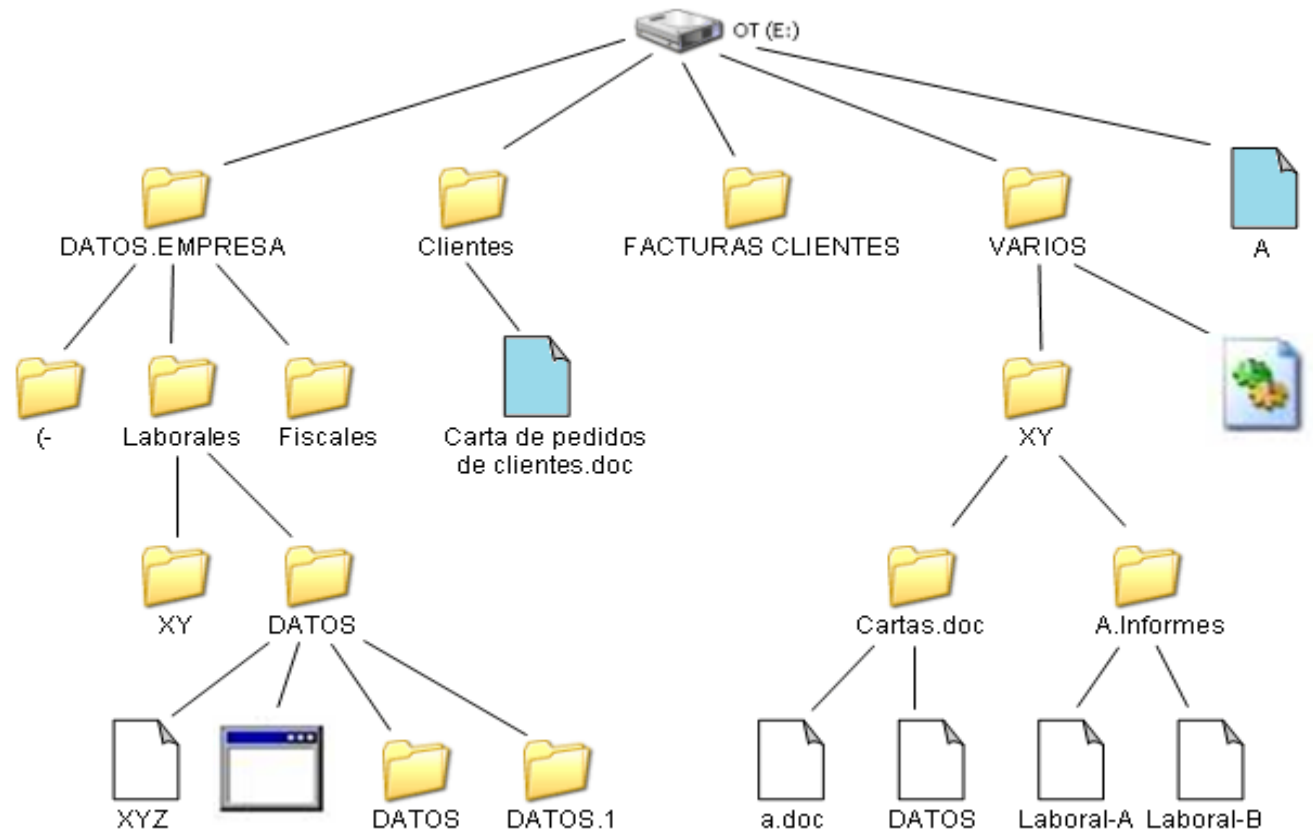
Ordenación

Algoritmos

Las estructuras de datos y sus Algoritmos son....



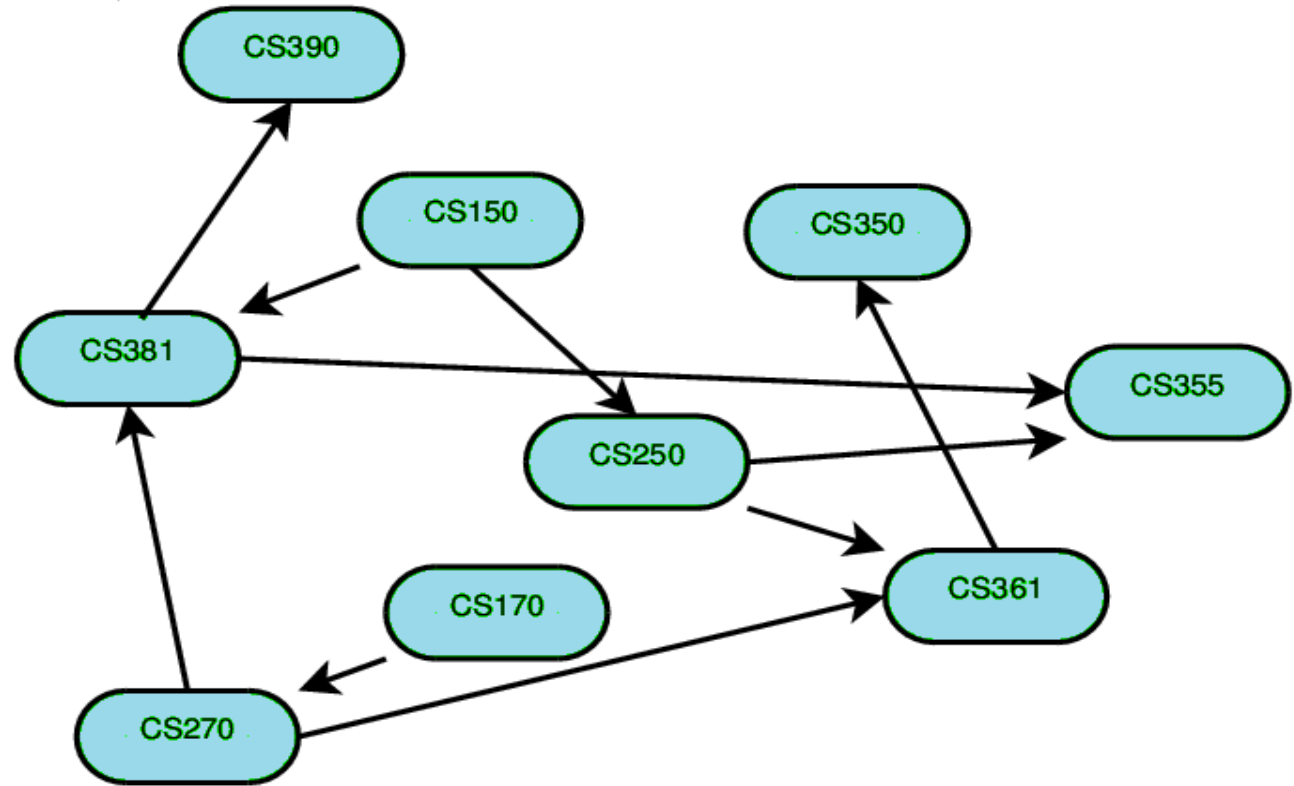
Ejemplo 1: Árbol de carpetas y archivos



Nodos: Carpetas/Archivos

Aristas: representan la relación “contiene”

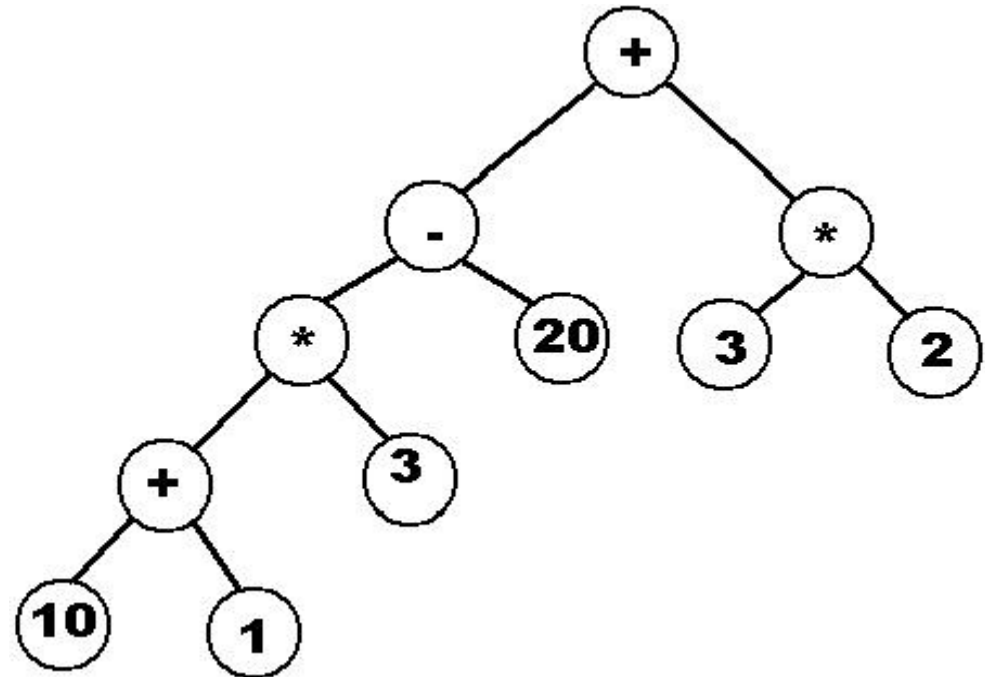
Ejemplo 2: Prerrequisitos de un curso



Nodos: Cursos

Aristas: relación de “prerrequisito”

