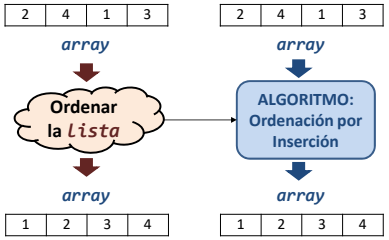
Universidad Politécnica de Madrid
Escuela Técnica Superior de Ingeniería de Sistemas Informáticos

Tema 2. Análisis de Complejidad

Algorítmica y Complejidad

Introducción

Problemas y Funciones: Ejemplos



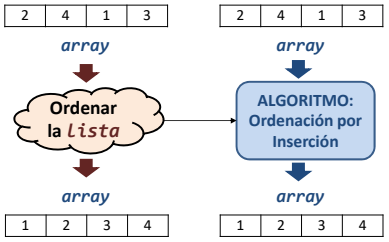
¿Cuánto tarda el algoritmo en ordenar?

- Dependerá del computador donde se ejecuta.

ESTUDIAREMOS UNA MEDIDA INDEPENDIENTE DEL COMPUTADOR

Introducción

Problemas y Funciones: Ejemplos

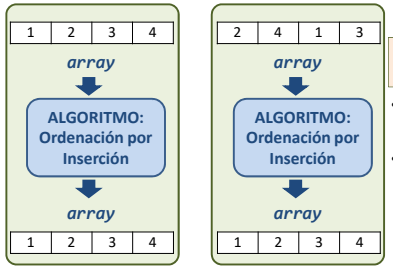


¿Cuánto tarda el algoritmo en ordenar?

- Dependerá de la **ENTRADA** (lista a ordenar).
- Estudiaremos características de la ENTRADA que reflejen el tamaño del problema:
  - Número de elementos del *array* (***N***).

Introducción

Problemas y Funciones: Ejemplos



¿Cuánto tarda el algoritmo en ordenar?

- Dependerá de la **ENTRADA** (lista a ordenar).
- Estudiaremos características de la ENTRADA que reflejen el tamaño del problema:
  - Número de elementos del *array* (***N***).

Para *arrays* con el mismo número de elementos, el algoritmo tarda un tiempo diferente.

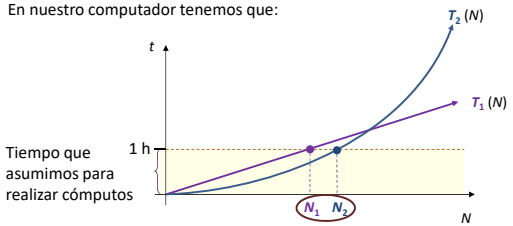
Caso Mejor | Caso Promedio | **Caso Peor**

Introducción

Motivación

N: Tamaño del Problema

En nuestro computador tenemos que:



Tiempo que asumimos para realizar cómputos

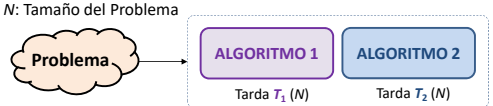
Circunstancia Coyuntural

ALGORITMO 2 es preferible en nuestro computador

Introducción

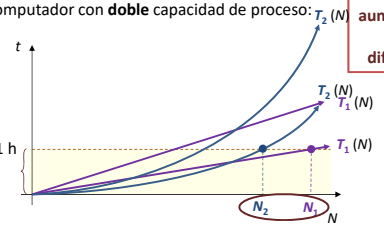
Motivación

N: Tamaño del Problema



Compramos un computador con **doble** capacidad de proceso:

A medida que aumenta la capacidad de proceso la diferencia aumenta



Tiempo que asumimos para realizar cómputos

Los tiempos se reducen a la mitad

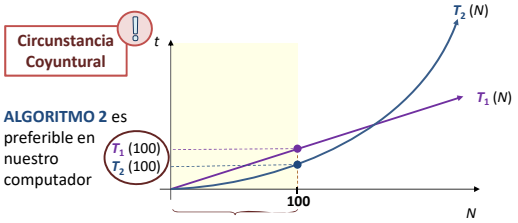
ALGORITMO 1 es ahora preferible en nuestro nuevo computador

Introducción

Motivación

N: Tamaño del Problema

Circunstancia Coyuntural



ALGORITMO 2 es preferible en nuestro computador

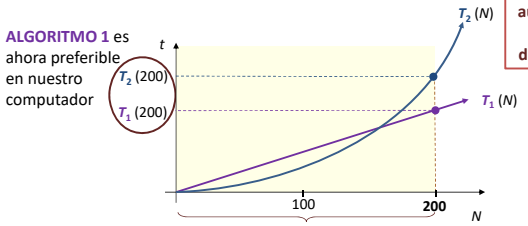
Tamaño del problema actual

Introducción

Motivación

N: Tamaño del Problema

A medida que aumenta el tamaño del problema la diferencia aumenta



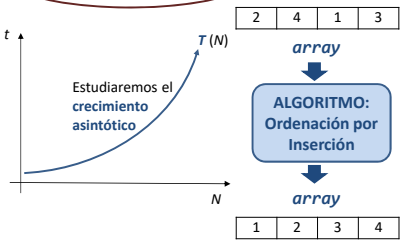
ALGORITMO 1 es ahora preferible en nuestro computador

Tamaño del problema actual

Notación Asintótica

Crecimiento

Notación: **O** **Ω** **Θ**



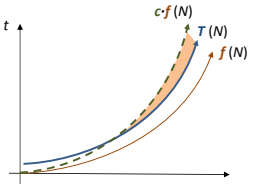
¿Cuánto tarda el algoritmo en ordenar?

- Dependerá de la **ENTRADA** (lista a ordenar).
- Estudiaremos características de la ENTRADA que reflejen el tamaño del problema:
  - Número de elementos del *array* (***N***).

