

Sesión 2 (3ª Semana)

David Morán (ddavidmorang@gmail.com)
Juan Quintana (juandavid.quintana@urjc.es)
Sergio Pérez (sergioperezp1995@gmail.com)

Contenidos

- Complejidad en tiempo y espacio
- Estructuras de Datos Básicas
 - Arrays, Matrices
 - Listas, Pilas, Colas
 - Árboles Generales y Binarios

Contenidos

- Algoritmos de Ordenamiento y Búsqueda
 - Bubble Sort, Selection Sort, Merge Sort, Quick Sort
 - Colas de Prioridad
 - Búsqueda Binaria, Búsqueda Ternaria
 - Exponenciación Logarítmica

Complejidad

- Motivación:
 - No se admite cualquier solución
- Restricciones:
 - Tiempo de ejecución
 - Memoria utilizada

Complejidad

- TLE (Time limit exceeded)
 - rebasó el tiempo asignado
 - condición muy restrictiva
- MLE (Memory limit exceeded)
 - agotó la memoria asignada
 - el límite no suele ser muy ajustado (depende de la competición)

Complejidad

- Según la competición
 - Especificaciones del juez (CPU/Cores, RAM, ...)
 - Sesión de pruebas antes del concurso
- ¿De qué me sirve esto?
 - Ejemplo: SWERC'18 era imposible conseguir TLE adrede

Complejidad

- En muchas competiciones formales te dan las especificaciones de la computadora donde se llevarán a cabo todas las evaluaciones ¿De qué me sirve esto?

Complejidad

- Observar un problema y pensar, ¿Qué complejidad se necesita?
 - ¿cuánto tarda en ejecutarse?
 - ¿cuánta memoria consume?
- Formular la complejidad exacta en tiempo y memoria es “laborioso”...

Complejidad

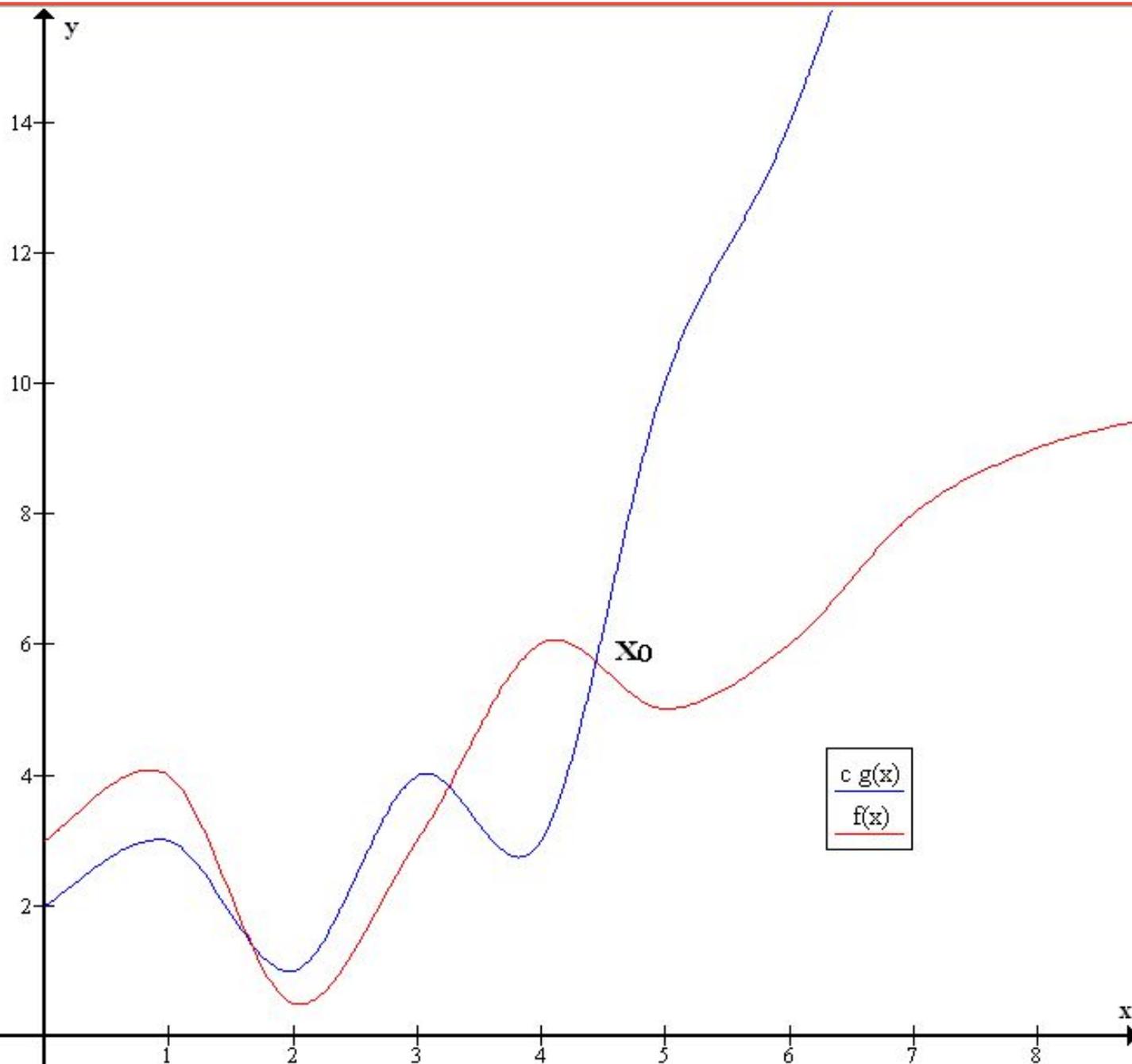
- NO necesitamos realizar un cálculo exacto de la complejidad (ni tendremos tiempo)
- Un cálculo intuitivo “a ojo” nos sirve para realizar una estimación