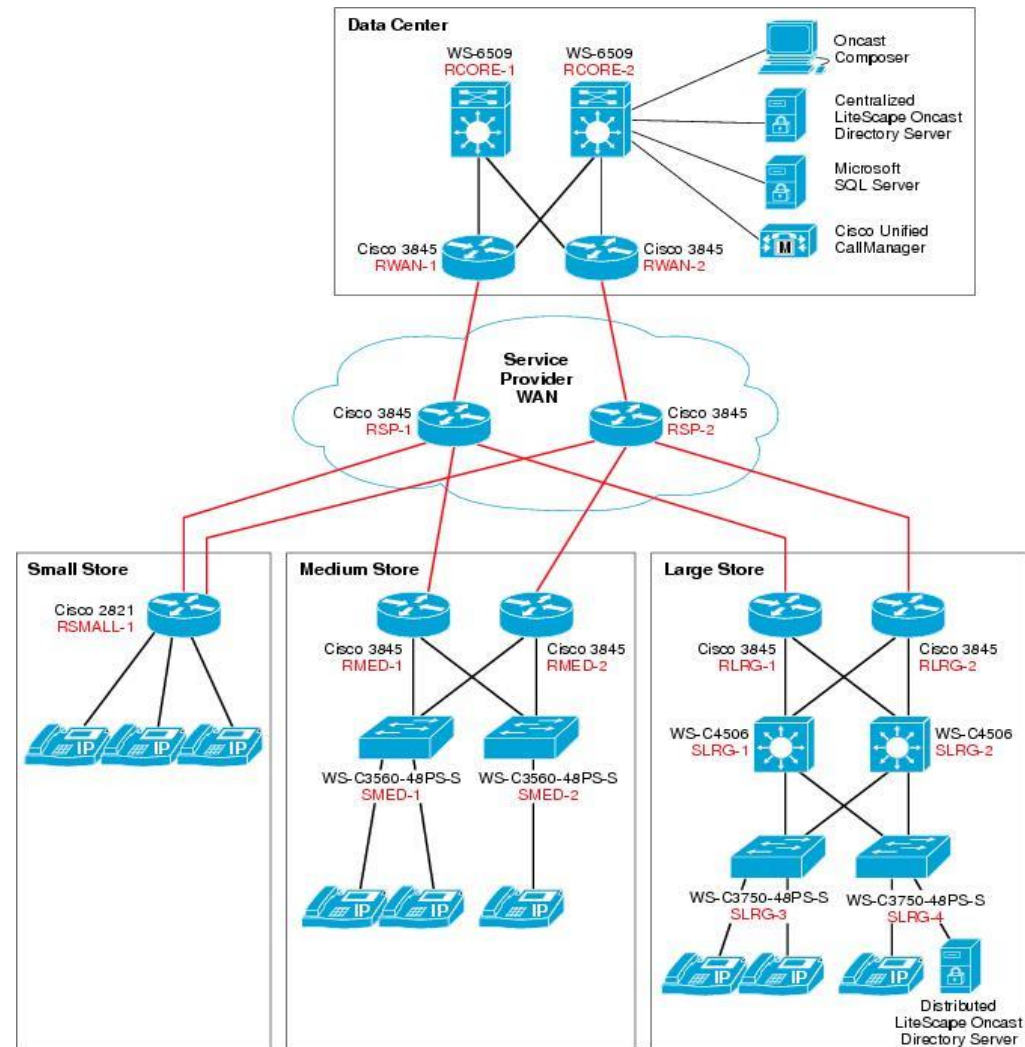
Ejemplo 3: Representación de una expresión en un compilador



Nodos: Operandos/Operadores

Aristas: representan las relaciones entre las operaciones

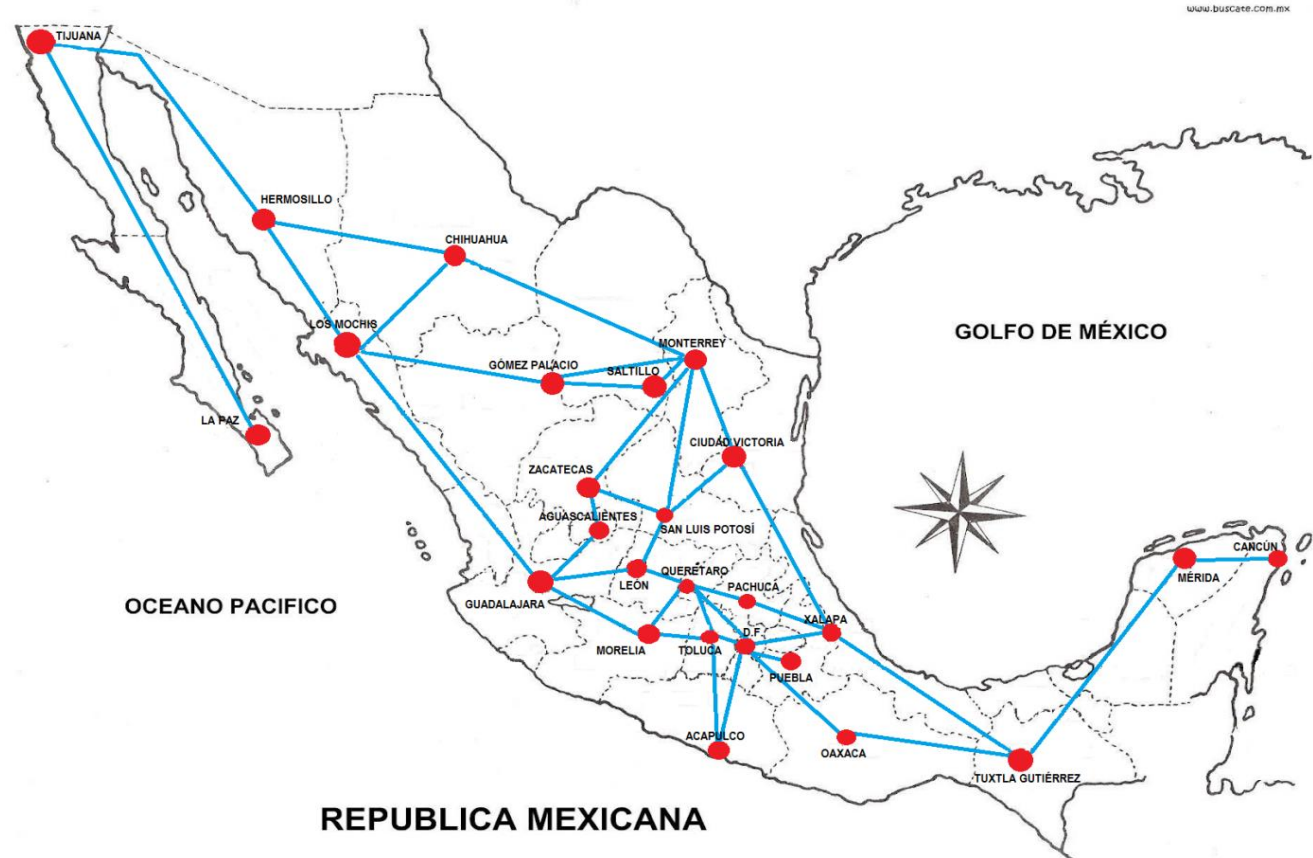
Ejemplo 4: Esquema de una red informática



Nodos: Equipos

Aristas: representan las conexiones

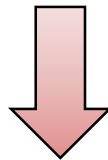
Ejemplo 5: Mapa de ciudades



Nodos: Ciudades
Aristas: Rutas

Estructuras de Datos: Qué, Cómo y Por qué?