Notación Asintótica

Notación O: Definición

Notación: **O** $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de O *"Cota asintótica Superior"*

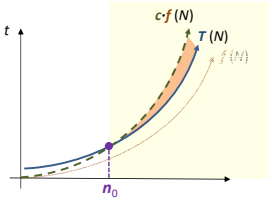
$T(N) \in O(f(N))$ si y solo si
$\exists c\ \exists n_0$ tal que $\forall n > n_0\ T(n) \le c \cdot f(n)$

Se puede entender como "$T \le f$"

Notación Asintótica

Notación O: Definición

Notación: **O** $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de O *"Cota asintótica Superior"*

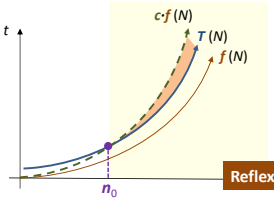
$T(N) \in O(f(N))$ si y solo si
$\exists c\ \exists n_0$ tal que $\forall n > n_0\ T(n) \le c \cdot f(n)$

Se puede entender como "$T \le f$"

Notación Asintótica

Notación O: Propiedades

Notación: **O** $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de O *"Cota asintótica Superior"*

$T(N) \in O(f(N))$ si y solo si
$\exists c\ \exists n_0$ tal que $\forall n > n_0\ T(n) \le c \cdot f(n)$

Se puede entender como "$T \le f$"

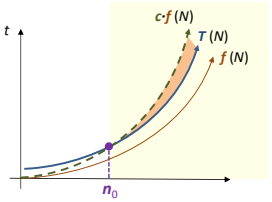
Reflexividad
- $f(N) \in O(f(N))$

Transitividad
- $f(N) \in O(g(N))$, $g(N) \in O(h(N)) \Rightarrow f(N) \in O(h(N))$

Notación Asintótica

Notación O: Cálculo a través de límites

Notación: **O** $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de O *"Cota asintótica Superior"*

$T(N) \in O(f(N))$ si y solo si
$\exists c\, \exists n_0$ tal que $\forall n > n_0\ T(n) \le c \cdot f(n)$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \Rightarrow f(N) \in O(g(N)),\ g(N) \notin O(f(N))$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} \in (0,\infty) \Rightarrow f(N) \in O(g(N)),\ g(N) \in O(f(N))$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \infty \Rightarrow f(N) \notin O(g(N)),\ g(N) \in O(f(N))$

Asignatura: *Análisis Matemático*

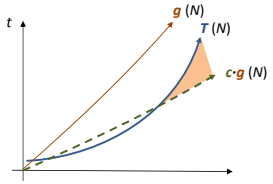
Cálculo de límites:
- Regla L'Hopital

2. Notación Asintótica
1 2 3

Notación Asintótica

Notación Ω: Definición

Notación: O $T(N) \in O(f(N))$ | **Ω** $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Ω *"Cota asintótica Inferior"*

$T(N) \in \Omega(g(N))$ si y solo si
$\exists c\, \exists n_0$ tal que $\forall n > n_0\ T(n) \ge c \cdot g(n)$

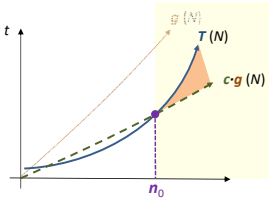
Se puede entender como "$T \ge g$"

2. Notación Asintótica
1 2 3

Notación Asintótica

Notación Ω: Definición

Notación: O $T(N) \in O(f(N))$ | **Ω** $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Ω *"Cota asintótica Inferior"*

$T(N) \in \Omega(g(N))$ si y solo si
$\exists c\, \exists n_0$ tal que $\forall n > n_0\ T(n) \ge c \cdot g(n)$

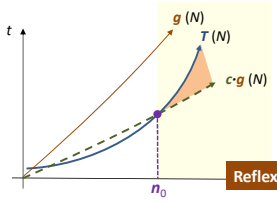
Se puede entender como "$T \ge g$"

2. Notación Asintótica
1 2 3

Notación Asintótica

Notación Ω: Propiedades

Notación: O $T(N) \in O(f(N))$ | **Ω** $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Ω *"Cota asintótica Inferior"*

$T(N) \in \Omega(g(N))$ si y solo si
$\exists c\, \exists n_0$ tal que $\forall n > n_0\ T(n) \ge c \cdot g(n)$

Se puede entender como "$T \ge g$"

Reflexividad
- $g(N) \in \Omega(g(N))$

Transitividad
- $g(N) \in \Omega(f(N))$, $f(N) \in \Omega(h(N)) \Rightarrow g(N) \in \Omega(h(N))$

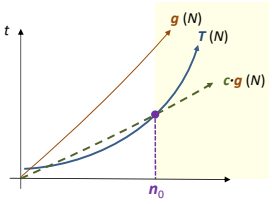
Dualidad
- $g(N) \in \Omega(f(N)) \Leftrightarrow f(N) \in O(g(N))$

2. Notación Asintótica
1 2 3

Notación Asintótica

Notación Ω: Cálculo a través de límites

Notación: O $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Ω *"Cota asintótica Inferior"*

$T(N) \in \Omega(g(N))$ si y solo si
$\exists c\ \exists n_0$ tal que $\forall n > n_0\ T(n) \geq c \cdot g(n)$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \Rightarrow f(N) \notin \Omega(g(N))$, $g(N) \in \Omega(f(N))$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} \in (0,\infty) \Rightarrow f(N) \in \Omega(g(N))$, $g(N) \in \Omega(f(N))$

$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \infty \Rightarrow f(N) \in \Omega(g(N))$, $g(N) \notin \Omega(f(N))$

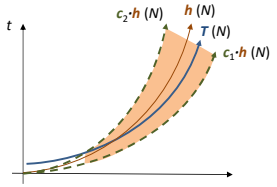
Asignatura:
Análisis Matemático

Cálculo de límites:
- Regla L'Hopital

Notación Asintótica

Notación Θ: Definición

Notación: O $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Θ

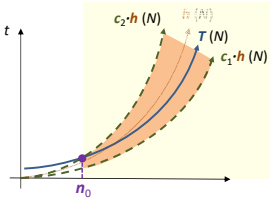
$T(N) \in \Theta(h(N))$ si y solo si
$\exists c_1\ \exists c_2\ \exists n_0$ tal que $\forall n > n_0\ c_1 \cdot h(n) \leq T(n) \leq c_2 \cdot h(n)$