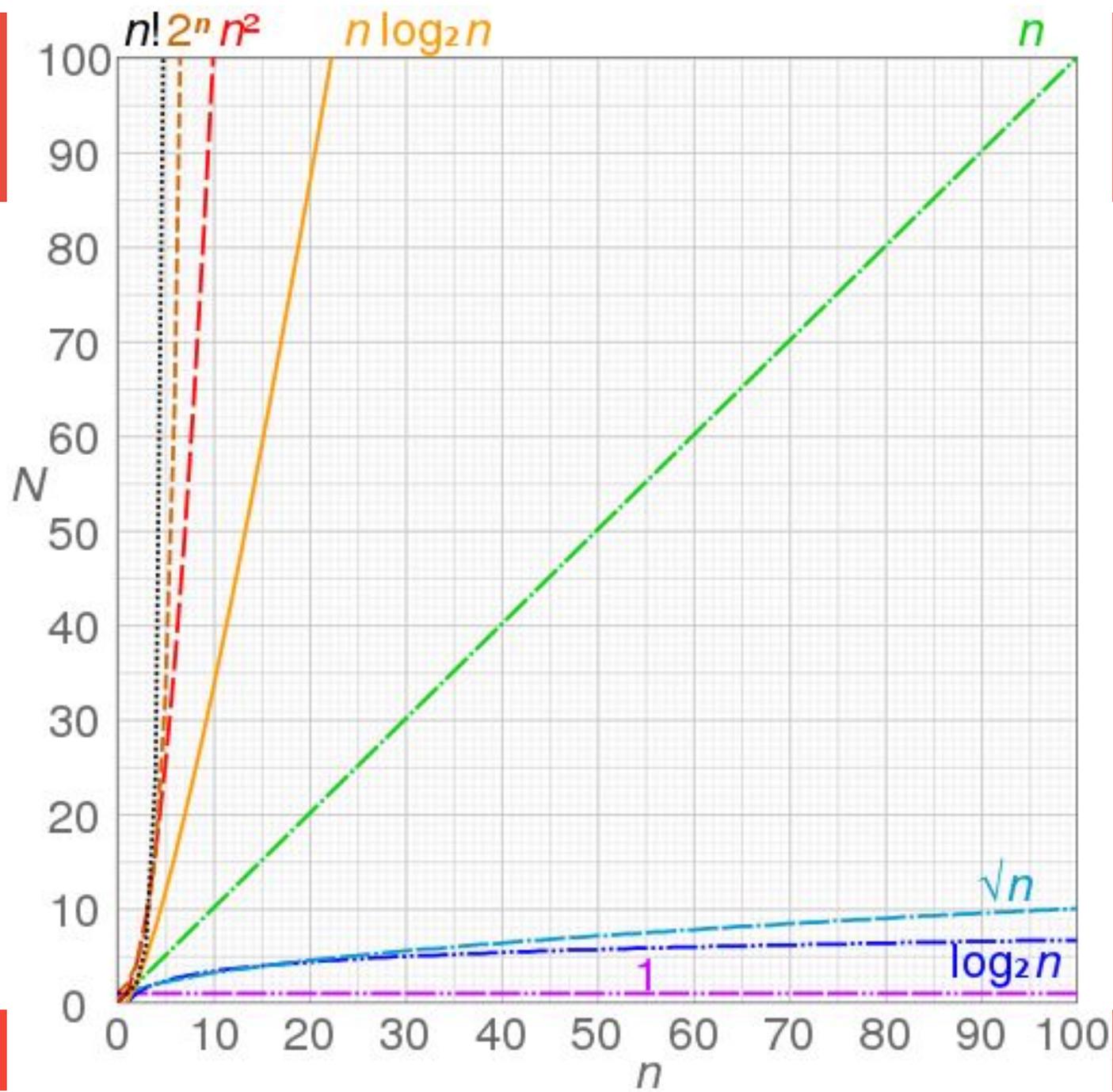
Complejidad

- Estimar la complejidad en tiempo:
 - Contar bucles anidados
 - Conocer la complejidad de:
 - funciones utilizadas
 - estructuras de datos y sus operaciones
- ¡Siempre pensar en el peor caso!

Complejidad