- Los programas reciben, procesan y devuelven datos
- Necesidad de organizar los datos de acuerdo al problema que vamos a resolver



Las estructuras de datos son formas de organización de los datos

Estructuras de Datos: Qué, Cómo y Por qué?

- Un programa depende fundamentalmente de la organización de los datos
- cómo se organizan los datos está relacionado con:
 - ➡ Implementación de algunas operaciones: pueden resultar más fácil o más difícil
 - ➡ La velocidad del programa: puede aumentar o disminuir
 - ➡ La memoria usada: puede aumentar o disminuir

Objetivos del curso respecto de las Estructuras de Datos

- Aprender a implementar las estructuras de datos usando abstracción
- Estudiar diferentes representaciones e implementaciones para las estructuras de datos
- Aprender a elegir la “mejor” estructura de datos para cada problema

Algoritmos y su Análisis

- ¿Qué es un Algoritmo?
 - Es una secuencia de pasos que resuelven un problema
 - Es independiente del lenguaje de programación
- Existen varios Algoritmos que resuelven correctamente un problema
- La elección de un Algoritmo particular tiene un enorme impacto en el tiempo y la memoria que utiliza

La elección de un Algoritmo y de la estructura de datos para resolver un problema son interdependientes

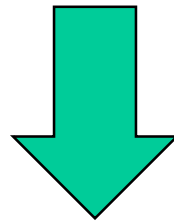
Objetivos del curso respecto del Análisis de los Algoritmos

- Entender los fundamentos matemáticos necesarios para analizar Algoritmos
- Aprender a comparar la eficiencia de diferentes Algoritmos en términos del tiempo de ejecución
- Estudiar algunos Algoritmos estándares para el manejo de las estructuras de datos y aprender a usarlos para resolver nuevos problemas

Problemas y Algoritmos

➤ Problemas:

- Buscar un elemento en un arreglo
- Ordenar una lista de elementos
- Encontrar el camino mínimo entre dos puntos



Encontrar **el Algoritmo** que lo resuelve

Caso: Buscar un elemento en un arreglo

El arreglo puede estar:

- desordenado
- ordenado

Si el arreglo está desordenado  **Búsqueda secuencial**

64	13	93	97	33	6	43	14	51	84	25	53	95
0	1	2	3	4	5	6	7	8	9	10	11	12



Algoritmo: Búsqueda secuencial

```
public static int seqSearch(int[] a, int
    key)
{
    int index = -1;
    for (int i = 0; i < N; i++)
        if (key == a[i])
            index = i;
    return index;
}
```

¿Cuántas comparaciones hace?

Caso: Buscar un elemento en un arreglo

El arreglo puede estar:


- desordenado
- ordenado

Si el arreglo está ordenado 

Búsqueda binaria: Comparo la clave con la entrada del centro

- Si es menor, voy hacia la izquierda
- Si es mayor, voy hacia la derecha
- Si es igual, la encontré

6	13	14	25	33	43	51	53	64	72	84	93	95	96	97
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

 lo  mid  hi

Algoritmo: Búsqueda binaria

```
public static int binarySearch(int[] a, int key)
{
    int lo = 0, hi = a.length-1;
    while (lo <= hi)
    {
        int mid = lo + (hi - lo) / 2;
        if (key < a[mid]) hi = mid - 1;
        else if (key > a[mid]) lo = mid + 1;
        else return mid;
    }
    return -1;
}
```

¿Cuántas comparaciones hace?

¿Cuántas operaciones hace cada Algoritmo?

**Búsqueda
secuencial**

N	Cantidad de operaciones
1000	1000
2000	2000
4000	4000
8000	8000
16000	16000



Hace N operaciones

**Búsqueda
binaria**

N	Cantidad de operaciones
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14



Hace $\log(N)$ operaciones

¿Cómo medir el tiempo?



✓ Manual

Tomando el tiempo que tarda

✓ Automática

Usando alguna instrucción del lenguaje para medir tiempo

Análisis empírico

Correr el programa para varios tamaños de la entrada y medir el tiempo. Suponemos que cada comparación tarda 1 seg.

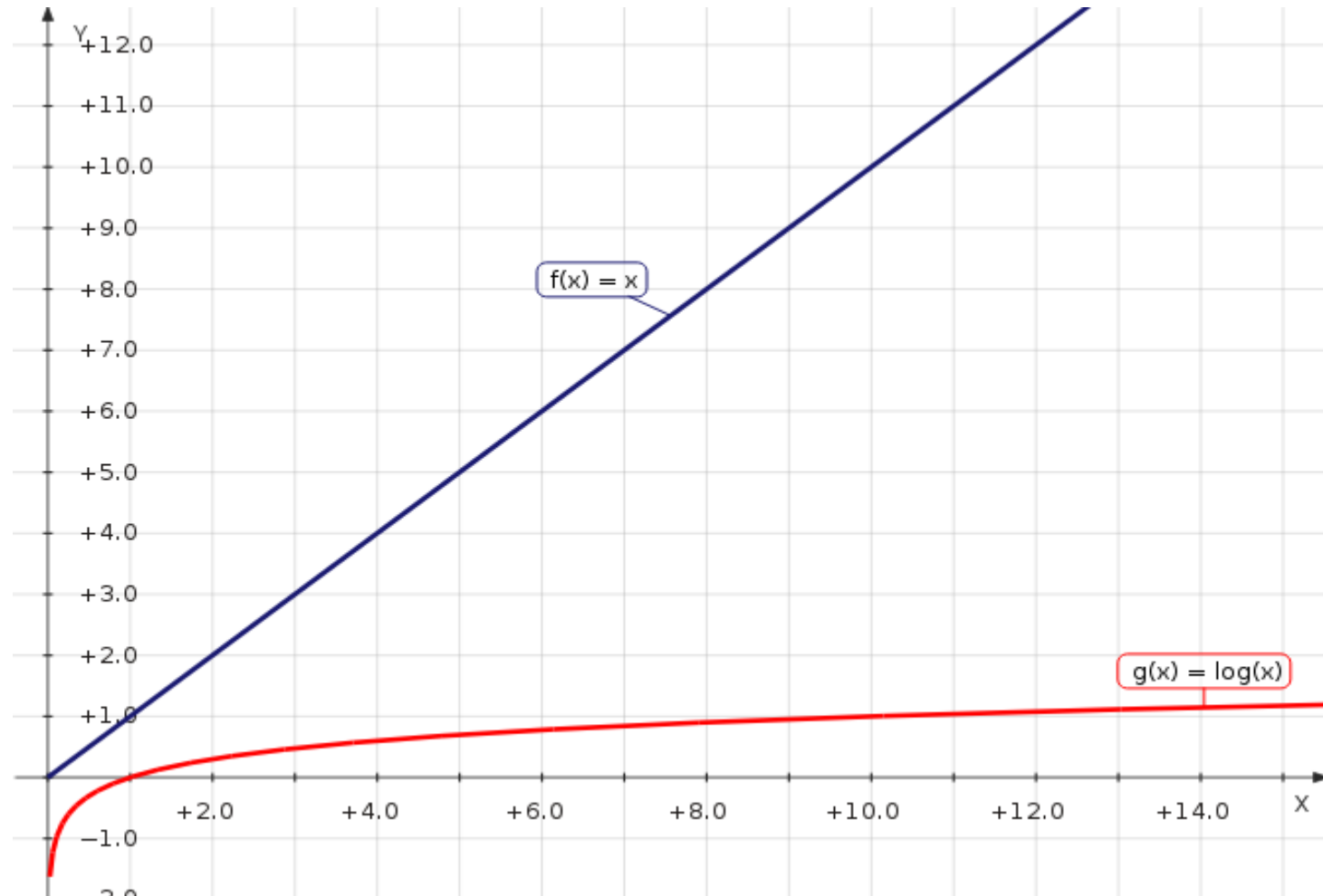
**Búsqueda
secuencial**

N	Tiempo (seg)
1000	1000
2000	2000
4000	4000 ~ 1 hs.
8000	8000 ~ 2 hs
16000	16000 ~ 4 hs.

