T
E
M
A

6

Análisis Formal de Algoritmos

Contenido del Tema

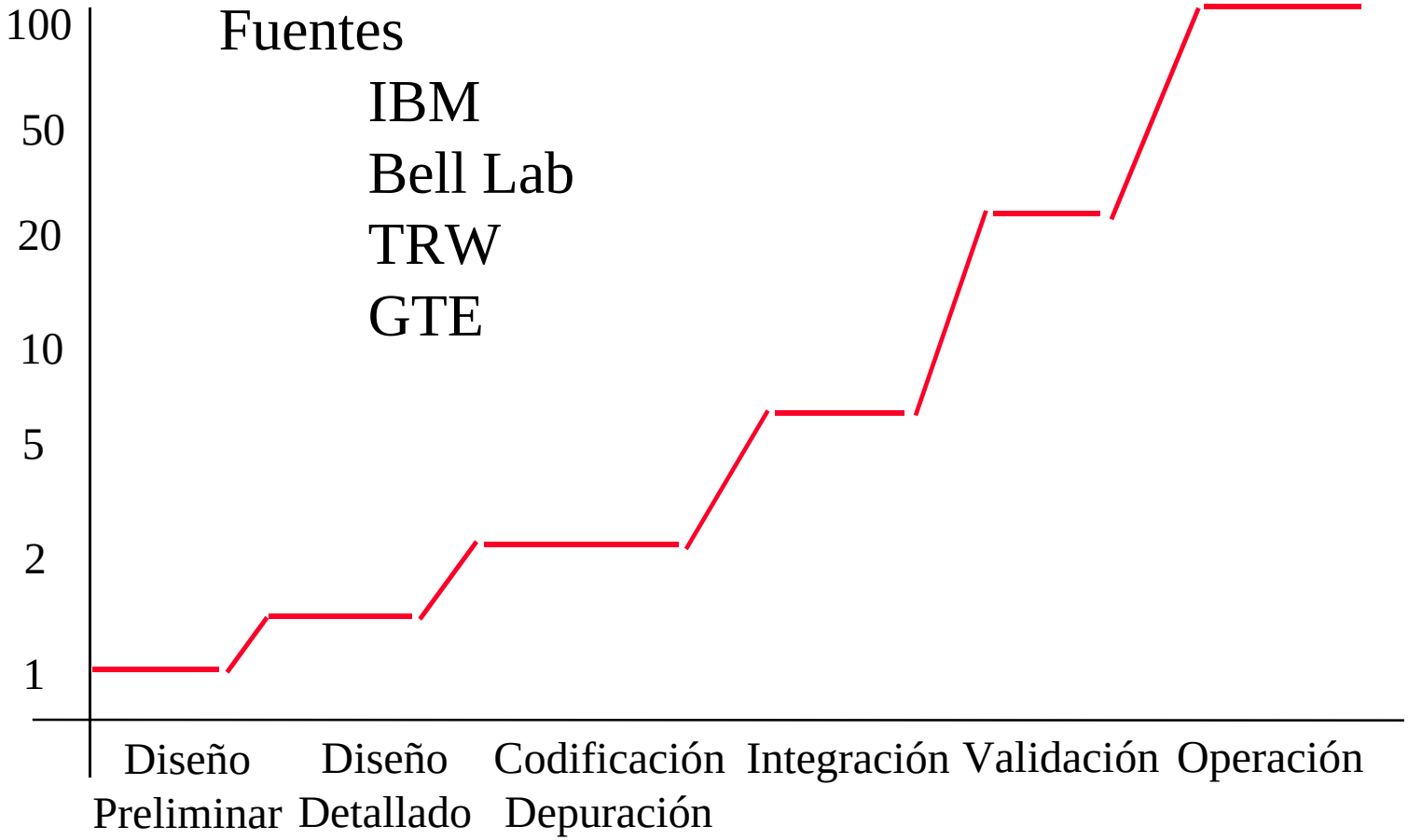
6.1. Verificación Formal

6.2. Análisis de Complejidad

Tipos de Errores

- Un programa puede presentar tres tipos de
  - De Especificación y Diseño: partir de una especificación incorrecta.
  - De Compilación: errores sintácticos en el código.
  - De Ejecución: errores lógicos.