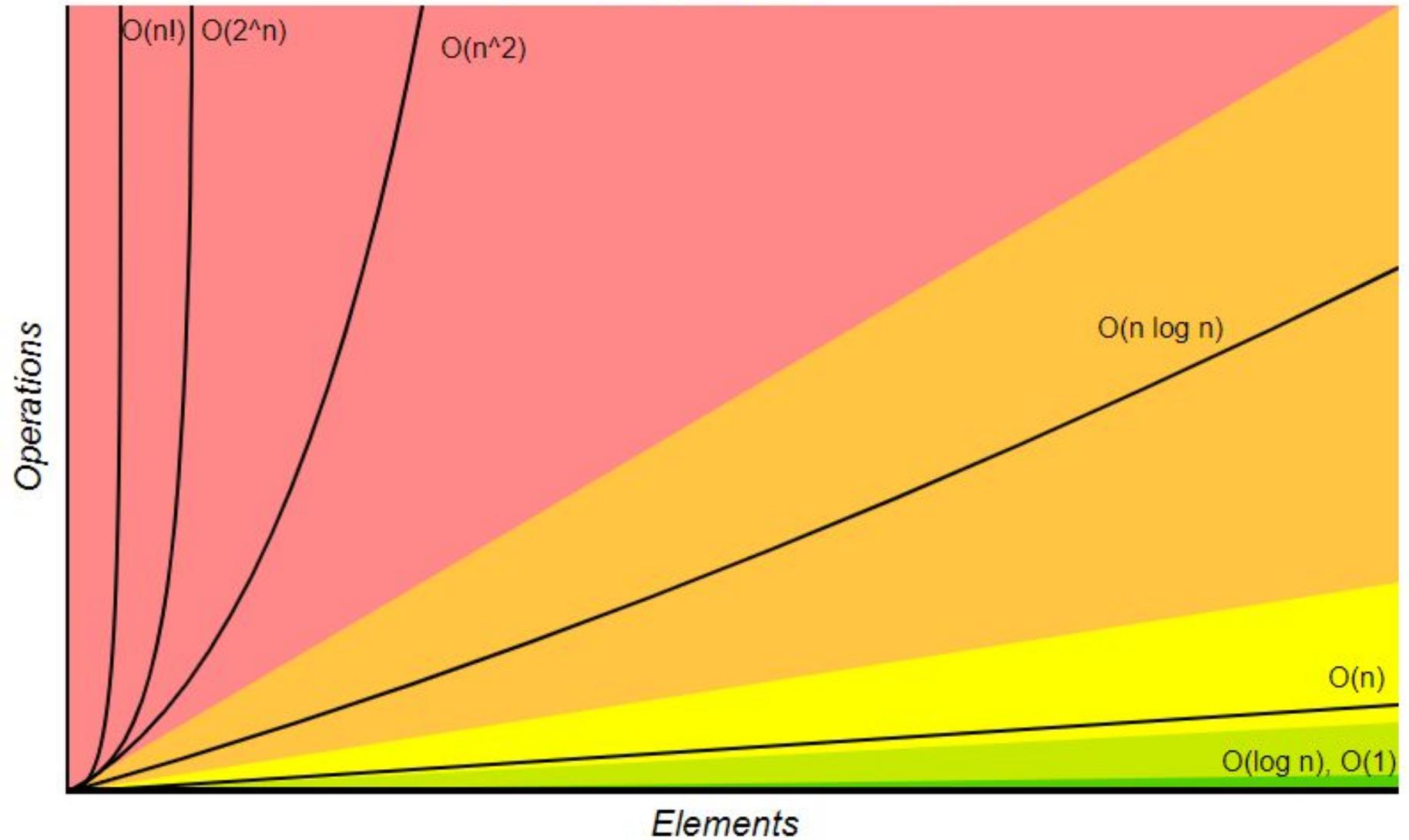
- Notación O-grande
- cota superior **asintótica**
 - ¿cómo se comporta nuestro algoritmo cuando el tamaño del problema (n) crece **indefinidamente**?
 - Nunca será mayor que su cota (a partir de cierto punto x_0)