**Búsqueda
binaria**

N	Tiempo (seg)
1000	~10
2000	~11
4000	~12
8000	~13
16000	~14

Análisis de Algoritmos



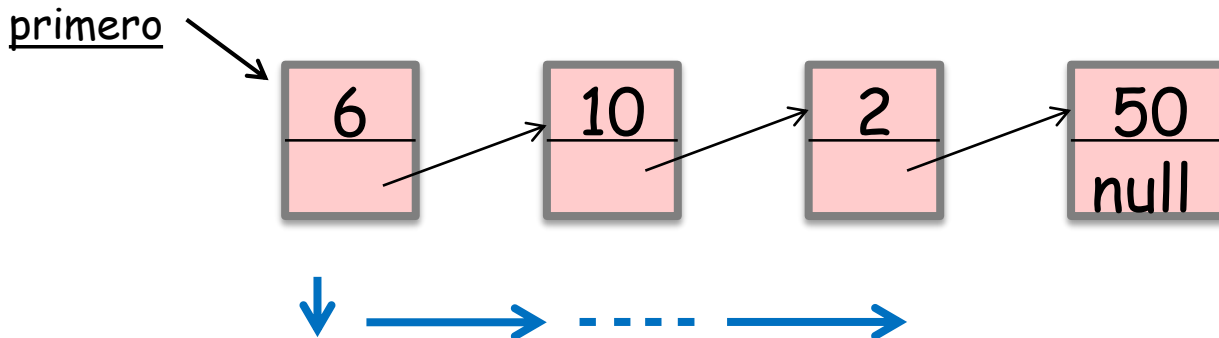
Caso: Buscar un elemento en una lista dinámica

Si los elementos están almacenados en una lista dinámica

La lista puede estar:

- desordenada
- ordenada

¿Cómo sería el Algoritmo de búsqueda?



¿Cuántas comparaciones hace?



Hace N comparaciones

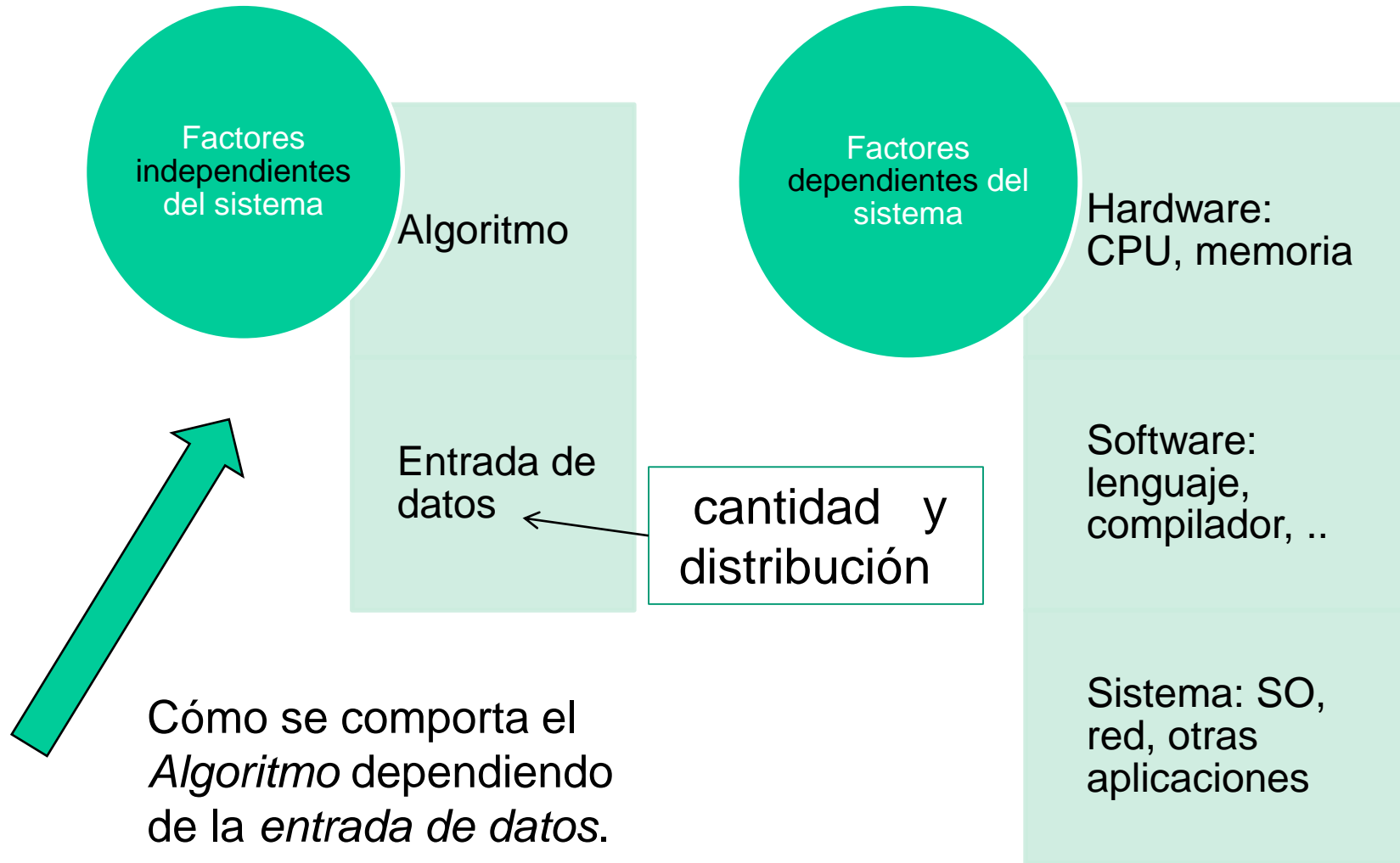
Análisis de Algoritmos

***Marco para predecir la performance y
comparar Algoritmos***

Desafío:

Escribir programas que puedan resolver en forma eficiente problemas con una gran entrada de datos

Análisis de Algoritmos



Análisis de Algoritmos

Existe un modelo matemático para medir el tiempo

Tiempo total:

*Suma del **costo** x **frecuencia** de todas las operaciones*

- Es necesario analizar el Algoritmo para determinar el conjunto de operaciones
 - **Costo** depende de la máquina, del compilador, del lenguaje
 - **Frecuencia** depende del Algoritmo y de la entrada

Repaso de Recursión

- Se dice que un objeto es recursivo cuando forma parte de sí mismo, es decir puede definirse en términos de sí mismo.
- En programación, la recursividad es la propiedad que tienen los Algoritmos de llamarse a sí mismos para resolver un problema.

Repaso de Recursión

➤ Ejemplos de definiciones recursivas:

- Factorial de un número

$$0! = 1$$

$$\text{Si } n > 0, n! = n * (n-1)!$$

- Potencia de un número

$$x^0 = 1$$

$$\text{Si } y > 0, x^y = x * x^{y-1}$$

- Estructuras de datos

Árboles

Repaso de Recursión

➤ Ejemplos de soluciones recursivas:

- Buscar un elemento en un arreglo
- Ordenar un arreglo de elementos
- Recorrer un árbol

Repaso de Recursión

➤ Soluciones recursivas:

- División sucesiva del problema original en problemas más pequeños del mismo tipo
- Se van resolviendo estos problemas más sencillos
- Con las soluciones de éstos se construyen las soluciones de los problemas más complejos

Repaso de Recursión

➤ Ejemplo:

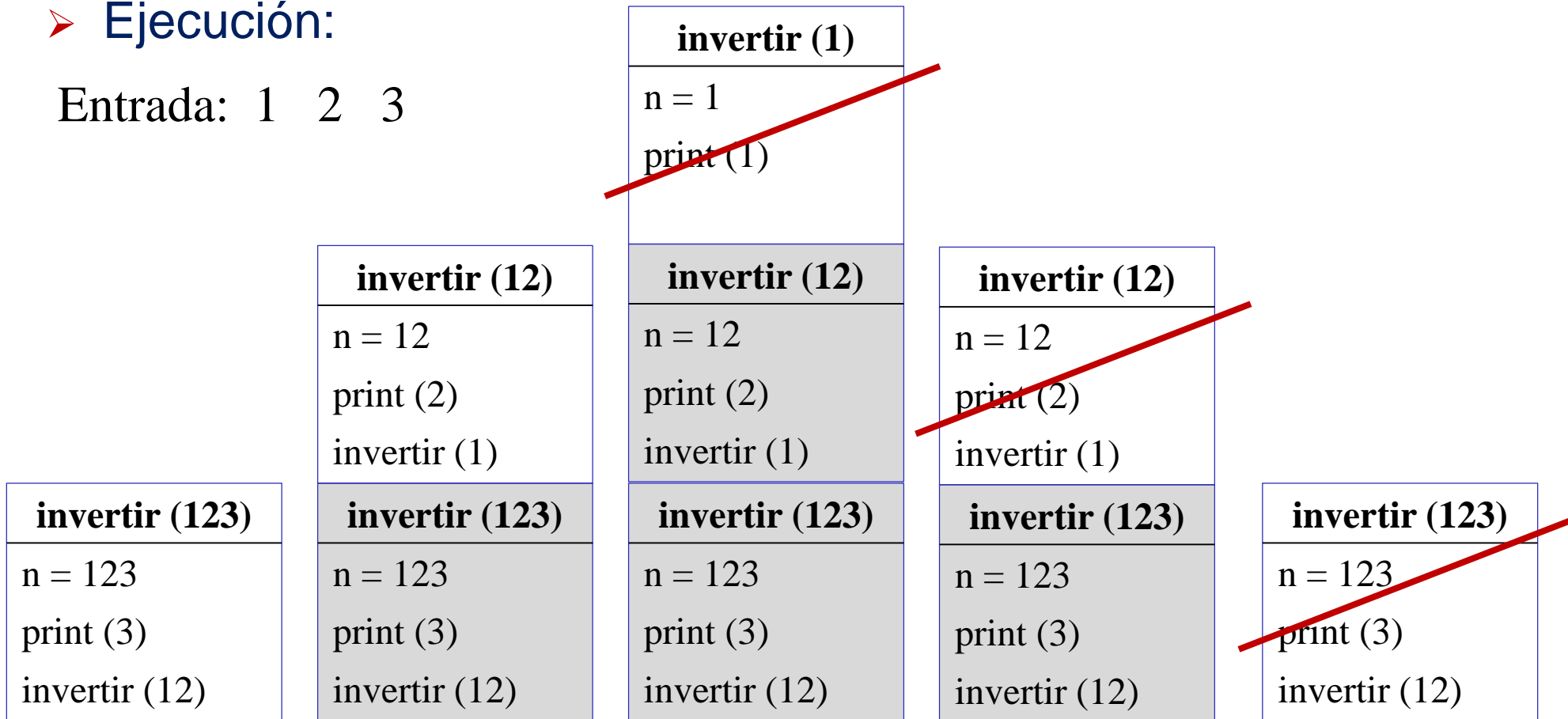
Programar un algoritmo recursivo que permita invertir un número

```
int invertir (int n)
{
    if (n < 10)      //caso base
        System.out.print(n);
    else
        System.out.print(n mod 10); // el resto de la división entera
        invertir (n div 10);
}
```

Repaso de Recursión

➤ Ejecución:

Entrada: 1 2 3



Salida: 3 2 1

Repaso de Recursión

➤ Ejemplo 2: Algoritmo de ordenación *MergeSort*

La estrategia del algoritmo consiste en dividir el vector en 2 partes (sub-vectores), ordenarlos y luego hacer un Merge de estos sub-vectores ya ordenados. Cada uno de esos sub-vectores se ordenan aplicando la misma estrategia, hasta tanto el vector contenga sólo un dato y en ese caso se lo devuelve (el sub-vector está ordenado).

Características *recursivas* del algoritmo

- ✓ Se resuelven 2 sub-problemas más pequeños
- ✓ Se combinan los resultados de cada solución
- ✓ Se cuenta con un caso base.

Repaso de Recursión

Ejemplo 2: Estrategia

Ordenar un arreglo con n elementos

mergesort(a, 0, 6)

Ordenar la 1er mitad del arreglo
n/2 elementos

mergesort(a, 0, 3)

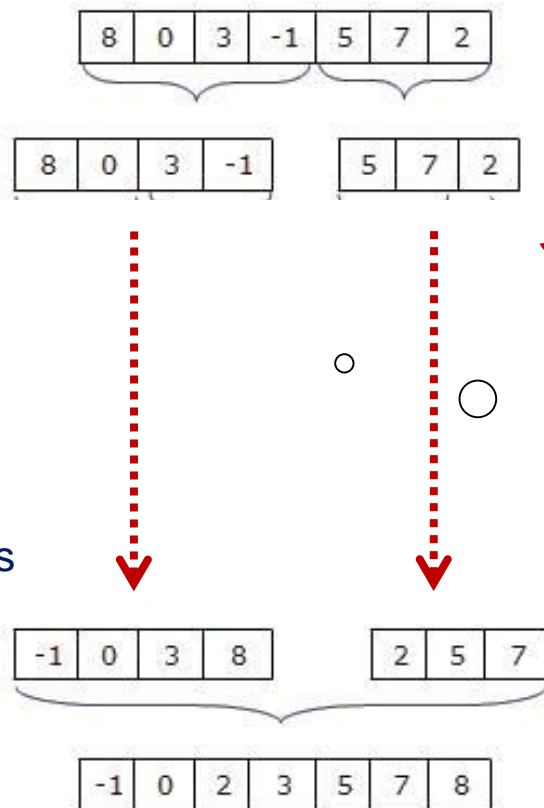
Ordenar la 2da mitad del arreglo
n/2 elementos

mergesort(a, 4, 6)

Mezclar ambas mitades ya ordenadas

merge(a, 0, 3, 6)

En cada paso
intermedio se
aplica la misma
estrategia



Repaso de Recursión

Ejemplo 2:

```
public static void mergesort (int a[],int izq, int der) {  
(a)     if ( izq<der ) {  
(b)         int m = (izq+der)/2;  
(c)         mergesort (a,izq, m);  
(d)         mergesort (a,m+1, der);  
(e)         merge (a, izq, m, der);  
           }  
}
```