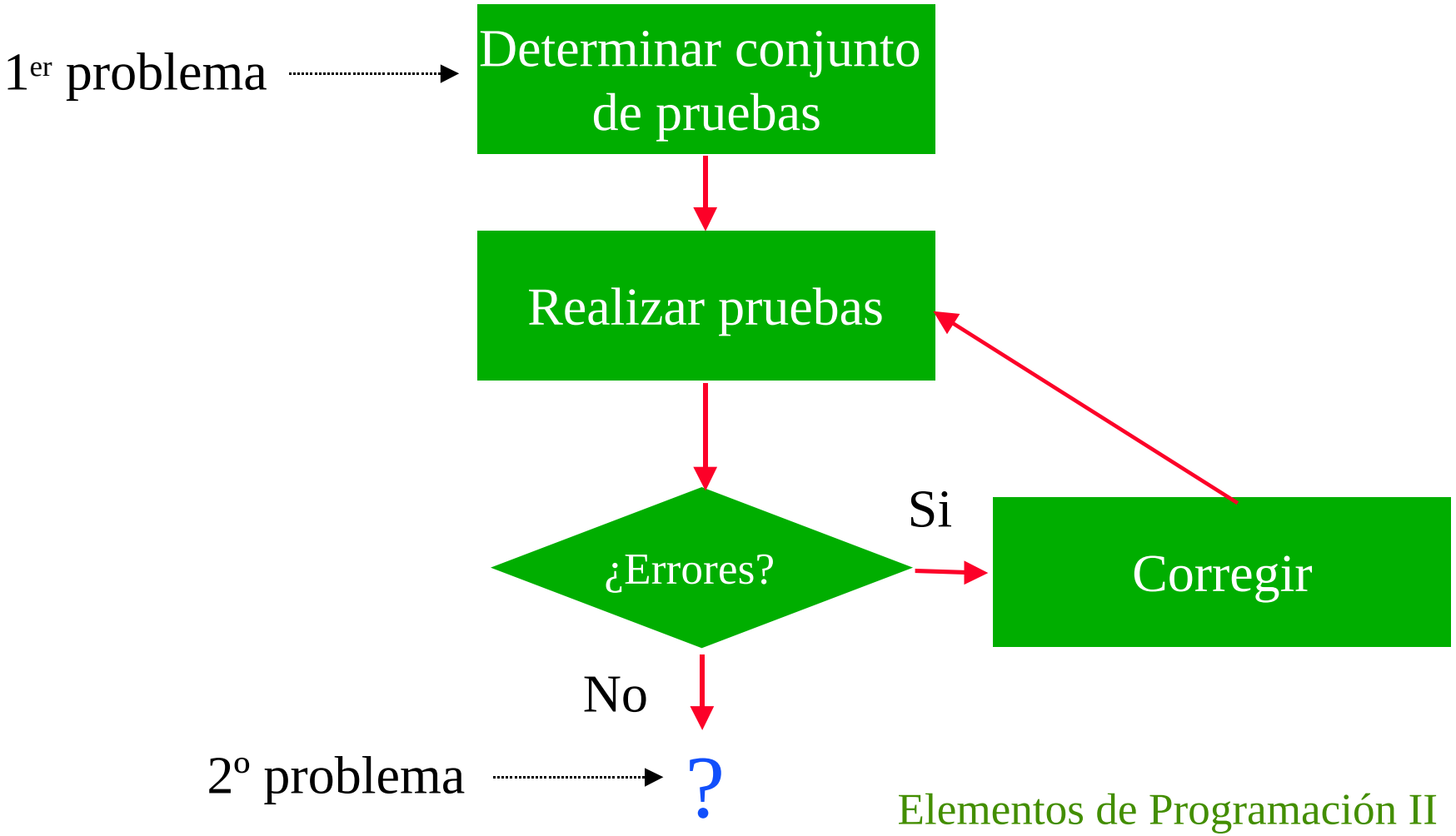
Coste de Corregir un Error



¿Cómo asegurar la corrección?

- Dos enfoques: Depuración y Verificación.
  - Depuración:
    - realizar pruebas experimentales.
    - proceso iterativo.
  - Verificación:
    - método matemático.
    - demostración de teoremas.

Depuración


1er problema
Determinar conjunto de pruebas
Realizar pruebas
¿Errores?
Si
Corregir
No
2º problema
?

Depuración

- Ejemplo: programa que calcula N^2 como la suma de los N primeros impares.

$$N^2 = 1+3+5+...+(2N-1)$$

- Se puede probar con un conjunto de números, pero ¿cómo se sabe que la fórmula no fallará para algún valor no

Depuración

- Solución: emplear un razonamiento matemático.

$$\sum_{i=1}^{N}(2i-1) = 2\sum_{i=1}^{N} i - N = 2\frac{N(N+1)}{2} - N = N^2 + N - N = N^2$$

- Esta solución se enmarca dentro de las técnicas de verificación formal.

Verificación

- Las técnicas de verificación formalmente la corrección de un programa.
- La corrección es una noción ciertas especificaciones, i.e., un programa es correcto si cumple sus especificaciones.
- Hay muchas técnicas de verificación. Vamos a ver la técnica de .