Se puede entender como "T = h"

Notación Asintótica

Notación Θ: Definición

Notación: O $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Θ

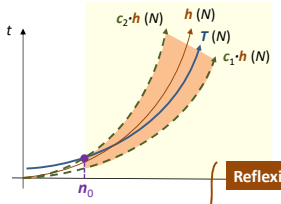
$T(N) \in \Theta(h(N))$ si y solo si
$\exists c_1\ \exists c_2\ \exists n_0$ tal que $\forall n > n_0\ c_1 \cdot h(n) \leq T(n) \leq c_2 \cdot h(n)$

Se puede entender como "T = h"

Notación Asintótica

Notación Θ: Propiedades

Notación: O $T(N) \in O(f(N))$ | Ω $T(N) \in \Omega(g(N))$ | Θ $T(N) \in \Theta(h(N))$



Definición de Θ

$T(N) \in \Theta(h(N))$ si y solo si
$\exists c_1\ \exists c_2\ \exists n_0$ tal que $\forall n > n_0\ c_1 \cdot h(n) \leq T(n) \leq c_2 \cdot h(n)$

Se puede entender como "T = h"

Relación de Equivalencia

Asignatura:
Matemática Discreta

Reflexividad
- $h(N) \in \Theta(h(N))$

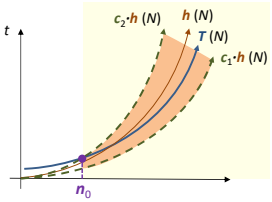
Transitividad
- $g(N) \in \Theta(f(N))$, $f(N) \in \Theta(h(N)) \Rightarrow g(N) \in \Theta(h(N))$

Simetría
- $f(N) \in \Theta(h(N)) \Leftrightarrow h(N) \in \Theta(f(N))$
- $f(N) \in O(h(N))$, $f(N) \in \Omega(h(N)) \Leftrightarrow f(N) \in \Theta(h(N))$

Notación Asintótica

Notación Θ: Cálculo a través de límites

Notación: O | $T(N) \in O(f(N))$ | Ω | $T(N) \in \Omega(g(N))$ | Θ | $T(N) \in \Theta(h(N))$



Definición de Θ

$T(N) \in \Theta(h(N))$ si y solo si
$\exists c_1 \exists c_2 \exists n_0$ tal que $\forall n > n_0\ c_1 \cdot h(n) \leq T(n) \leq c_2 \cdot h(n)$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = 0 \Rightarrow f(N) \notin \Theta(g(N)),\ f(N) \in O(g(N))$$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} \in (0,\infty) \Rightarrow f(N) \in \Theta(g(N))$$

$$\lim_{N\to\infty} \frac{f(N)}{g(N)} = \infty \Rightarrow f(N) \notin \Theta(g(N)),\ f(N) \in \Omega(g(N))$$

Asignatura:
Análisis Matemático

Cálculo de límites:
- Regla L'Hopital

Notación Asintótica

Actividad 2.1. Justifica si es verdadero o falso las siguientes afirmaciones:
- $\sqrt{N} \in O(\ln N)$
- $\sqrt{N} \in \Omega(\ln N)$
- $\sqrt{N} \in \Theta(\ln N)$

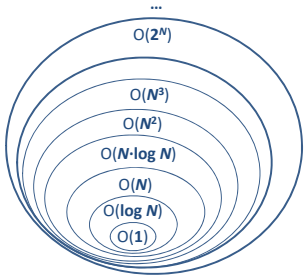
Notación Asintótica

Actividad 2.2. Justifica si es verdadero o falso las siguientes afirmaciones:
- $\sqrt{N} \cdot \ln N \in O(N)$
- $\sqrt{N} \cdot \ln N \in \Omega(N)$
- $\sqrt{N} \cdot \ln N \in \Theta(N)$

Notación Asintótica

Jerarquía de Ordenes

Notación: O | $T(N) \in O(f(N))$ "$T \leq f$" | Ω | $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ | $T(N) \in \Theta(h(N))$ "$T = h$"

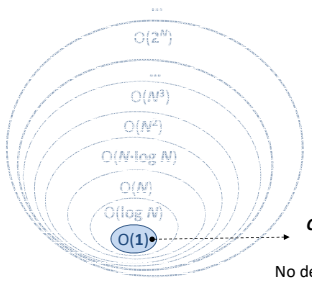


Notación Asintótica

Problemas y Funciones: Ejemplos

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"

Algoritmos Eficientes



Complejidad Constante

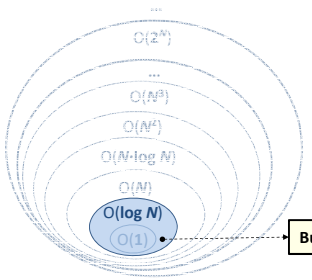
No depende de la entrada

Notación Asintótica

Problemas y Funciones: Ejemplos

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"

Algoritmos Eficientes



Complejidad Logarítmica

Búsqueda Binaria

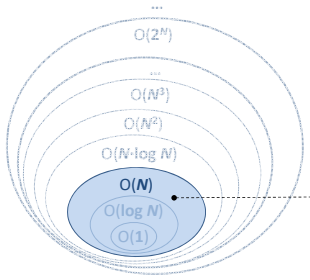
Búsqueda en un vector

Notación Asintótica

Problemas y Funciones: Ejemplos

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"

Algoritmos Eficientes



Complejidad Lineal

Algoritmo

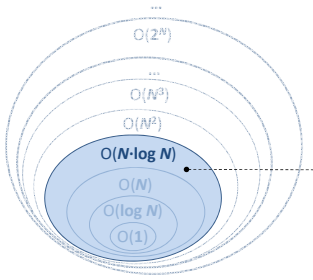
Máximo de un array

Notación Asintótica

Algoritmos Quasi-lineales

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"