Big-O Complexity Chart

Horrible Bad Fair Good Excellent



Complejidad

- Para $O(\text{Lg}(N))$ $N \Rightarrow [0, 2^{1000000}]$
- Para $O(N)$ $N \Rightarrow [0, 1.000.000]$
- Para $O(N\text{Lg}(N))$ $N \Rightarrow [0, 100.000]$
- Para $O(N^2)$ $N \Rightarrow [0, 5.000]$
- Para $O(N^3)$ $N \Rightarrow [0, 500]$
- Para $O(N^4)$ $N \Rightarrow [0, 60]$
- Para $O(2^N)$ $N \Rightarrow [0, 25]$
- Para $O(N!)$ $N \Rightarrow [0, 12]$

Complejidad

Ejemplo 1

```
const int MAXN = 1000;
const int MAXM = 2500;

int mat[MAXN][MAXM];

int main(){
    int N,M;
    scanf("%d %d",&N,&M);
    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            scanf("%d",&mat[i][j]);
        }
    }
}
```

Complejidad

Ejemplo 2

```
int mat[MAXN][MAXM];

int main(){
    int N,M;
    scanf("%d %d",&N,&M);
    for(int i=0;i<N;i++){
        for(int j=0;j<M;j++){
            scanf("%d",&mat[i][j]);
            if(mat[i][j] % 3 == 0){
                for(int k=0;k<N;k++){
                    mat[k][j] = mat[i][j]-20;
                }
            }
            else{
                mat[i][j]++;
            }
        }
    }
}
```

Estructuras de Datos Básicas

- Vectores
- Listas
- Pilas
- Colas
- Colas de prioridad (heap)
- Map (estructura <clave,valor>)
- Set (AVL)

Estructuras de Datos Básicas

Vectores

- Estructura en la cual se guardan valores sobre un elemento
- Funciona como un array, en memoria se guarda en espacios contiguos

Estructuras de Datos Básicas

Vectores / Vector

- ¡Si crece en mucha medida sin una previa reserva puede ser mortal!
- Su tamaño puede cambiar dinámicamente y esto lo convierte en un objetivo ideal para crear matrices dispersas
- El equivalente en Java podría ser ArrayList