Conceptos previos

- *Estado* de un programa en un instante dado: valor de los objetos que lo componen en

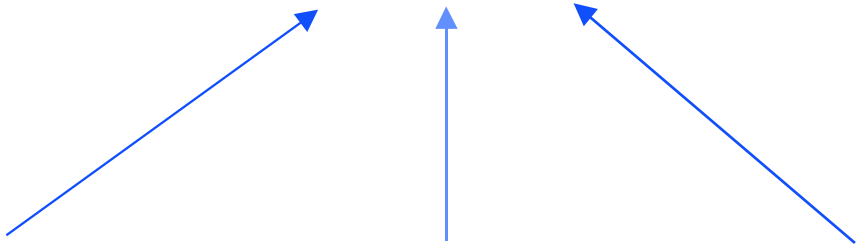
- *Aserto*: afirmación sobre el estado del programa. Puede ser cierta o falsa.
- *Fortaleza* de un aserto: A_1 es un aserto más fuerte que A_2 si, y sólo si, $A_1 \Rightarrow A_2$

Conceptos previos

- Existen tres tipos de asertos:
  - Precondiciones: asertos que deben cumplirse antes de ejecutar el algoritmo para que sea
  - : asertos que expresan el resultado del algoritmo.
  - Asertos intermedios: asertos situados en distintos puntos del algoritmo.

Notación

{P} S {Q}



Precondición Algoritmo Postcondición

Si P se cumple y se ejecuta S, entonces (si S termina) Q se cumple

Verificar un algoritmo consiste en demostrar que cumple su

/

Mecanismo de Verificación

- Realizar un proceso "hacia atrás": Buscar la precondición más debil para el cumplimiento de la postcondición deseada.
- Verificar por separado cada sentencia del programa usando unas reglas de inferencia.
- Emplear la regla de la secuencia:

$$\{P\}S_1\{R\} \wedge \{R\}S_2\{Q\} \Rightarrow \{P\}S_1;S_2\{Q\}$$

Sentencia de Asignación

$$\{Q^{v}_{e}\}\ v := e\ \{Q\}$$

Se substituyen todas las apariciones de *v* por *e*.

- Ejemplo:

$$\{x=5\}\ x := x - 1\ \{x=4\} \equiv \{Q\}$$

$$\{Q^{x}_{x-1}\} \equiv \{x-1=4\} \equiv \{x=5\}$$ Correcto

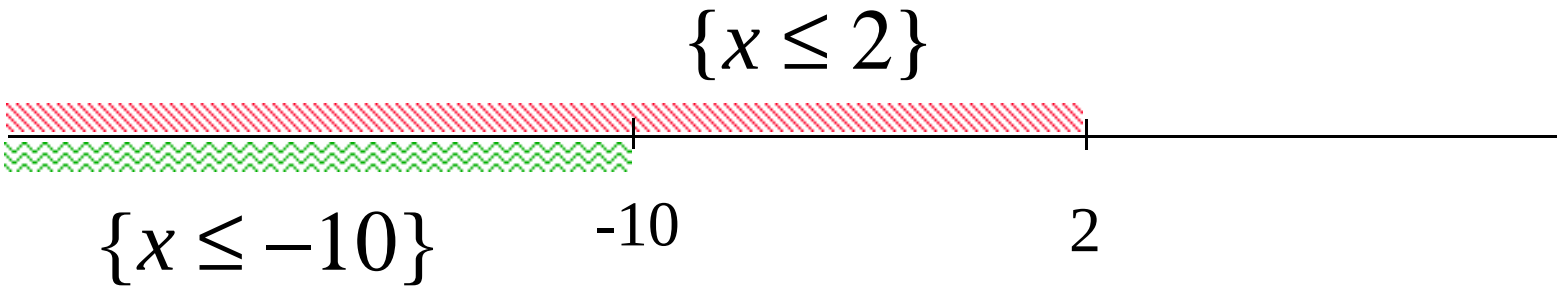
Sentencia de Asignación

- Ejemplo:

$$\{P\} \equiv \{x \leq -10\}\ x = x + 8\ \{x \leq 10\} \equiv \{Q\}$$

$$\{Q^{x}_{x+8}\} \equiv \{x + 8 \leq 10\} \equiv \{x \leq 2\}$$

$$\{x \leq -10\} \Rightarrow \{x \leq 2\}$$



Sentencia de Asignación

- Ejemplo:

$$\{(x=x_0) \wedge (y=y_0)\} \equiv \{P\}$$

$$\{P_3\} \equiv \{(x=x_0) \wedge (y=y_0)\}$$

$t := x$

$$\{P_2\} \equiv \{(t=x_0) \wedge (y=y_0)\}$$

$x := y$

$$\{P_1\} \equiv \{(t=x_0) \wedge (x=y_0)\}$$

$y := t$

$$\{(y=x_0) \wedge (x=y_0)\} \equiv \{Q\}$$

Sentencia de Asignación

- Ejemplos:
  - $\{j = 8\}\ j := j + 1\ \{j = 7\}$
  - $\{D[i] = j\}\ j := j + 1\ \{D[i] = j - 1\}$
  - $\{i = j^k\}$

