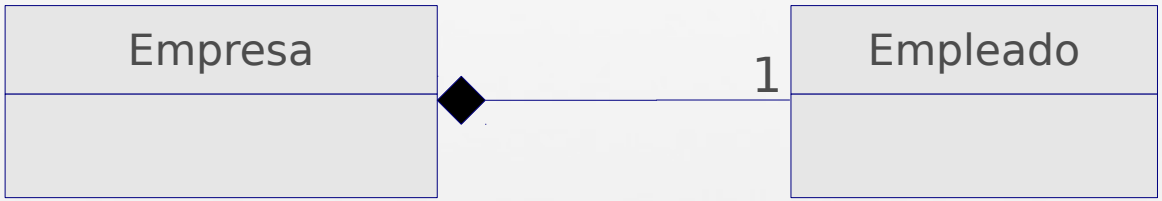
Lección 2:
Contenedores y Tipos

- Motivación
- Definiciones
- Clasificación de las EEDD
- Eficiencia de las EEDD
- Análisis de la eficiencia
- Cálculo del análisis de la eficiencia
- De ahora en adelante....

Motivación

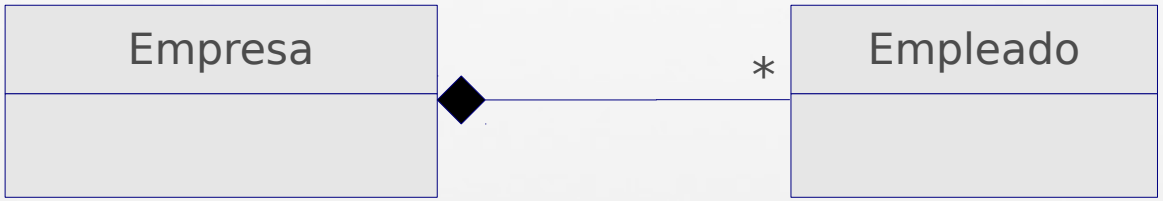
Alberto tiene una pequeña empresa, tiene un empleado llamado Ramiro




```
class Empresa{
    Empleado Ramiro;
Public:
    Empresa (Empleado &r);
    …
};
```

Motivación

Pero la empresa de Elías tiene 25 empleados y piensa ampliar pronto la plantilla.



Necesita una estructura de datos para almacenar **muchos** datos

```
class Empresa{
    vector<Empleado> empleados;
public:
    Empresa ():empleados();
    nuevoEmpleado (Empleado &e);
    …
};
```

Definiciones

Las siguientes definiciones son afines:

- Una **estructura de datos** es una forma particular de guardar y organizar la información en un ordenador de forma que pueda ser usado eficientemente.
- Un **contenedor** es una estructura de datos que sirve para guardar colecciones de otros objetos.
- Un **tipo de dato abstracto (TDA)** es un modelo matemático para una estructura de datos definida indirectamente por las operaciones que la manejan.
- Una **clase de objetos** es la representación de un TDA que se hace en un lenguaje orientado a objetos.

Clasificación de las EEDD

Las estructuras de datos pueden clasificarse de muy distintos modos.

- Atendiendo al tipo de acceso:
  - Acceso directo: **vectores, matrices, pilas, colas, etc.**
  - Acceso secuencial: **listas, matrices dispersas, etc.**
  - Acceso por clave: **árboles, tablas hash, etc.**
- Atendiendo a su estructura interna:
  - Lineales: **vectores, listas, pilas, matrices, etc.**
  - No lineales: **heap, árbol AVL, quadtree, grafos**.