Estructuras de Datos Básicas

Listas / List

- Estructura en la cual se guardan valores sobre un elemento, insertando en cualquier lugar un nuevo elemento
- Posiciones de memoria no-contiguas
- Inserciones y borrados mucho más fáciles de lograr internamente

Estructuras de Datos Básicas

Listas / List

- No se puede tener acceso directo a un elemento en específico (salvo primero o último)
- Puede servir tanto de pila como de cola
- Equivalente en Java: LinkedList

Estructuras de Datos Básicas

Pilas / stack

- Estructura donde el último elemento en llegar es el primero en salir (LIFO)
- No utiliza espacio contiguo de memoria
- Sólo se puede insertar elementos apilándolos
- Sólo se puede pedir elementos del tope de la pila

Estructuras de Datos Básicas

Pilas / stack

- En algunos casos, la pila puede “simular” una pila de sistema para reducir el tiempo de ejecución
- Ideal para DFS y otros algoritmos recursivos
- Equivalente en Java: ~~Stack (obsoleto)~~

Estructuras de Datos Básicas

Colas / queue

- Estructura donde el primer elemento en llegar es el primero en salir (FIFO)
- No utiliza espacio contiguo de memoria
- Sólo se puede insertar elementos al final (encolando)

Estructuras de Datos Básicas

Colas / queue

- Sólo se puede pedir el elemento del principio de la cola
- Ideal para BFS
- Equivalente en Java: Queue

Estructuras de Datos Básicas

Colas de prioridad / priority_queue

- Se ordenan de mayor a menor sus elementos (prioridad), internamente se puede representar como un heap (montículo)
- Solo se puede sacar el elemento tope de la estructura (en este caso el mayor elemento vendrá primero)
- Equivalente en Java: PriorityQueue

Estructuras de Datos Básicas

Mapa / map

- Contenedor asociativo que guarda claves únicas y les asocia a un valor
- Se puede acceder directamente a elementos guardados si se coloca su clave, en tal caso, se devolverá el valor asociado
- Al ser un árbol binario balanceado, todas sus operaciones son logarítmicas (también podría ser tabla hash)
- Equivalente en Java: HashMap / TreeMap

Estructuras de Datos Básicas

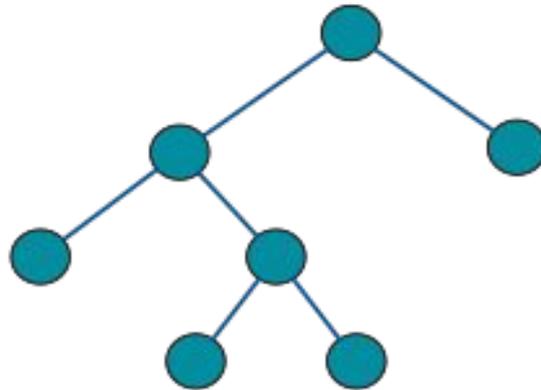
Conjuntos / set

- Árbol binario balanceado, ordena naturalmente de menor a mayor (podría ser tabla hash)
- No admite elementos repetidos
- Equivalente en Java: TreeSet / HashSet

Estructuras de Datos Básicas

Arboles Binarios

- Estructura que tiene un nodo raíz
- Desde el nodo raíz podrán haber hasta dos hijos, de esos hijos pueden haber otros dos hijos más y así sucesivamente.



Estructuras de Datos Básicas

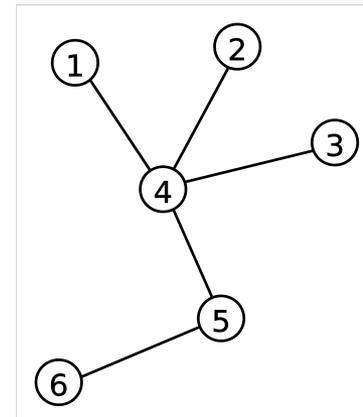
Arboles Binarios

- Se puede implementar con punteros (llevando cuenta en una estructura aparte cuál es el nodo derecho e izquierdo)
- Con arrays, teniendo en cuenta que el índice 0 es la raíz, el hijo izquierdo sería $0*2+1$ y el derecho sería $0*2+2$ (para el índice n , el hijo izquierdo será $n*2+1$ y el derecho $n*2+2$)