    $k := k + 1$

    $i := i * j$

    $\{i = j^k\}$

Sentencia de Asignación

- Ejemplos:
  - $\{(x = x_0) \wedge (y = y_0) \wedge (z = z_0)\}$

$$z := x + y + z$$

$$x := z - x - y$$

$$y := z - x - y$$

$$z := z - x - y$$

$\{(x = z_0) \wedge (y = x_0) \wedge (z = y_0)\}$

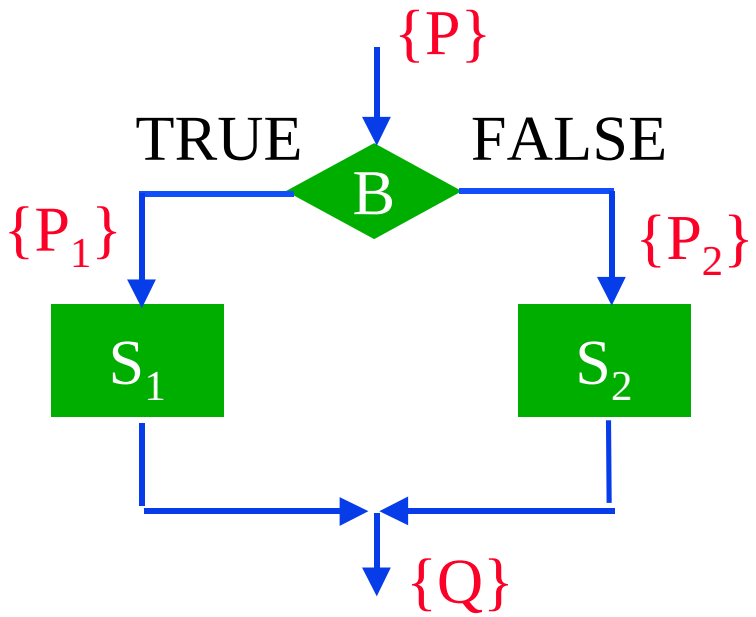
Sentencia de Selección

- Verificar por separado cada rama de la sentencia de selección.

- para cada rama y el valor correspondiente de la condición de control con el valor de la global a la sentencia.

Sentencia de Selección

- Sentencia de selección doble



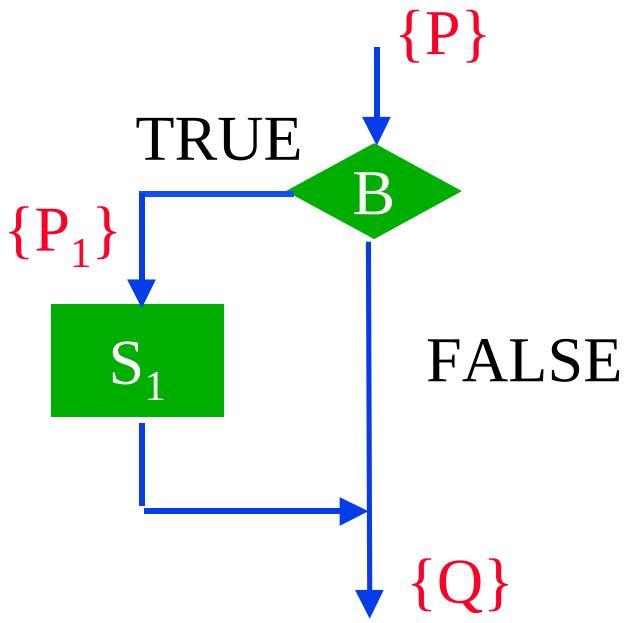
$$\frac{\{P_1\}\ S_1\ \{Q\} \qquad \{P_2\}\ S_2\ \{Q\}}{P \wedge B \Rightarrow P_1 \qquad P \wedge \neg B \Rightarrow P_2}$$

La precondición más débil es por lo tanto

$$(P_1 \wedge B) \vee (P_2 \wedge \neg B)$$

Sentencia de Selección

- Sentencia de selección simple



$$\frac{\{P_1\}\ S_1\ \{Q\}}{P \wedge B \Rightarrow P_1 \qquad P \wedge \neg B \Rightarrow Q}$$

La precondición más débil es por lo tanto

$$(P_1 \wedge B) \vee (Q \wedge \neg B)$$

Sentencia de Selección

- Ejemplo:

$\{(i = 0) \vee (i = 1)\} \equiv \{P\}$

SI i = 1 ENTONCES

$\{P_1\} \equiv \{i = 1\}$

    j := i

EN OTRO CASO

$\{P_2\} \equiv \{i + 1 = 1\} \equiv \{i = 0\}$

    j := i + 1

FINSI

$\{j = 1\} \equiv \{Q\}$

$P \wedge B \equiv [(i=0) \vee (i=1)] \wedge (i=1) \equiv$
$\equiv (i=1) \Rightarrow P_1$

$P \wedge \neg B \equiv [(i=0) \vee (i=1)] \wedge (i \neq 1) \equiv$
$\equiv (i=0) \Rightarrow P_2$

Sentencia de Selección

- Ejemplo:

$\{i = 8\} \equiv \{P\}$

SI (i ≠ 0) ENTONCES

$\{P_1\} \equiv \{0 = 0\} \equiv TRUE$

  i := 0

FINSI

$\{i = 0\}$

Aserto más débil

Aserto más fuerte

$P \wedge \neg B \equiv \{(i=8) \wedge (i=0)\} \equiv FALSE$

Sentencia de Selección

- Ejemplos:

$\{(i = j) \wedge (k = j)\}$

SI i = j ENTONCES

    m := k

EN OTRO CASO

    j := k

FINSI

$\{k = j = m\}$

¿{P}?