Clasificación de las EEDD

- Atendiendo al manejo de memoria
  - Estáticas: **vectores estáticos, matrices estáticas, etc.**
  - Dinámicas: **listas, vectores dinámicos, árboles.**
- Atendiendo a su ubicación:
  - Memoria: **vectores, listas, pilas, etc.**
  - Fichero: **árboles B, ficheros dispersos, etc.**
  - Mixtos: **índices simples, índices secundarios, etc.**
- Atendiendo a su dimensionalidad:
  - Unidimensionales: **vectores, listas, pilas, etc.**
  - Bi/Multi-dimensionales: **matrices, grafos, quadtrees, etc.**

Clasificación de las EEDD

La elección de una estructura de datos depende del tipo de algoritmo que resuelve el problema:

- Búsqueda eficiente para grandes cantidades de datos:
  - Se usa un campo para la búsqueda: **árboles avl, tablas hash, eedd en ficheros, etc.**
  - Se organizan por dos (o más) campos: **quadtree, mallas regulares, etc.**
- Acceso específico según el problema:
  - El dato buscado siempre está en una misma ubicación: **pilas, colas, colas con prioridad, etc.**
  - Problemas de caminos: **grafos**

Eficiencia de las EEDD

La biblioteca de la universidad tiene más de 30.000 libros, cómo puedo organizarlos para localizarlos de forma **eficiente** por ISBN? Posible opciones:

- Búsqueda lineal: uno por uno hasta 30.000 libros.
- En un array ordenado: búsqueda binaria o dicotómica con unos 15 accesos.

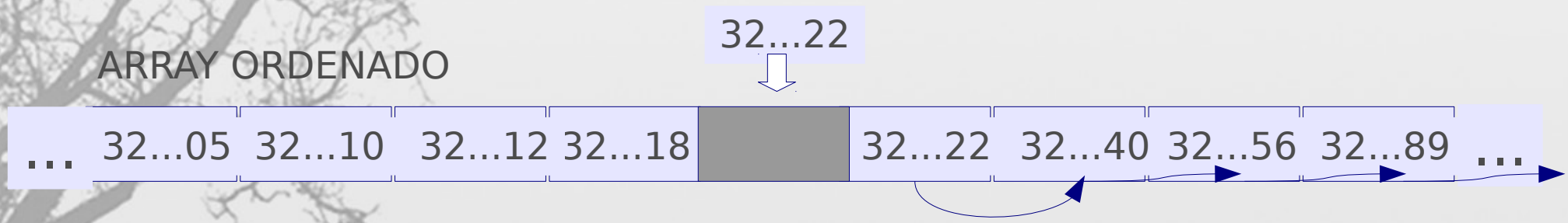
ARRAY DESORDENADO

ARRAY ORDENADO

Eficiencia de las EEDD

El tiempo que tarda en ejecutarse un algoritmo depende de **n**, el tamaño de los datos a procesar

- En el ejemplo n=30.000
- En el primer caso se tarda un tiempo proporcional a ***n*** para realizar la búsqueda
- En el segundo caso es proporcional a $\boldsymbol{log_2 n}$
- Pero insertar un nuevo dato necesitaría ***n*** accesos. Ninguna solución es eficiente



Eficiencia de las EEDD

Es muy importante elegir la estructura de datos más adecuada en cada caso:

- Para el ejemplo de la biblioteca lo ideal es utilizar un **contenedor asociativo o de acceso por clave** que permita búsquedas en tiempo logarítmico e inserciones también eficientes
- Nuestro objetivo en el curso es buscar la estructura de datos **más adecuada** para cada problema
- Las estructuras de datos y los algoritmos están íntimamente relacionados a la hora de resolver un problema

Análisis de eficiencia

- El tiempo de ejecución de un programa puede depender de:
  - el número de datos ***n***,
  - lo rápido que sea el ordenador,
  - la secuencia de los datos de entrada,
  - la calidad del diseño del algoritmo.
- Se expresa como ***T(n)*** el tiempo que tarda en ejecutase un programa ***P***
- Un problema puede resolverse con dos algoritmos distintos, uno tardará $T_1(n)$ *y el otro* $T_2(n)$. Si $T_1(n)<T_2(n)$ entonces el primero es más eficiente