Estructuras de Datos Básicas

Árboles Generales

- Implementándolos a través de una clase llevando cuenta del nodo raíz y en cada nodo llevar cuenta de los hijos en un array.
- Ideales para Tries (futuro)



Estructuras de Datos Básicas

Estructura	Adición	Delección	Búsqueda
Vector / Array	$O(1)^*$	$O(1)^*$	$O(N)^*$
Lista	$O(1)$	$O(1)$	$O(N)$
Pila	$O(1)$	$O(1)$	$O(N)$
Cola	$O(1)$	$O(1)$	$O(N)$
Set	$O(\text{Lg}N)^*$	$O(\text{Lg}N)^*$	$O(\text{Lg}N)^*$
Map	$O(\text{Lg}N)^*$	$O(\text{Lg}N)^*$	$O(\text{Lg}N)$
Priority_Queue	$O(\text{Lg}N)$	$O(\text{Lg}N)$	$O(N)$

Algoritmos de Ordenamiento

Algunos problemas requieren tener ordenados una serie de elementos para dar una respuesta

- Algoritmos de ordenación
- Estructuras de datos ordenadas

... es importante su eficiencia

Algoritmos de Ordenamiento

- BubbleSort y SelectionSort
 - Complejidad: $O(n^2)$ (ineficientes)
 - No se incluyen en las bibliotecas estándar
 - Se implementan como ejercicio de aprendizaje

Algoritmos de Ordenamiento

- Quicksort / Mergesort
 - Complejidad: $O(n\log(n))$
 - Ya están implementadas en las bibliotecas básicas ¡No hay que programarlos!

Algoritmos de Ordenamiento

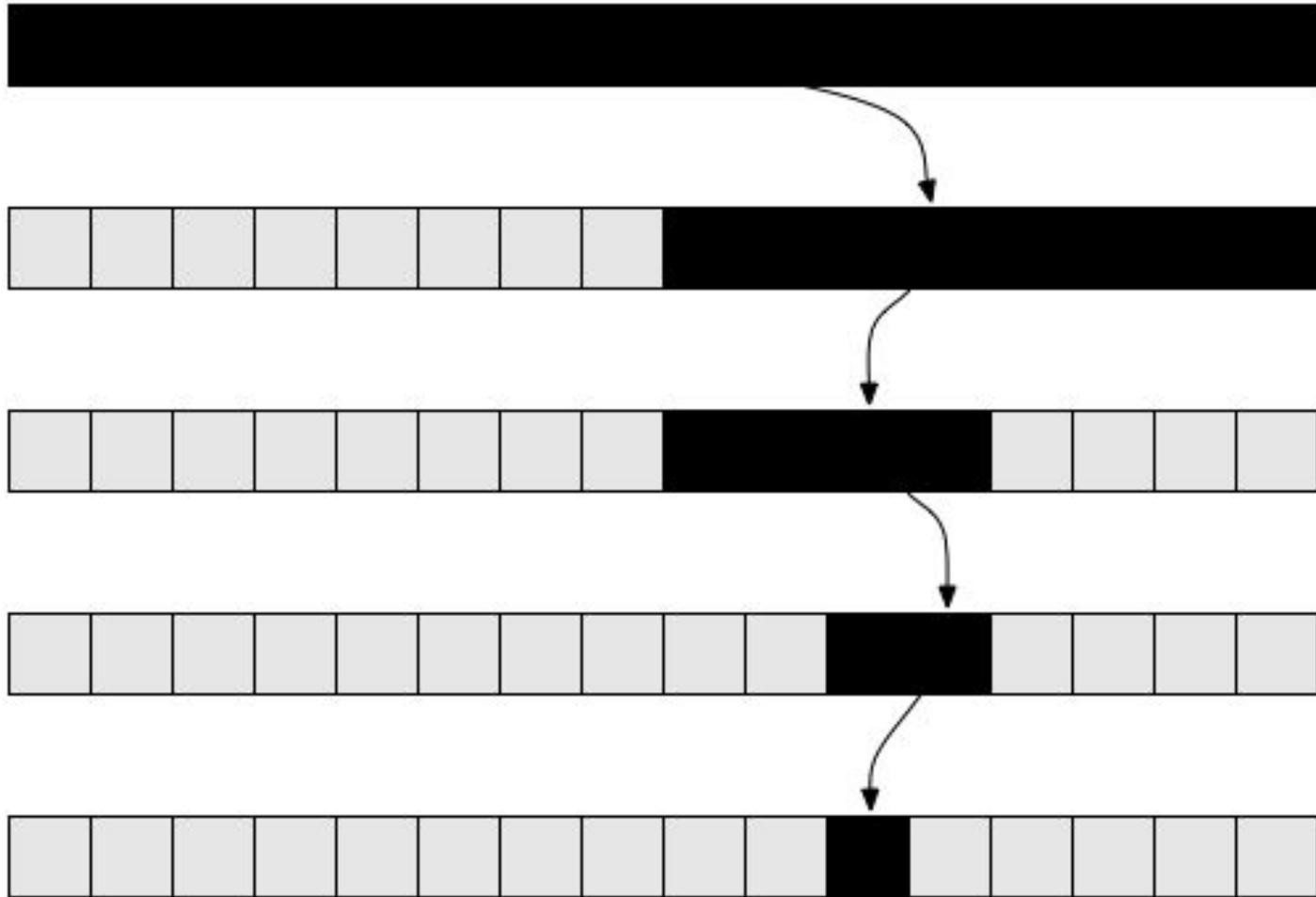
- C++
 - `std::sort()`
- Java
 - `Collections.sort()` -> MergeSort*
 - `Arrays.sort()` -> Quicksort* (para tipos primitivos)