Algoritmos Eficientes



Complejidad Quasi-Lineal

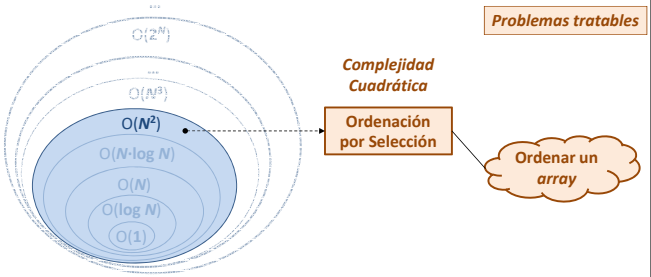
MergeSort

Ordenar un *array*

Notación Asintótica

Problemas y Funciones: Ejemplos

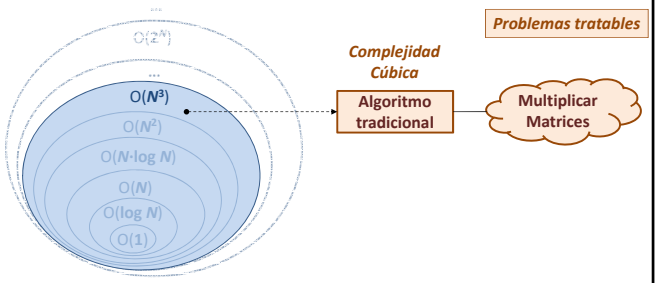
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Problemas y Funciones: Ejemplos

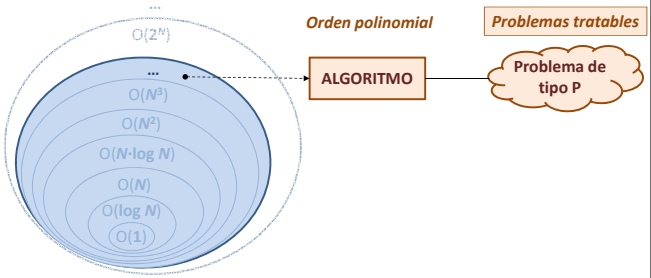
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Problemas y Funciones: Ejemplos

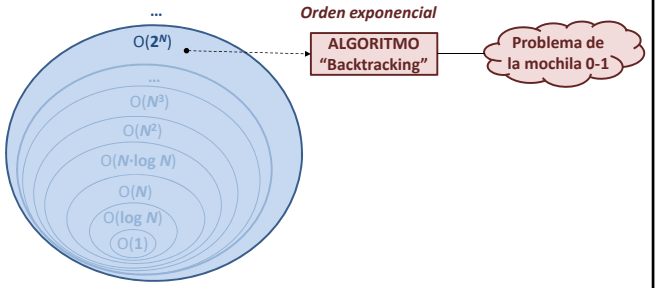
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Cálculo de la complejidad

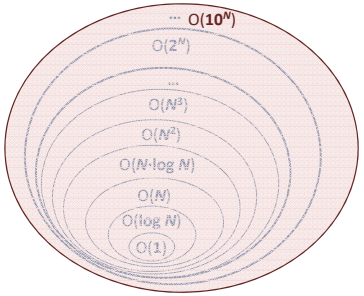
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Otros órdenes de complejidad: O(log log *N*)

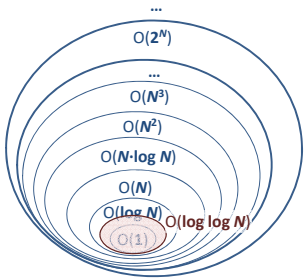
Notación: O $T(N) \in \mathbf{O}(f(N))$ "$T \leq f$" | Ω $T(N) \in \mathbf{\Omega}(g(N))$ "$T \geq g$" | Θ $T(N) \in \mathbf{\Theta}(h(N))$ "$T = h$"



Notación Asintótica

Otros órdenes de complejidad: O(log log *N*)

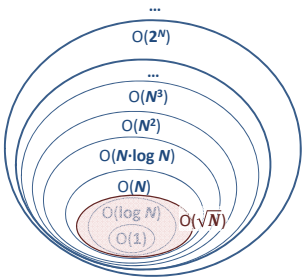
Notación: O $T(N) \in \mathbf{O}(f(N))$ "$T \leq f$" | Ω $T(N) \in \mathbf{\Omega}(g(N))$ "$T \geq g$" | Θ $T(N) \in \mathbf{\Theta}(h(N))$ "$T = h$"



Notación Asintótica

Otros órdenes de complejidad: O($\sqrt{N}$)

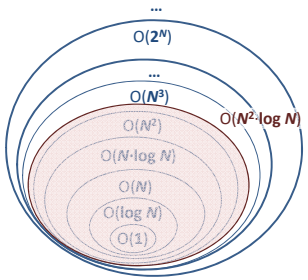
Notación: O $T(N) \in \mathbf{O}(f(N))$ "$T \leq f$" | Ω $T(N) \in \mathbf{\Omega}(g(N))$ "$T \geq g$" | Θ $T(N) \in \mathbf{\Theta}(h(N))$ "$T = h$"



Notación Asintótica

Otros órdenes de complejidad: O($N^2 \cdot \log N$)

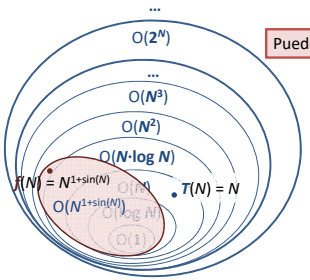
Notación: O $T(N) \in \mathbf{O}(f(N))$ "$T \leq f$" | Ω $T(N) \in \mathbf{\Omega}(g(N))$ "$T \geq g$" | Θ $T(N) \in \mathbf{\Theta}(h(N))$ "$T = h$"



Notación Asintótica

Funciones "raras": Tricotomía

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Puede ocurrir que $T(N) \notin O(f(N))$ y $T(N) \notin \Omega(f(N))$