SI i > 1 ENTONCES

    i := i + 1

    j := j + i

FINSI

$\{j \leq (i + 1)\}$

Sentencia de Selección

- Ejemplos:

{TRUE}

SI x > y ENTONCES

  max := x

EN OTRO CASO

  max := y

FINSI

{*max* = máximo (*x*, *y*)}

{*i* = 0}

SI i = 0 ENTONCES

  j := 0

EN OTRO CASO

  j := 1

FINSI

{*j* = 1}

Sentencia de Selección

- Ejemplos:

$\{i = (j - 1)\}$

```
SI i > j ENTONCES
   j := j + 1
EN OTRO CASO
   i := i + 1
FINSI
```

$\{i \geq j\}$

$\{x = 3\}$

```
SI impar(x) ENTONCES
   x := x + 1
FINSI
```

$\{par(x)\}$

Sentencia de Selección

- Ejemplos:

$\{[(y \geq z) \wedge (y \geq x)] \vee [(z > y) \wedge (z \geq x)]\}$

SI y ≥ z ENTONCES

    intercambiar (x, y)

EN OTRO CASO

    intercambiar (x, z)

FINSI

$\{(x \geq y) \wedge (x \geq z)\}$

Sentencias Iterativas

- Cualquier sentencia de iteración se puede expresar mediante un bucle .

Sólo se considerará este tipo de bucle.

- Es necesario comprobar que el bucle

- razonamiento inductivo y análisis del invariante.

Razonamiento Inductivo

{P}

MIENTRAS B HACER

C

FINMIENTRAS

{Q}

- Se determinará un conjunto de $\{P_k\}$, donde k es el número de iteraciones realizadas, i.e., $\{P_k\}$ es la precondición más débil que asegura que el bucle termina k iteraciones y alcanza la postcondición.

Razonamiento Inductivo

- $\{P_0\}$ es la precondición para que no se ejecute el bucle y se :

$$P_0 \equiv \neg B \wedge Q$$

- $\{P_k\}$, $k>0$, es la precondición para que el bucle se ejecute k veces y se cumpla la postcondición, es decir, se ejecuta 1 vez y se alcanza la precondición necesaria para que se ejecute $k-1$ veces y se cumpla la postcondición.

$$\{P_k\}$$

$$C$$

$$\{P_{k-1}\}$$

Razonamiento Inductivo

- Ejemplo:

MIENTRAS (i ≠ n) HACER

i := i + 1

s := s + i

FINMIENTRAS

$\{Q\} \equiv \{s = n(n+1)/2\}$

$P_0 \equiv \neg B \wedge Q \equiv (i = n) \wedge [s = n(n+1)/2]$

Razonamiento Inductivo

$\{P_1\}$

i := i + 1

s := s + i $\{R\}$

$\{P_0\} \equiv (i = n) \wedge [s = n(n+1)/2]$

$\{P_2\}$

i := i + 1

s := s + i $\{R\}$

$\{P_1\} \equiv (i = n-1) \wedge [s = n(n-1)/2]$

$\{R\} \equiv (i = n) \wedge [s+n = n(n+1)/2]$

$\equiv (i = n) \wedge [s = n(n-1)/2]$

$\{P_1\} \equiv (i+1 = n) \wedge [s = n(n-1)/2]$

$\equiv (i = n-1) \wedge [s = n(n-1)/2]$

$\{R\} \equiv (i = n-1) \wedge [s+n-1 = n(n-1)/2]$

$\equiv (i = n-1) \wedge [s = (n-1)(n-2)/2]$

$\{P_2\} \equiv (i+1 = n-1) \wedge [s = (n-1)(n-2)/2]$

$\equiv (i = n-2) \wedge [s = (n-1)(n-2)/2]$

Razonamiento Inductivo

$\{P_3\}$

i := i + 1

s := s + i

$\{R\}$

$\{R\} \equiv (i = n–2) \wedge [s+n–2 = (n–1)(n–2)/2]$

$\equiv (i = n–2) \wedge [s = (n–2)(n–3)/2]$

$\{P_2\} \equiv (i = n–2) \wedge [s = (n–1)(n–2)/2]$

$\{P_3\} \equiv (i+1 = n–2) \wedge [s = (n–2)(n–3)/2]$

$\equiv (i = n–3) \wedge [s = (n–2)(n–3)/2]$

Puede verse que la forma general de $\{P_k\}$ es

$\{P_k\} \equiv (i = n–k) \wedge [s = (n–k+1)(n–k)/2]$

Razonamiento Inductivo

- En el caso concreto de

$\{(i=0) \wedge (s=0)\} \equiv \{P\}$

MIENTRAS $(i \neq n)$ HACER

i := i + 1

s := s + i

FINMIENTRAS

es necesario verificar que $\exists k : \{P\} \Rightarrow \{P_k\}$. Puesto que $(i=0)$ sólo se verifica cuando $k=n$, se comprueba si $P \Rightarrow P_n$, lo cual es cierto.