Análisis de eficiencia

- Un algoritmo se ejecuta en tiempo $T_1(n) = 4\ \boldsymbol{log_2 n}$
- Pero otra secuencia de datos del mismo programa tarda $T_2(n) = 2\ \boldsymbol{n}$
- La mejor ejecución resultó ser $T_3(n) = 2\ \boldsymbol{log_2 n}$

Para hacer la medición independiente de la ejecución, se emplea la complejidad en **sentido asintótico**, es decir, para un tañano muy grande (tendiendo a infinito)

- $T(n)=O(f(n))$ → **peor caso o notación O grande**
  - Si y sólo si existen dos valores reales C y C_0 t.q.: $T(n) <= C\ f(n)$ para todo $C>C_0$
  - Lo que indica que f(n) es una **cota superior** y cualquier ejecución está por debajo de este valor.

Análisis de eficiencia

• En el ejemplo anterior como la peor ejecución fue T(n)=5n, decimos que T(n) = O(n).

La notación O grande es la más usada por ser la cota superior, pero también se puede estudiar el mejor caso:

- T(n)=Ω (f(n)) → **mejor caso** o Gran Omega
  - Si y sólo si existen dos valores reales C y C_0 t.q.: T(n) >= C f(n) para todo C>C_0
  - Lo que indica que f(n) es una **cota inferior** y cualquier ejecución está por encima de este valor.
- Esta notación es útil porque a veces se prevé que un algoritmo se comportará de forma óptima

Análisis de eficiencia

Si $T(n)=c_1n^2+c_2n$ para c1 y c2 > 0, entonces:

$$|c_1n^2+c_2n| \geq |c_1n^2| \geq c_1 |n^2|$$

Por lo tanto, T(n) está en $\Omega(n^2)$.

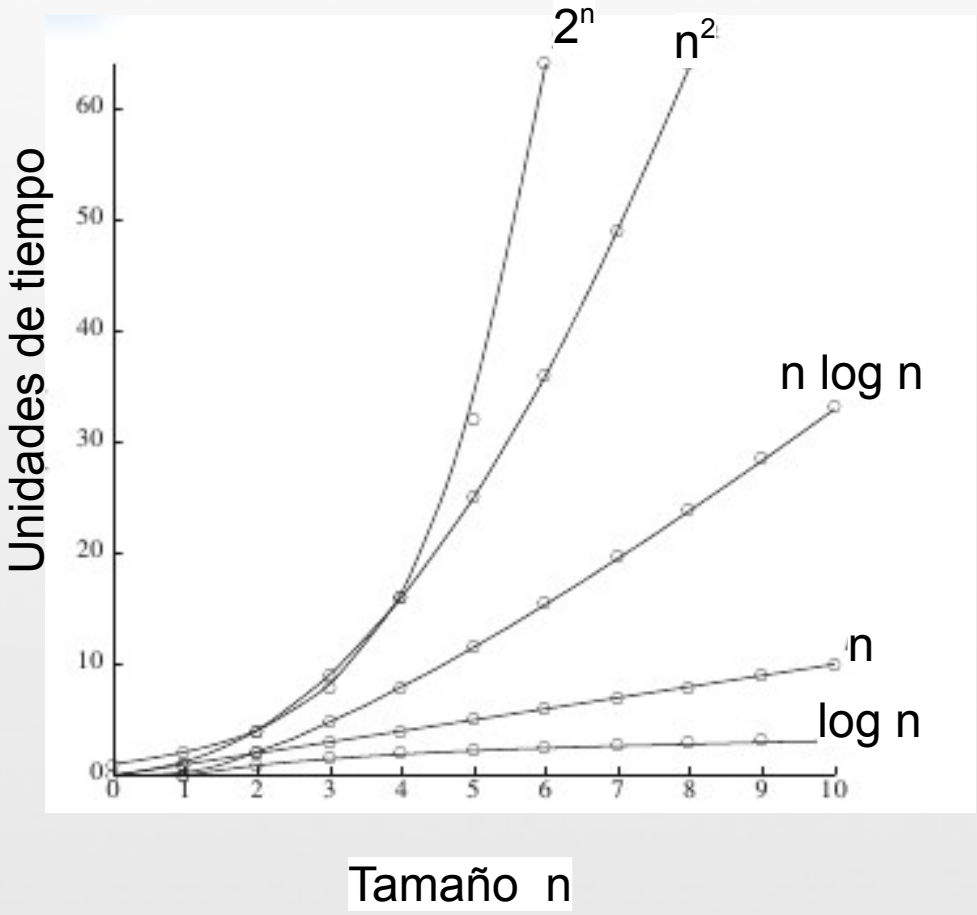
- Las funciones suelen simplificarse y las constantes obviarse en las notaciones asintóticas
- Cuando la cota superior e inferior coinciden se usa otra notación: $T(n)= \Theta(f(n))$
  - Si un problema se resuelve en tiempo $\Theta(f(n))$, sabemos que hemos encontrado la solución óptima, no se puede mejorar asintóticamente