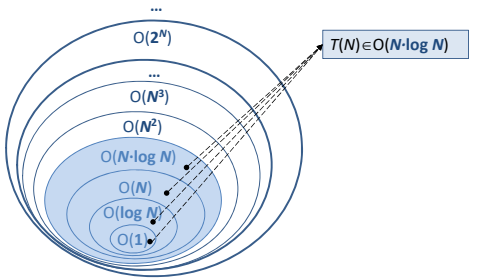
Cuidado

$T(N) = N$
$f(N) = N^{1+\sin(N)}$

Notación Asintótica

Jerarquía de Órdenes

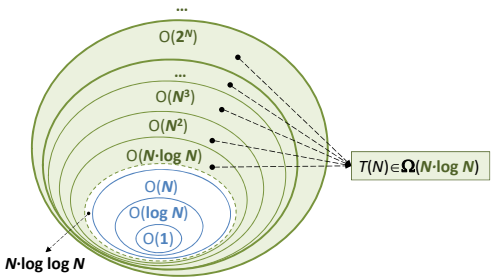
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Jerarquía de Órdenes

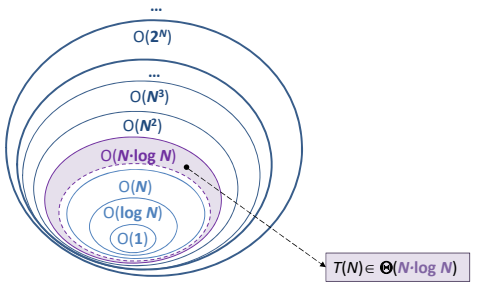
Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"



Notación Asintótica

Jerarquía de Órdenes

Notación: O $T(N) \in O(f(N))$ "$T \leq f$" | Ω $T(N) \in \Omega(g(N))$ "$T \geq g$" | Θ $T(N) \in \Theta(h(N))$ "$T = h$"

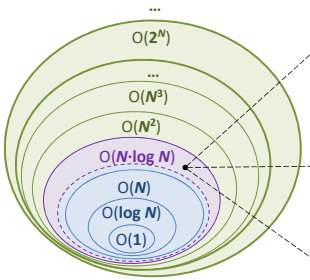


Notación Asintótica

Jerarquía de Órdenes

Notación:

O	$T(N) \in O(f(N))$	"$T \leq f$"
Ω	$T(N) \in \Omega(g(N))$	"$T \geq g$"
Θ	$T(N) \in \Theta(h(N))$	"$T = h$"



$T(N) \in O(N \cdot \log N)$ También es cierto que:
- $T(N) \in O(N^2)$
- $T(N) \in O(N^3)$
- $T(N) \in O(2^N)$

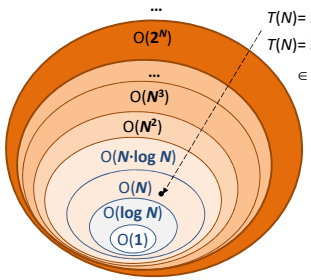
$T(N) \in \Omega(N \cdot \log N)$ También es cierto que:
- $T(N) \in \Omega(N)$
- $T(N) \in \Omega(\log N)$
- $T(N) \in \Omega(1)$

$T(N) \in \Theta(N \cdot \log N)$

Notación Asintótica

Operaciones con conjuntos

Sea a una constante
- $\Theta(a) = \Theta(1)$
- $a \cdot \Theta(f(N)) = \Theta(a \cdot f(N)) = \Theta(f(N))$
- $\Theta(f(N)) + \Theta(g(N)) = \Theta(f(N) + g(N)) = \Theta(\max\{f(N), g(N)\})$
- $\Theta(f(N)) \cdot \Theta(g(N)) = \Theta(f(N) \cdot g(N))$



$T(N) = 21 + 25 \cdot N + 34 \cdot \ln(N)$

$T(N) = \mathbf{21 + 25 \cdot N + 34 \cdot \ln(N)} \in \Theta(1) + \mathbf{25 \cdot N + 34 \cdot \ln(N)}$

$\in \Theta(1) + \Theta(N) + \Theta(\log(N)) \in \Theta(N)$

Notación Asintótica

Actividades

Actividad 2.3. Un algoritmo tiene orden de complejidad **O(*N*)**. Si para un tamaño ***N*=10**, el algoritmo tarda **2 segundos** en nuestro computador.

a) ¿Podemos conocer **cuánto tardará** el algoritmo para un tamaño ***N*=20**?

b) ¿Podemos dar una **cota superior** de cuánto tardará el algoritmo?

Notación Asintótica

Actividades

Actividad 2.4. Considérese dos algoritmos:
- El algoritmo **A1** tiene orden de complejidad $\Theta(N)$.
- El algoritmo **A2** tiene orden de complejidad $\Theta(N^2)$.

¿ El algoritmo **A1** tarda siempre menos que **A2** ?

Notación Asintótica

Actividades

Actividad 2.5. Considérese dos algoritmos:

- El algoritmo **A1** tiene orden de complejidad $\Theta(N)$.
- El algoritmo **A2** tiene orden de complejidad $\Theta(N^2)$.

¿ Existe un tamaño del problema ***N*** a partir del cual el algoritmo **A1** tarda siempre menos que el algoritmo **A2** ?

Notación Asintótica

Actividades

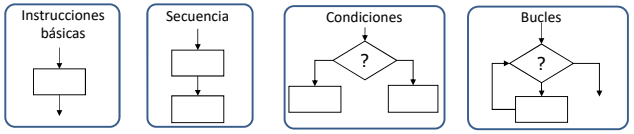
Actividad 2.6. Cualquier algoritmo de ordenación basado en comparaciones tiene un tiempo de complejidad $\Omega(N \cdot \ln N)$ en el caso peor:

a) ¿Es posible diseñar un algoritmo de ordenación basado en comparaciones que tenga orden de complejidad **O(N)** en el **caso peor**?

b) ¿Es posible diseñar un algoritmo de ordenación basado en comparaciones que tenga orden de complejidad **O(N)** en el **caso mejor**?