Sentencia de Iteración

- Ejemplos:

$\{(exp=0) \wedge (val=1)\}$

MIENTRAS exp<n HACER

    exp := exp + 1

    val := val * x

FINMIENTRAS

$\{val = x^n\}$

$\{(cont=0) \wedge (fact=1)\}$

MIENTRAS cont<n HACER

    cont := cont + 1

    fact := fact * cont

FINMIENTRAS

$\{fact = n!\}$

Sentencia de Iteración

- Ejemplos:

$\{(i=0) \wedge (s=0)\}$

MIENTRAS i ≠ n HACER

    k := –k

    s := s + k

    i := i + 1

FINMIENTRAS

$\{s = 0\}$

$\{c = x \cdot y\}$

MIENTRAS NOT impar(x) HACER

    y := y * 2

    x := x DIV 2

FINMIENTRAS

$\{c = x \cdot y\}$

Calculabilidad

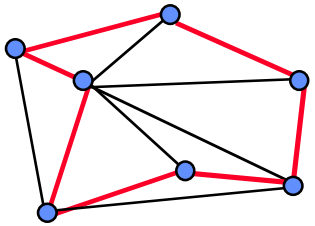
- Problema algorítmicamente resoluble: existe un algoritmo que ante cualquier entrada del problema termina y da la solución correcta.

- Existen problemas no resolubles:
  - Problema de la Parada (*halting problem*)
  - Problema del Castor Afanoso (*busy beaver problem*)

- Teoría de la Calculabilidad: estudia y caracteriza que problemas son algorítmicamente resolubles.

Complejidad

- Existen problemas resolubles pero intratables:
  - Problema del Viajante de Comercio.



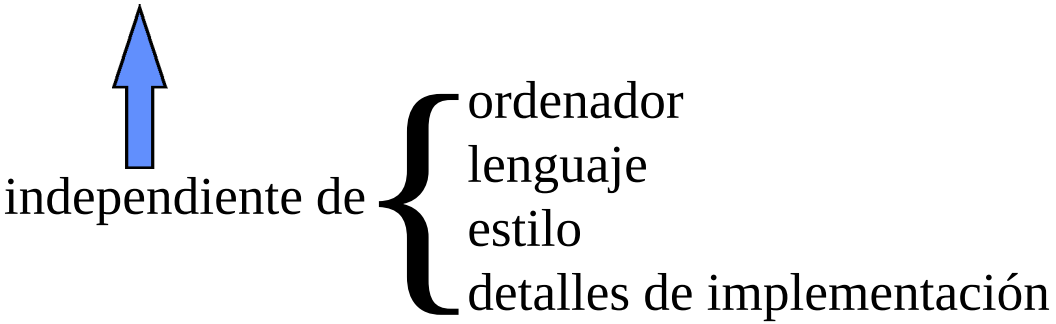
n! caminos diferentes.

En el peor caso hay que examinarlos todos.

- Existen algoritmos resolubles y tratables en
  y .
- La Teoría de la Complejidad estudia estos

Criterios de Eficiencia

- La eficiencia suele estar enfrentada a la claridad, legibilidad y flexibilidad del algoritmo.
- Medidas de eficiencia:
  - Nº de instrucciones en código máquina.
  - Nº de instrucciones en lenguaje de alto nivel.
  - Nº de operaciones básicas.



Operaciones Básicas

- Determinar operación básica:

Problema	Operación Básica
Búsqueda en una lista →	Comparación entre elementos
Multiplicación de matrices →	Multiplicación de valores
Ordenación de una lista →	Comparación entre elementos

- Contar el nº de operaciones básicas que se realizan en el algoritmo

Tiempo de Ejecución

- Factores que influyen en el tiempo de ejecución:
  - de entrada al algoritmo
  - del código compilado
  - Características del
  - operaciones básicas

} afectan proporcionalmente

- El tiempo de ejecución no se expresará en segundos, sino en número de operaciones básicas en relación con el tamaño de los datos de entrada.