*En realidad son variantes más eficientes

Búsqueda Binaria

- Se implementan sobre funciones monótonas crecientes / decrecientes
- Para todo x se tiene que cumplir:
 - $f(x) > f(x+1)$
 - ó $f(x) < f(x+1)$
- Su complejidad es logarítmica

Búsqueda Binaria



Búsqueda Binaria

Pseudocódigo:

inferior = 0, superior = N

mientras superior-inferior > 1

 medio = (superior+inferior)/2

 si $f(\text{medio}) < \text{objetivo}$

 inferior = medio

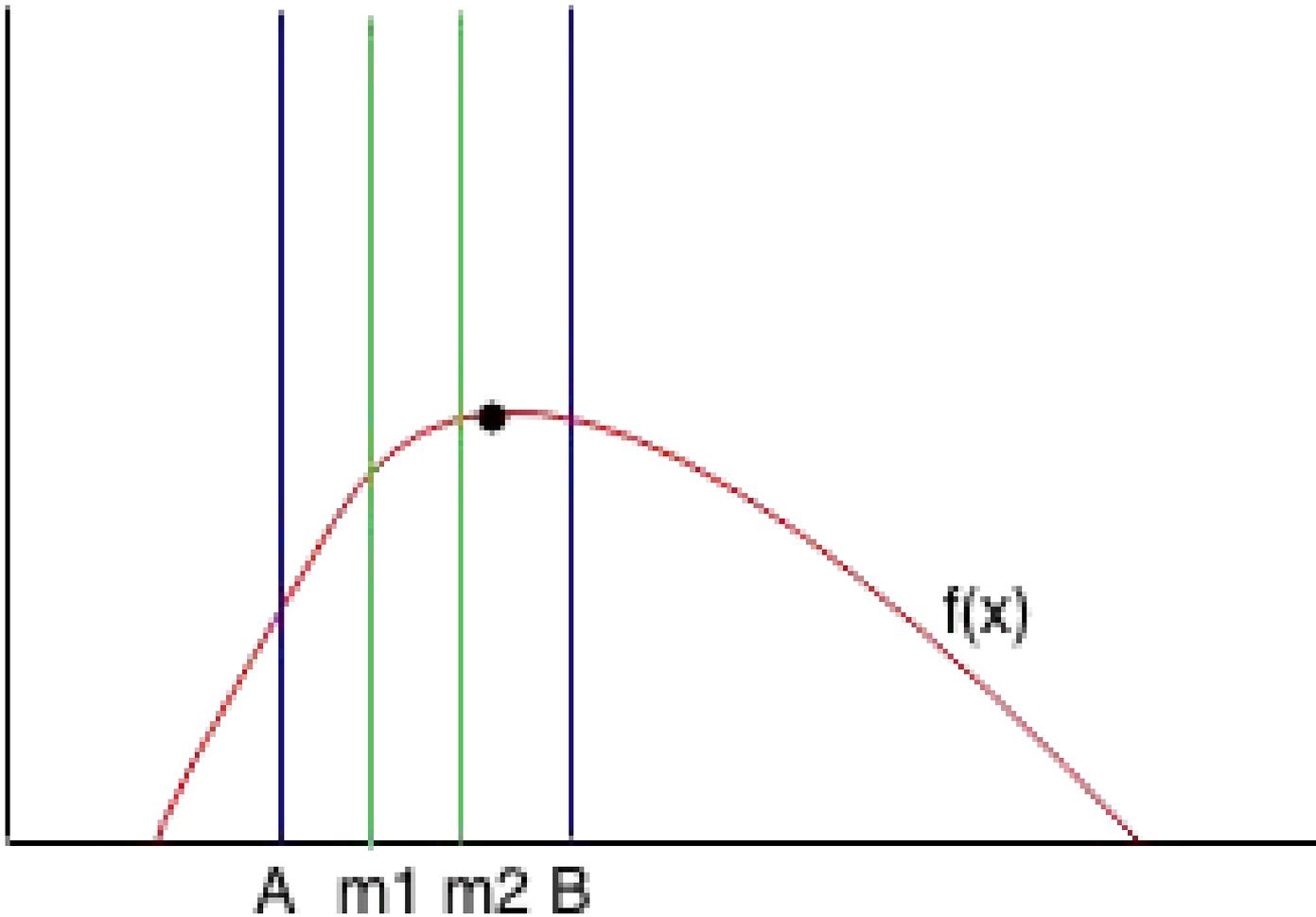
 sino

 superior = medio

Búsqueda Ternaria

- Se implementan sobre funciones parabólicas donde haya un solo punto mínimo ó máximo
- Su complejidad es $2 \cdot \log_3(N)$
- No es muy común encontrarse con este tipo de problemas

Búsqueda Ternaria



Búsqueda Ternaria

Pseudocódigo:

inferior = 0, superior = N

mientras superior-inferior > 2

$m1 = (\text{superior} + \text{inferior} * 2) / 3$

$m2 = (\text{superior} * 2 + \text{inferior}) / 3$

 si $f(m1) < f(m2)$

 inferior = m1

 sino

 superior = m2

Exponenciación Logarítmica