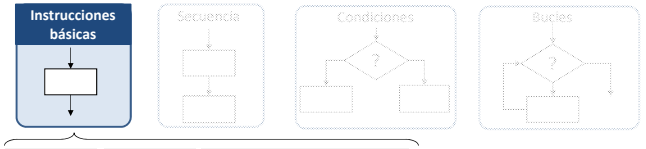
Complejidad en algoritmos sencillos

Instrucciones Sencillas



Complejidad en algoritmos sencillos

Instrucciones Sencillas




```
…
 a = 3;
…
```

```
…
 a = b;
…
```

```
…
return (b+2)*3-7/3;
…
```

…

asignación de variables

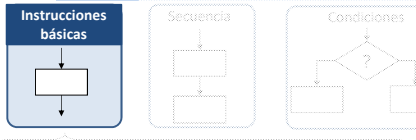
expresiones aritméticas

$T(N)$= constante

$\Theta(1)$

Complejidad en algoritmos sencillos

Instrucciones Sencillas



$T(N)$= constante

$T(N) = \Theta(1)$

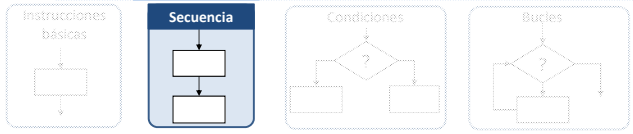
N: Tamaño de `vector[]`

```
int primero(int[] vector) {
   return vector[0];
 }
```

$T(n) \in \Theta(1)$

Complejidad en algoritmos sencillos

Secuencias




```
…
P1;
P2;
…
```

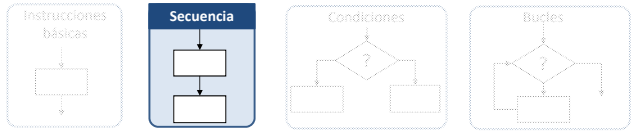
$T_1(N)$, $T_2(N)$ → $T(N)$

$T(N)= T_1(N) + T_2(N)$

$T(N) \in \Theta(\max\{T_1(N), T_2(N)\})$

Complejidad en algoritmos sencillos

Secuencias: Ejemplo




```
…
P1;
P2;
…
```

$T_1(N)$, $T_2(N)$ → $T(N)$

$T(N)= T_1(N) + T_2(N)$

$T(N) = \Theta(\max\{T_1(N), T_2(N)\})$

N: Tamaño de `vector[]`

```
int maxValor(int[] vector) {
   ordenar(vector);          } Θ(N·log N)
   return vector[0];         } Θ(1)
 }
```

$T(n) \in \Theta(N \cdot \log N) + \Theta(1) = \mathbf{\Theta(N \cdot \log N)}$

Complejidad en algoritmos sencillos

Secuencias

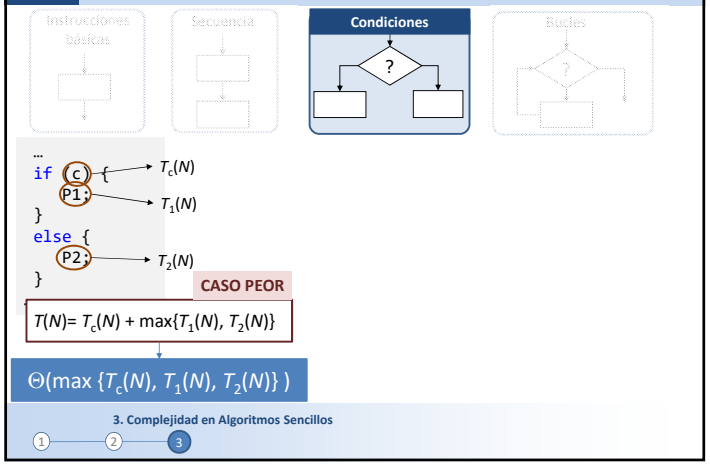
Actividad 3.1. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector.

N: Tamaño de `vector[]`

```
boolean contiene(int vector[], int elemento) {
   ordenar(vector);                          } Θ(N·log N)
   return busquedaOrdenada(vector, elemento);} Θ(log N)
 }
```

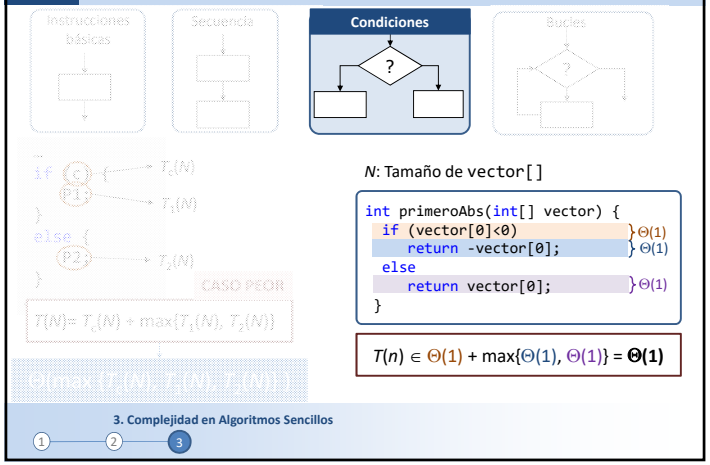
Complejidad en algoritmos sencillos

Condición: `if` …



Complejidad en algoritmos sencillos

Condición: `if` …



Complejidad en algoritmos sencillos

Secuencias

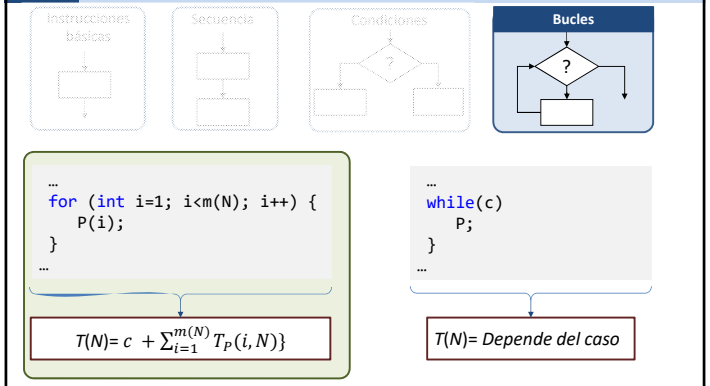
Actividad 3.2. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector.