Tamaño del Problema

- Factor que determina el número de operaciones básicas realizadas:

Problema		Tamaño del Problema
Búsqueda en una lista	→	Número de elementos
Multiplicación de matrices	→	Dimensión de las matrices
Resolución Ecuaciones	→	Número de ecuaciones e incognitas

- Si *n* es el tamaño del problema, T(*n*) es el número de operaciones básicas realizadas para dicho

Complejidad Computacional

- Ejemplos (operación básica = suma):

```
x := x + y
```
→ $T(n) = 1$

```
PARA i:=1 HASTA n HACER
    x := x + y
FIN PARA
```
→ $T(n) = n$

```
PARA i:=1 HASTA n HACER
    PARA j:=1 HASTA n HACER
        x := x + y
    FIN PARA
FIN PARA
```
→ $T(n) = n^2$

Análisis de Complejidad

- T no depende únicamente de *n*.
  - Ejemplo: búsqueda secuencial
- Se consideran dos medidas:
  - P(*n*): peor caso para tamaño *n*.
    - $P(n) = max\ \{T(I) : I\ \hat{I}\ D_n\}$
  - M(*n*): caso medio para tamaño *n*.
    - $M(n) = \sum_{I \in D_n} p(I) \cdot T(I)$

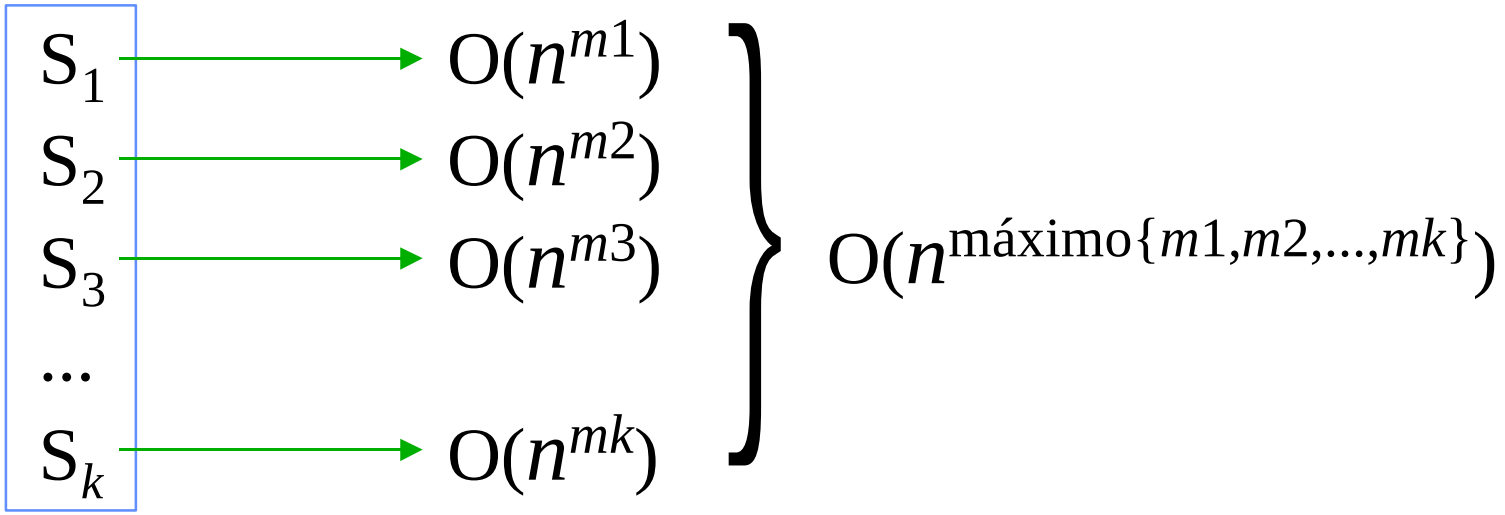
Notación Asintótica

- $f(n) = \Omega(g(n))$: orden inferior
  - $\exists c, n_0 : |g(n)| \leq c \cdot |f(n)|, \forall n \geq n_0$
- $f(n) = O(g(n))$: orden superior
  - $\exists c, n_0 : |f(n)| \leq c \cdot |g(n)|, \forall n \geq n_0$
- $f(n) = \Theta(g(n))$: orden exacto
  - $\exists c_1, c_2, n_0 : c_1 \cdot |g(n)| \leq |f(n)| \leq c_2 \cdot |g(n)|, \forall n \geq n_0$

Cálculo de la Complejidad

TEOREMA: Si $A(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_0$, entonces $A(n) = O(n^m)$.