Repaso de Recursión

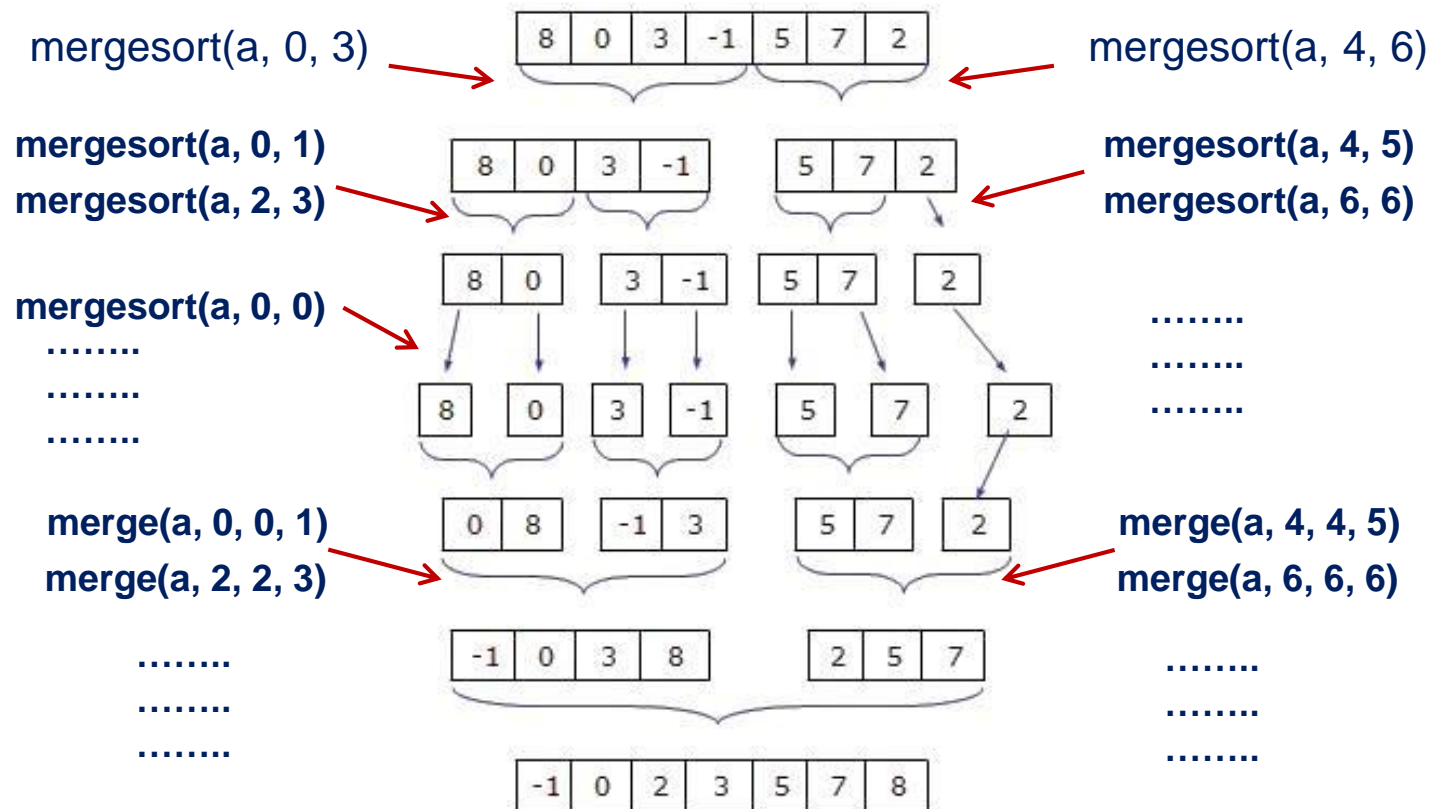
Ejemplo 2 (cont.) :

```
public static void merge ( int a[], int izq, int m, int der) {  
    int i, j, k;  
    int [] b = new int [a.length];    //array auxiliar  
    for ( i=izq; i<=der; i++ ) {      //copia ambas mitades en el array auxiliar  
        b[i]=a[i]; }  
    i=izq; j=m+1; k=izq;  
    while (i<=m && j<=der) { //copia el siguiente elemento más grande  
        if (b[i]<=b[j])  
            a[k++]=b[i++];  
        else  
            a[k++]=b[j++];  
    }  
    while (i<=m) //copia los elementos que quedan de la  
        a[k++]=b[i++]; //primera mitad (si los hay)  
}
```

Repaso de Recursión

Ejemplo 2: Ejecución

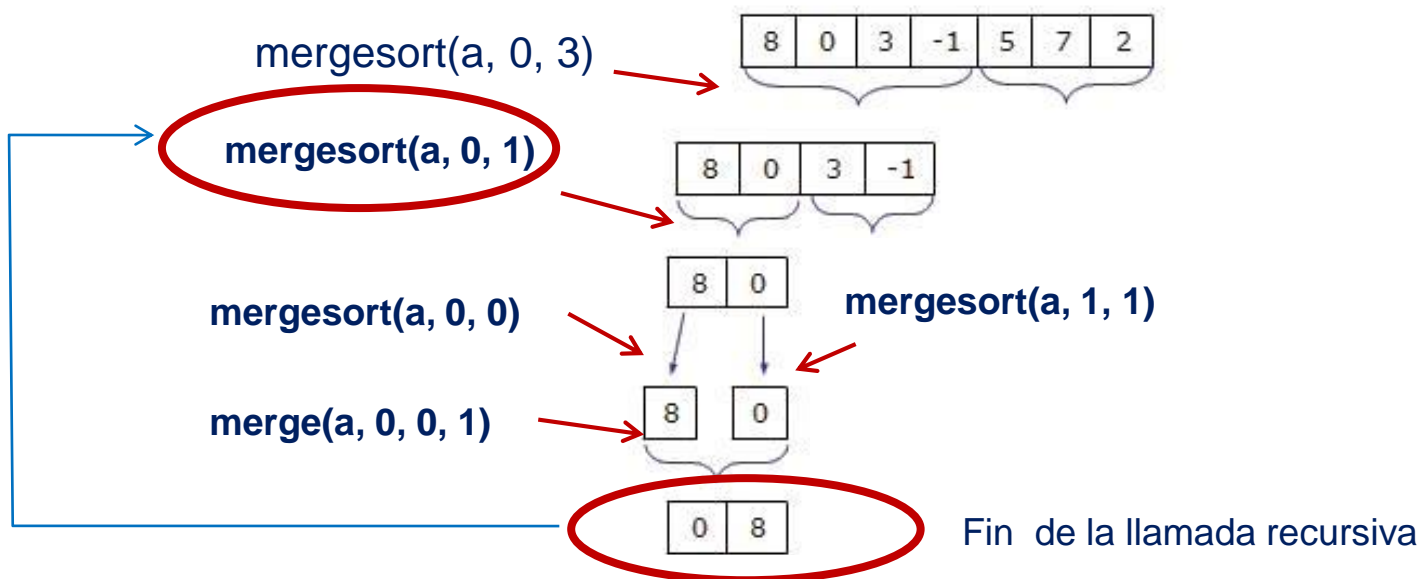
mergesort(a, 0, 6)



Repaso de Recursión

Ejemplo 2: Ejecución

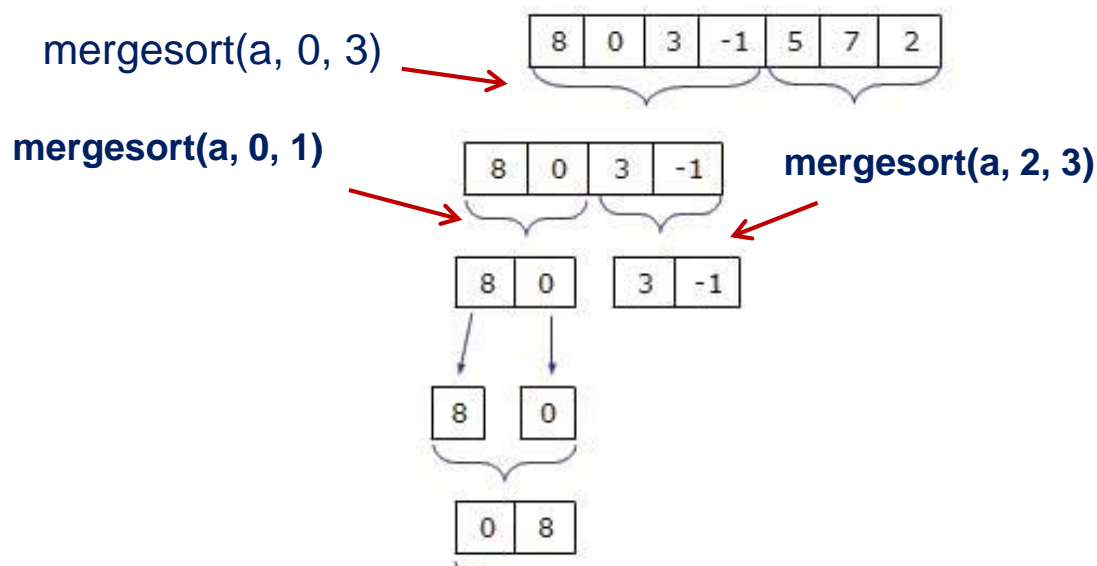
mergesort(a, 0, 6)