N: Tamaño de vector[]

```
void borrarElemento(int[] vector, int e) {
  int p = buscar(vector, elemento);}Θ(log N)
  if (p>=0)
    desplazarVector(vector,p);  } Θ(N)
}
```

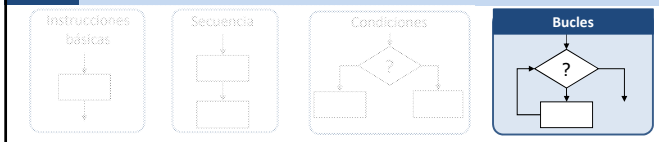

Complejidad en algoritmos sencillos

Bucle: `for`…



Complejidad en algoritmos sencillos

Bucle: for…




```
int factorial(int n) {
   int r=1;
   for (int i=1; i<=n; i++)
       r=r*i;
   return r;
 }
```


Complejidad en algoritmos sencillos

Bucle: for…

Actividad 3.3. Calcula la complejidad del siguiente algoritmo en función de *n*.

```
int func1(int n) {
   int l=0;
   for (int i=1; i<=n; i++)
     for (int j=1; j<=i; j++)
       l++;
   return l;
 }
```

$T(n) \in \Theta(1)+\Theta(n^2)+\Theta(1) = \Theta(n^2)$

Complejidad en algoritmos sencillos

Bucle: for…

Actividad 3.4. Calcula la complejidad del siguiente algoritmo en función de *n*.

```
int func2(int n) {
   int l=0;
   for (int i=1; i<=n; i++)
     for (int j=1; j<=i; j++)
       for (int k=1; k<=n; k++)
         l++;
   return l;
 }
```


Complejidad en algoritmos sencillos

Bucle: for…

Actividad 3.5. Calcula la complejidad del siguiente algoritmo en función de *n*.

```
int func3(int n) {
   int l=0;
   for (int i=0; i<n; i++)
     for (int j=0; j<n*n; j++)
       for (int k=0; k<n*n*n; k++)
         l++;
   return l;
 }
```

Complejidad en algoritmos sencillos

Bucle: for…

Actividad 3.6. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector.

N: Tamaño de vector[]

```
int suma(int[] vector) {
   int l=0;
   for (int i=0; i<vector.length; i++)
     l+=vector[i];
   return l;
 }
```


Complejidad en algoritmos sencillos

Bucle: for…

Actividad 3.7. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector de entrada.

N: Tamaño de vector[]

```
int max(int[] vector) {
   int m=Integer.MIN_VALUE;
   for (int i=0; i<vector.length; i++)
     if (vector[i]>=m) m = vector[i];
   return m;
 }
```


Complejidad en algoritmos sencillos

Bucle: for…

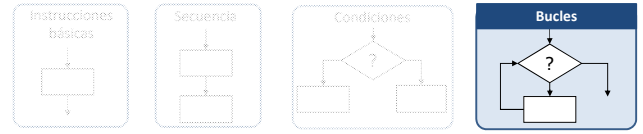
Actividad 3.8. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector de entrada.

N: Tamaño del vector x[]

```
int min_distancia(int[] x) {
   int n=x.length;
   int d=Integer.MAX_VALUE;
   for (int i=0; i<n; i++)
     for (int j=i+1; j<n; j++) {
         int d2 = x[i]-x[j];
         if (d2<0) d2=-d2;
         if (d2<d) d=d2;
      }
   return d;
 }
```


Complejidad en algoritmos sencillos

Bucle: for…




```
…
for (int i=1; i<m(N); i++) {
   P(i);
}
…
```

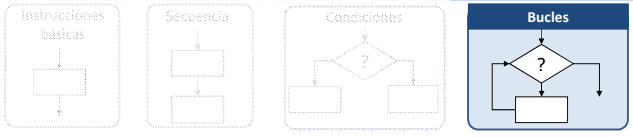
$T(N) = c + \sum_{i=1}^{m(N)} T_P(i, N)\}$

```
…
while(c)
   P;
}
…
```

T(*N*)= *Depende del caso*

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…



Caso Peor | Caso Promedio | Caso Mejor

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;
      while (i<vector.length && !encontrado) {
        if (vector[i] == elemento) encontrado = true;
        i++;
      }
      return encontrado;
}
```

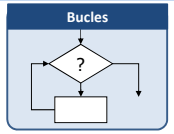
¿ Nº de Iteraciones? — *Dependerá del caso concreto*

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

Caso Peor:
- El *array* **NO** contiene el elemento.



Caso Peor | Caso Promedio | Caso Mejor

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;
      while (i<vector.length && !encontrado) {
        if (vector[i] == elemento) encontrado = true;
        i++;
      }
      return encontrado;
}
```

Se ejecuta N veces

Siempre se cumple: `encontrado: false`

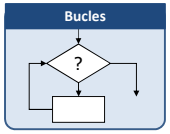
Consideramos que siempre se cumple: `vector[i]!=elemento`

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

Caso Peor:
- El *array* **NO** contiene el elemento.



Θ(*N*)

Caso Peor | Caso Promedio | Caso Mejor

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;        } Θ(1)
      int i = 0;
      while (i<vector.length && !encontrado) {
        if (vector[i] == elemento) encontrado = true;   } Θ(1)
        i++;
      }
      return encontrado;                 } Θ(1)
}
```

Se ejecuta N veces

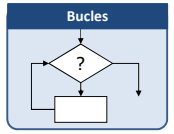
$T(N) \in \Theta(1) + N \cdot \Theta(1) + \Theta(1) = \mathbf{\Theta(N)}$

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

Caso Mejor:
- El *array* contiene el elemento en su primera posición.



Caso Peor | Caso Promedio | **Caso Mejor**

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;
      while (i<vector.length && !encontrado) {
        if (vector[i] == elemento) encontrado = true;
        i++;
      }
      return encontrado;
}
```

Se ejecuta 1 vez

Para i:1 `encontrado:true`

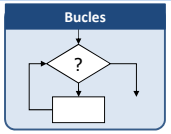
Consideramos que se cumple: `vector[0]==elemento`

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

Caso Mejor:
- El *array* contiene el elemento en su primera posición.