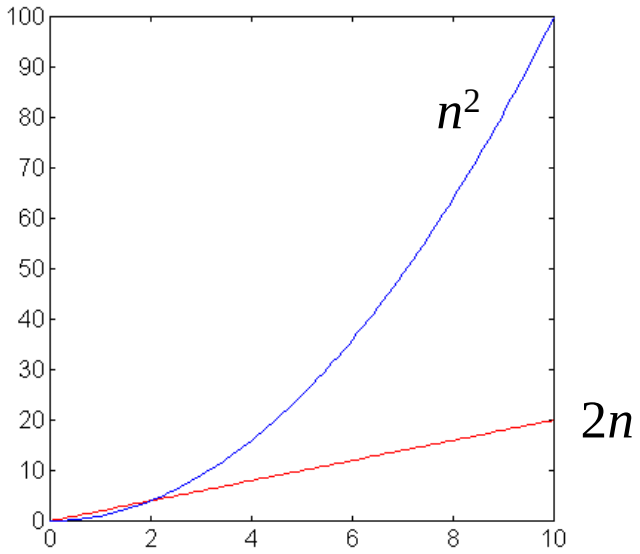
COROLARIO:



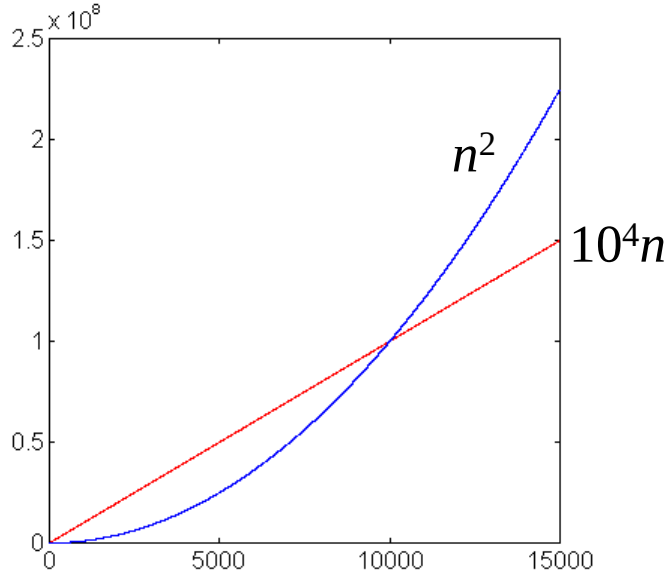
Comparación entre Algoritmos

- ¿Es mejor O(n) que O(n^2)?

$2n$ vs n^2



10^4n vs n^2



$O(1) < O(\log n) < O(n) < O(n \cdot \log n) < O(n^2) < \ldots < O(2^n)$

Cálculo de Complejidad

- Ejemplos:
  - Buscar máximo valor de un vector:
    - $O(n)$
    - $\Omega(n)$

    } → $\Theta(n) = n$ (siempre hay que recorrer el vector completo)
  - Búsqueda secuencial:
    - $O(n)$ → (peor caso: el elemento no está)
    - $\Omega(1)$ → (mejor caso: el elemento está en la 1ª posición)

Cálculo de Complejidad

- Ejemplos:
  - Búsqueda binaria:
    - $O(\log n)$ → (En cada paso se divide el array por la mitad. Esta división se puede hacer como máximo $\log n$ veces)
    - $\Omega(1)$ → (mejor caso: el elemento se encuentra a la 1ª)
  - Ordenación por intercambio (burbuja):
    - $$\Theta(n) = \sum_{i=1}^{n-1}(n-i) = n\cdot(n-1) - \sum_{i=1}^{n-1} i = n^2 - n - \frac{n^2-n}{2} =$$

$$= \frac{n^2-n}{2} = \Theta(n^2)$$

Complejidad y Recursión

- Programas recursivos ➡ Ecuaciones Recurrentes
- Ejemplo:

```
Algoritmo Factorial (n: N)
Inicio
        SI n = 0 ENTONCES DEVOLVER 1
        EN OTRO CASO DEVOLVER n*Factorial(n-1)
Fin
```

Operación Básica

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

Complejidad y Recursión

$T(n) = 1 + T(n–1) = 1 + 1 + T(n–2) = 1 + 1 + 1 + T(n–3) =$

$= 1+...^{k}...+1 + T(n–k)$ ➡ caso base cuando $k = n$ ➡ $T(n) = n$

- Ejemplo: Torres de Hanoi (operación básica = mover disco)

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

$T(n) = 2T(n–1) + 1 = 2^2T(n–2) + 2 + 1 = 2^3T(n–3) + 2^2 + 2 + 1 =$

$= 2^kT(n–k) + 2^{k-1} + 2^{k-2} + ... + 2^2 + 2 + 1$ ➡ caso base cuando $k=n-1$

⬇

$$T(n) = 2^{n-1} + 2^{n-2} + ... + 2^2 + 2 + 1 = 2^n–1 = O(2^n)$$

Complejidad y Recursión

- Ejercicios Propuestos:
  - Fibonacci iterativo
  - Fibonacci recursivo
  - Quicksort
  - Mergesort

Bibliografía

- Aho A., Hopcroft J.E., Ullman J.D. (1988), *Estructuras de Datos y Algoritmos*, Addison-Wesley Iberoamericana
- Backhouse R. (1986), *Program Construction and Verification*, Prentice Hall International,Series in Computer Science
- Brassard G., Bratley P. (1988), *Algorithmics: Theory and Practice*, Prentice Hall International,Series in Computer Science