Análisis de eficiencia

En este contexto ***log*** siempre es en base 2

Lectura de órdenes

- O(1) constante
- O(log n) logarítmico
- O(n) lineal
- O(n log n)
- $O(n^2)$ cuadrático
- $O(n^a)$ polinomial
- $O(a^n)(a > 2)$ exponencial
- O(n!)(a > 2) factorial



Análisis de eficiencia

- En general **O(1)<O(log n)<O(n)<O(nlogn)<O(n^2)**
- Pero no siempre la mejor solución asintótica es más rápida para un conjunto pequeño de datos.
- Las constantes obviadas son importantes para tamaños no elevados de los datos de entrada
- Pero un algoritmo debe diseñarse para cualquier n

Por ejemplo, para n=100:

- $5n < 200 \log n$
- $2n \log n < 30n$
- $n^2 < 25 n \log n$
- $n^3 < 125n^2$

Cálculo del análisis de eficiencia

```
if (a>b) {
    revisarTodo();
} else {
    revisarAlgo();
}
```

- La función revisarTodo() es $O(n^2)$
- La función revisarAlgo() es $O(n)$
- El fragmento completo es $\max(n,n^2)$, por tanto $O(n^2)$

```
void granProblema(1,n){
….
  if (n<3) {
    resuelvo()
  } else {
    a=granProblema(1,n/2)
    b=granProblema(n/2+1,n)
    c=unirSoluciones(a,b)
  }
}
```

- resuelvo() = constante
- unirSoluciones = cn
- $T(n)= 2T(n/2) + cn =$

$= 2(2T(n/4) + cn/2) + cn = 4T(n/4) + 2c$
$= 4(2T(n/8) + cn/4) + 2cn$
$= 8T(n/8) + 3cn \ldots\ldots = nT(1) + cn \log_2 n$
$= O(n \log n)$

Cálculo del análisis de eficiencia

```
sum=0;
for(i=0; i<n; i++)
    sum += n;
```

- La primera línea es O(1).
- El ciclo “for” es repetido n veces.
  - La tercera línea también es O(1)
- El fragmento completo es O(n).

```
sum=0;
for(j=0; j<n; j++)
    for(i=0; i<j; i++)
        sum++;
for(k=0; k<n; k++)
    A[k]= k-1;
```

- La primera línea es O(1).
- Los dos primeros bucles anidados son $O(n^2)$
- El último bucle es O(n)
- El fragmento completo es $O(n^2)$.

De ahora en adelante....

- Para cada estructura de datos consideraremos el tiempo que necesitan para:
  - Insertar/borrar
  - Buscar
  - Ordenar (en su caso)
- En base a esto se adaptarán mejor o peor a la solución de un problema.
- En muchas ocasiones la elección de la EEDD correcta determina directamente la eficiencia global del algoritmo