Caso Peor | Caso Promedio | **Caso Mejor** — **Θ(1)**

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;        } Θ(1)
      int i = 0;
      while (i<vector.length && !encontrado) {
Se ejecuta 1 vez  if (vector[i] == elemento) encontrado = true;   } Θ(1)
            i++;
      }
      return encontrado; } Θ(1)
}
```

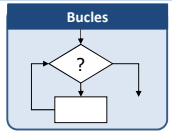
$T(N) \in \Theta(1) + 1 \cdot \Theta(1) + \Theta(1) = \mathbf{\Theta(1)}$

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

¿**Distribución de probabilidad** sobre donde se encuentra el elemento en el vector?
- El elemento tiene la **misma probabilidad** de encontrarse por primera vez en cada uno de los componentes del *array*.



Caso Peor | **Caso Promedio** | Caso Mejor

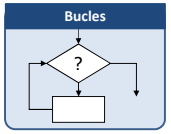
```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;
      while (i<vector.length && !encontrado) {
          if (vector[i] == elemento) encontrado = true;
          i++;
      }
      return encontrado;
}
```

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

¿**Distribución de probabilidad** sobre donde se encuentra el elemento en el vector?
- El elemento tiene la **misma probabilidad** de encontrarse por primera vez en cada uno de los componentes del *array*.



Caso Peor | **Caso Promedio** | Caso Mejor

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;
      while (i<vector.length && !encontrado) {
Se ejecuta N/2 veces  if (vector[i] == elemento) encontrado = true;
            i++;
      }
      return encontrado;
}
```

Para i: *N*/2 `encontrado:true`

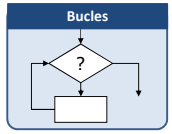
Consideramos que se cumple:
Para i:1…*N*/2-1
`vector[i]!=elemento`
`vector[N/2]==elemento`

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…

¿**Distribución de probabilidad** sobre donde se encuentra el elemento en el vector?
- El elemento tiene la **misma probabilidad** de encontrarse por primera vez en cada uno de los componentes del *array*.



Caso Peor | **Caso Promedio** — **Θ(*N*)** | Caso Mejor

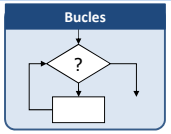
```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;        } Θ(1)
      int i = 0;
      while (i<vector.length && !encontrado) {
Se ejecuta N/2 veces  if (vector[i] == elemento) encontrado = true;   } Θ(1)
            i++;
      }
      return encontrado; } Θ(1)
}
```

$T(N) \in \Theta(1) + N/2 \cdot \Theta(1) + \Theta(1) = \mathbf{\Theta(N)}$

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Ejemplo bucle: `while`…



Θ(*N*) Caso Peor | **Θ(*N*)** Caso Promedio | **Θ(1)** Caso Mejor

```
boolean contieneValor (int[] vector, int elemento){
      boolean encontrado = false;
      int i = 0;                                  } Θ(1)
      while (i<vector.length && !encontrado) {
          if (vector[i] == elemento) encontrado = true;
          i++;
      }
      return encontrado; } Θ(1)
}
```

N:Tamaño del vector (`vector.length`)

Complejidad en algoritmos sencillos

Bucle: `for`…

Actividad 3.9. Calcula la complejidad del siguiente algoritmo en función del tamaño del vector.

```
boolean esta_Ordenado (int[] vector){
      boolean ordenado = true;
      int i = 1;
      while (i<vector.length && ordenado) {
          if (vector[i-1] > vector[i]) ordenado= false;
          i++;
      }
      return ordenado;
}
```

N:Tamaño del vector (`vector.length`)

Actividades Adicionales

Actividad 4.1. Demostrar que si se cumple $f(n) \in \Theta(g(n))$, entonces también se cumple $\Theta(f(n)) = \Theta(g(n))$

Actividad 4.2. Demostrar que $O(f(n)) = O(g(n))$ es equivalente a $\Theta(f(n)) = \Theta(g(n))$

Actividad 4.3. Demostrar que $\Omega(f(n)) = \Omega(g(n))$ es equivalente a $\Theta(f(n)) = \Theta(g(n))$

Actividad 4.4. Considérese dos algoritmos:
- El algoritmo **A1** tiene orden de complejidad $O(N)$.
- El algoritmo **A2** tiene orden de complejidad $O(N^2)$.

¿ Es posible que el algoritmo **A1** tarde siempre más que el algoritmo **A2** ?

Actividades Adicionales

Actividad 4.4. Justifica si es verdadero o falso las siguientes afirmaciones:
- $5^N \in O(2^N)$
- $5^N \in \Omega(2^N)$
- $5^N \in \Theta(2^N)$

Actividad 4.5. Justifica si es verdadero o falso las siguientes afirmaciones:
- $N^2 \in O(N^3)$
- $N^2 \in \Omega(N^3)$
- $2^N \in \Theta(2^{N+1})$
- $(N+1)! \in \Theta(N!)$

Actividades

Actividad 4.6. Calcula la complejidad de los siguientes algoritmos:

```
int funcDistr (int[] vector, int p){
   int suma = 0;
   for (int i = 0; i< vector.length; i++) {
     if (vector[i] <= p) suma+=vector[i];
   }
   return suma;
}
```

```
int funcDistrOrdenado (int[] vector, int p){
      boolean mayor = false;
      int suma = 0;
      int i = 0;
      while (i<vector.length && !mayor) {
          if (vector[i] <= p)
             suma+=vector[i];
          else
             mayor= true;
          i++;
      }
      return suma;
}
```

77 de 97