Repaso de Recursión

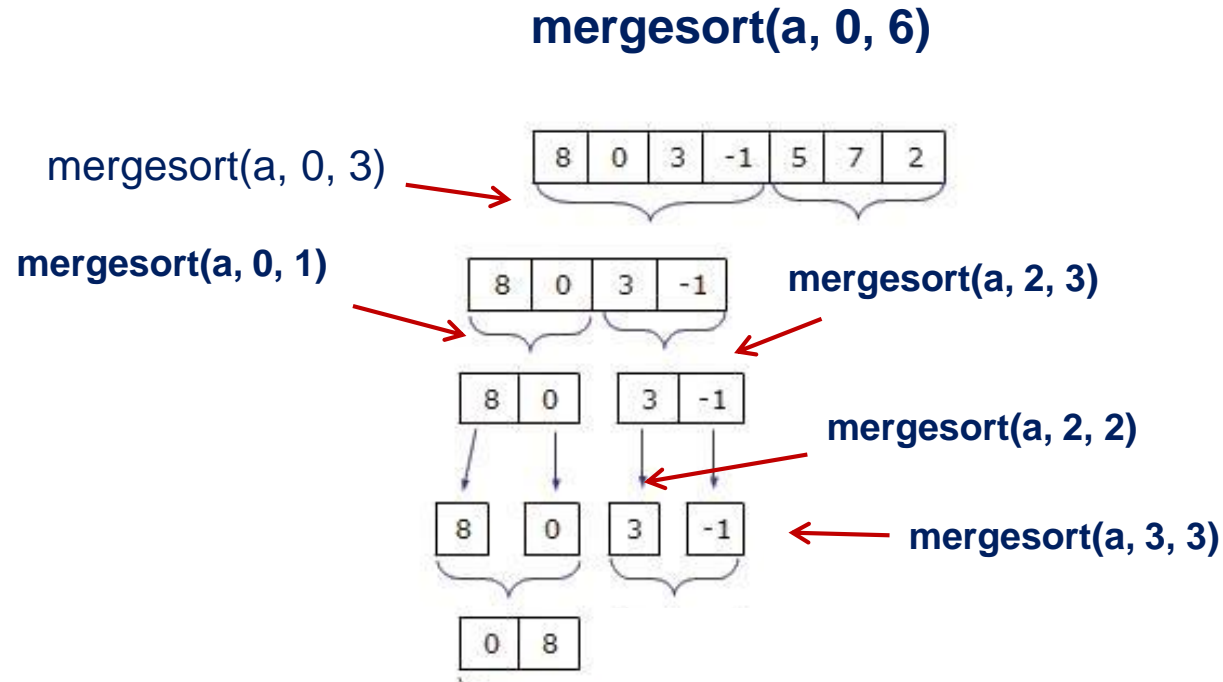
Ejemplo 2: Ejecución

mergesort(a, 0, 6)



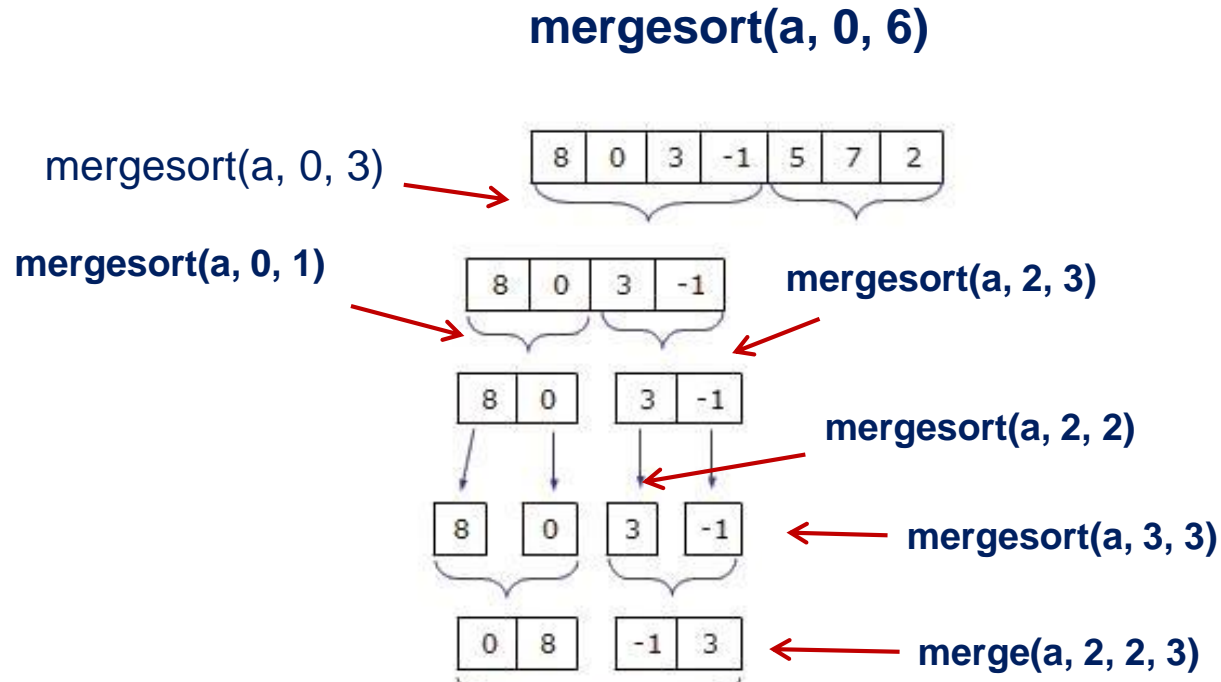
Repaso de Recursión

Ejemplo 2: Ejecución



Repaso de Recursión

Ejemplo 2: Ejecución

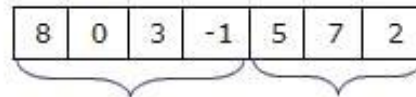


Repaso de Recursión

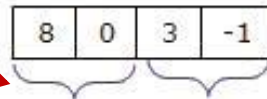
Ejemplo 2: Ejecución

mergesort(a, 0, 6)

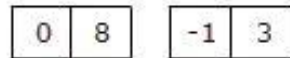
mergesort(a, 0, 3)



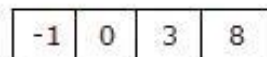
mergesort(a, 0, 1)



mergesort(a, 2, 3)



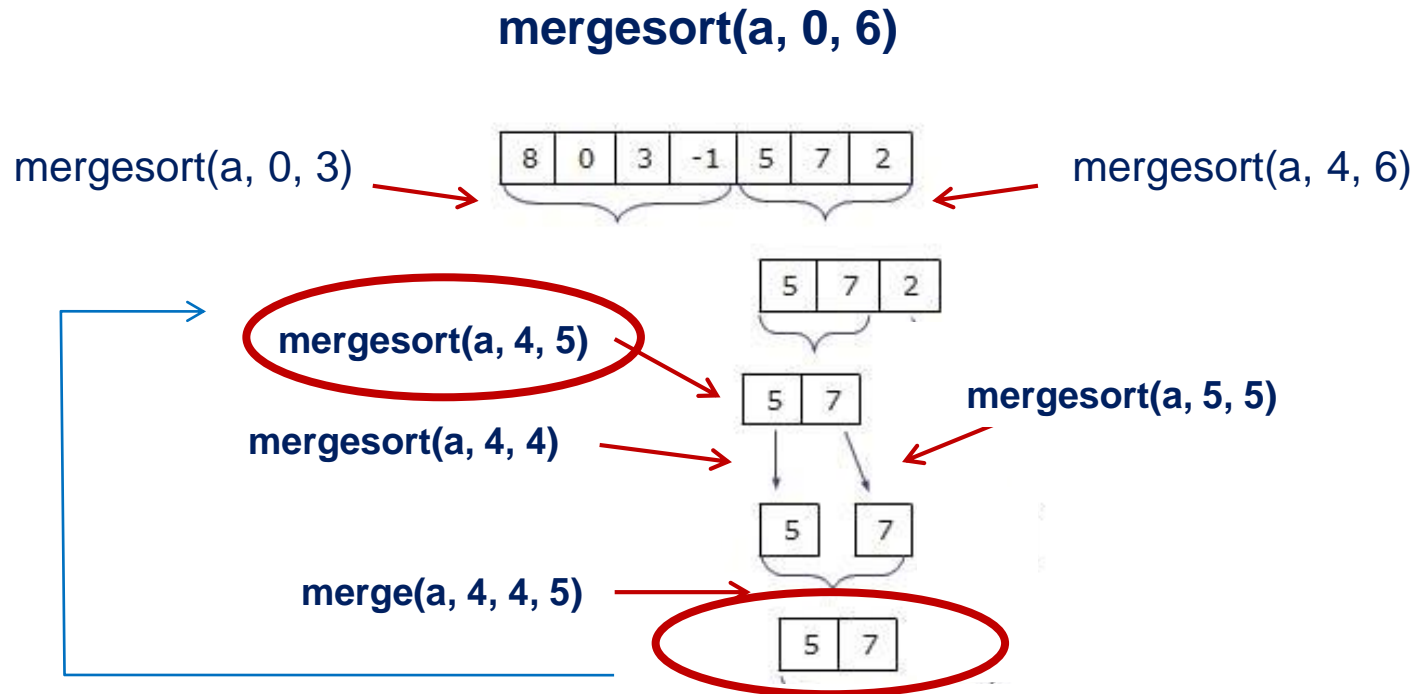
merge(a, 0, 1, 3)