- Una forma de elevar a^b siempre que b sea entero
 - Se deja a ejercicio razonar como se haría en una matriz
- Parte de la base de divide y vencerás y de la propiedad de que $a^b * a^b = a^{(b+b)}$

Exponenciación Logarítmica

Pseudocódigo

```
si b == 1 retornar a
sino
  tmp = f(a, b/2)
  si b % 2 (impar)
    retornar tmp*tmp*a
  sino
    retornar tmp*tmp
```

Exponenciación Logarítmica

Ejemplo:

$$2^{10} \Rightarrow 2^{5*2^5}$$

$$2^5 \Rightarrow 2^{2*2^2*2}$$

$$2^2 \Rightarrow 2^{1*2^1}$$

$$2^1 = 2$$

Exponenciación Logarítmica

Ejemplo:

$$2^{10} \Rightarrow 2^{5*2} = 32*32 = 1024$$

$$2^5 \Rightarrow 2^{2*2*2} = 4*4*2 = 32$$

$$2^2 \Rightarrow 2^{1*2} = 2*2 = 4$$

$$2^1 = 2$$

Semana que viene

- Algoritmos Voraces
- Grafos

¡Hasta la próxima semana!

Ante cualquier duda sobre el curso o sobre los problemas podéis escribirnos (preferiblemente copia a los tres)

David Morán (ddavidmorang@gmail.com)

Juan Quintana (juandavid.quintana@urjc.es)

Sergio Pérez (sergioperezp1995@gmail.com)