Fin de la llamada recursiva

Repaso de Recursión

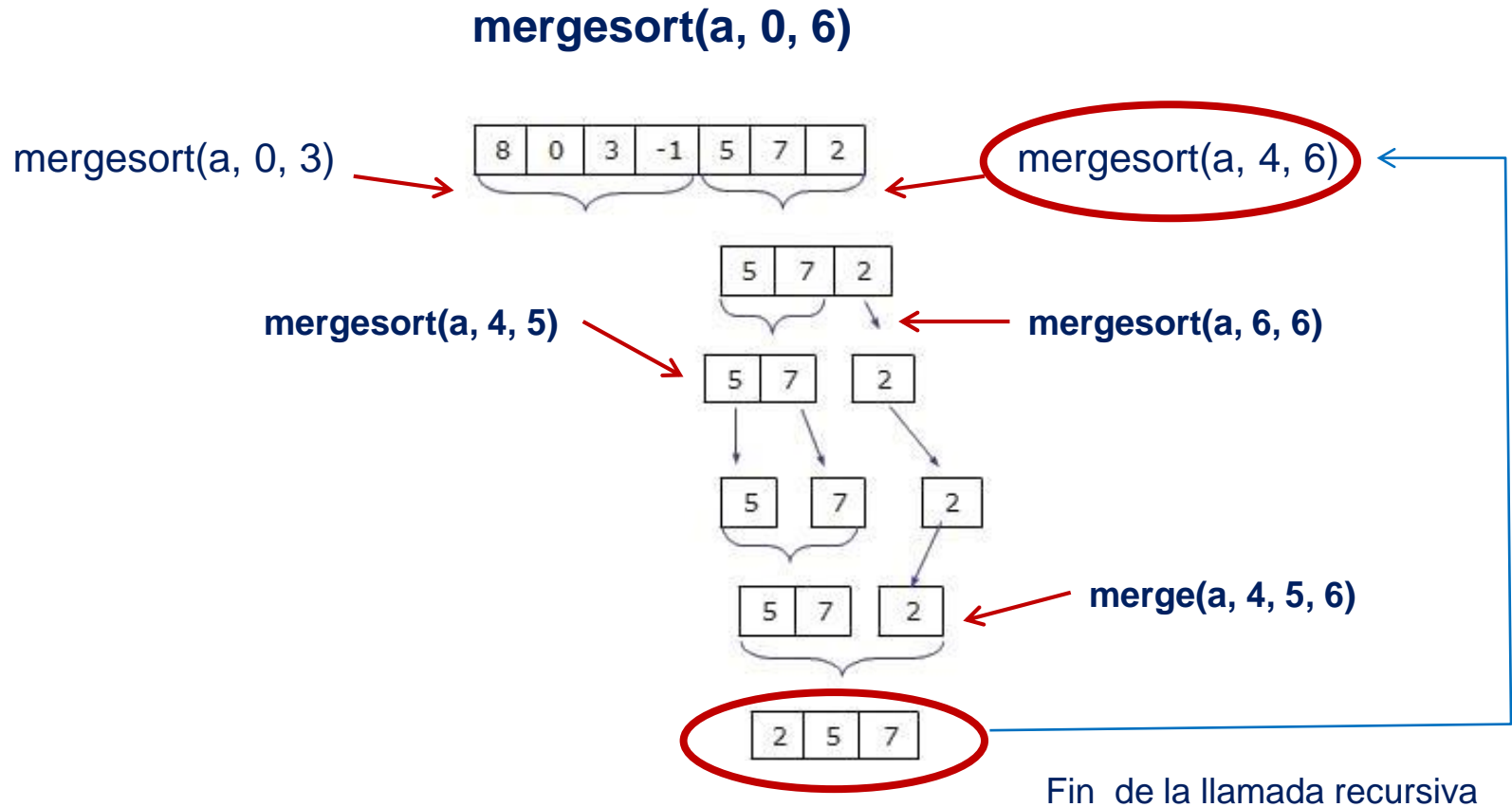
Ejemplo 2: Ejecución



Fin de la llamada recursiva

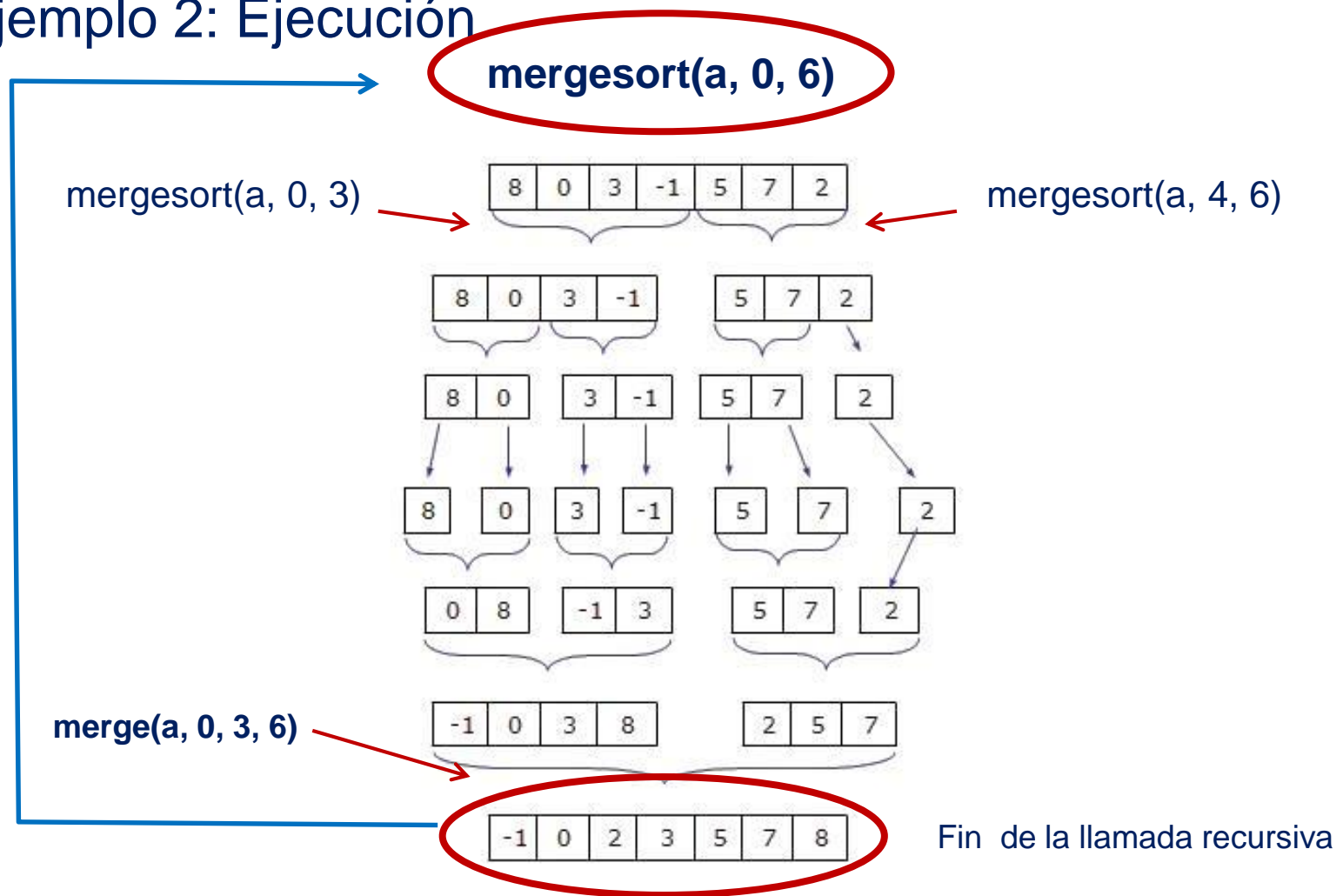
Repaso de Recursión

Ejemplo 2: Ejecución



Repaso de Recursión

Ejemplo 2: Ejecución



FIN