

1) Algoritmo correcto (correctez)

- 1) tarda tiempo finito (ciclos finitos)
- 2) correctez (invariante ciclo)
 - 1) Inicialización (cierto antes de ejecutar primera iteración)
 - 2) Mantenimiento (cierto antes de iteración, cierto en la siguiente)
 - 3) Terminación (es invariante al finalizar el ciclo)

Ordenar objetos : Insertion Sort
• Eficiente n pequeña

Problema de búsqueda

Recursos:

- Instrucciones simples
- Cada instrucción con tiempo constante
- Secuencia
- Unico procesador
- Sin concurrencia

Tiempo computacional : RAM'

Depende de:

+ Entrada (tamaño)

+ forma y caract. de la entrada

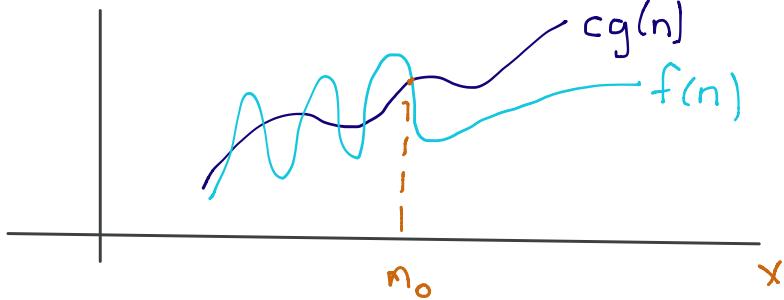
Tiempo de ejecución está dado por el número de pasos primitivos que ejecuta en una entrada particular.

- Mejor caso
- Caso promedio
- Peor caso

Clase 4 de Septiembre

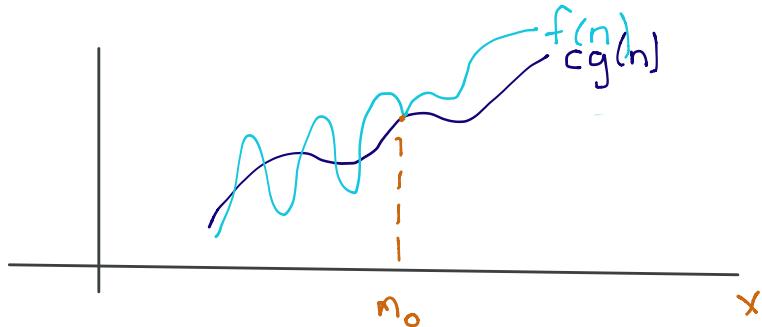
Cotas asintóticas

$$\Theta(g(n)) = \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ s.t. } 0 \leq f(n) \leq cg(n) \forall n \geq n_0 \}$$



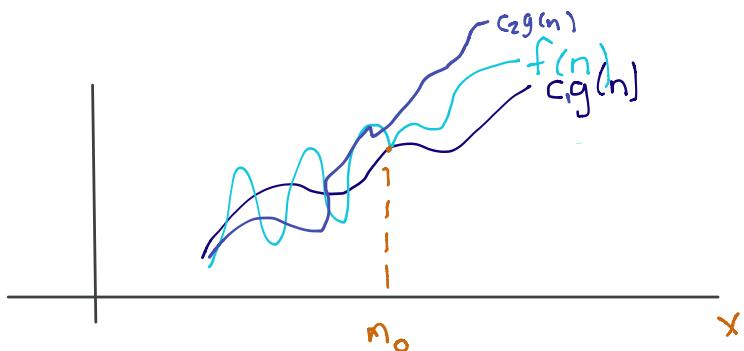
$f(n) \in \Theta(g(n))$
 $g(n)$ es cota superior asintótica de $f(n)$

$$\Omega(g(n)) = \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c > 0, n_0 \in \mathbb{N} \text{ s.t. } cg(n) \leq f(n) \forall n \geq n_0 \}$$



$f(n) \in \Omega(g(n))$
 $g(n)$ es cota inferior asintótica de $f(n)$

$$\Theta(g(n)) = \{ f: \mathbb{N} \rightarrow \mathbb{N} \mid \exists c_1, c_2 > 0, n_0 \in \mathbb{N} \text{ s.t. } c_1 g(n) \leq f(n) \leq c_2 g(n) \forall n \geq n_0 \}$$



$f(n) \in \Theta(g(n))$
 $g(n)$ es cota asintótica de $f(n)$

Ejemplos

1) $f(n) = \frac{1}{2}n^2 - 3n$ Mostrar que $f(n) = \Theta(n^2)$

Tenemos que encontrar $c_1, c_2 > 0$ y $n_0 \in \mathbb{N}$ s.t.

$$\begin{aligned} C_1 n^2 &\leq \frac{1}{2}n^2 - 3n \leq C_2 n^2 \\ C_1 &\leq \frac{1}{2} - \frac{3}{n} \leq C_2 \quad n > 0 \end{aligned}$$

$$C_1 \leq \frac{1}{2} - \frac{3}{n} \quad \text{cierto si} \quad n \geq 7, \quad C_1 \leq \frac{1}{4}$$

$$\frac{1}{2} - \frac{3}{n} \leq C_2 \quad \text{cierto si} \quad n \geq 1, \quad C_2 \geq \frac{1}{2}$$

$$\therefore \text{es cierto si } n \geq 7, \quad C_1 \leq \frac{1}{4}, \quad C_2 \geq \frac{1}{2}$$

2) $f(n) = 6n^3$ Mostrar que $f(n) \neq \Theta(n^2)$

Por contradicción $f(n) = \Theta(n^2)$

$$\exists c_1, c_2 > 0, \quad n_0 \in \mathbb{N} \quad \text{s.t.} \quad c_1 n^2 \leq f(n) \leq c_2 n^2$$

De la segunda desigualdad:

$$6n^3 \leq c_2 n^2 \quad \forall n \geq n_0$$

$$6n \leq c_2$$

$$n \leq \frac{c_2}{6} \quad \forall n \geq n_0 \quad \Rightarrow n_0 \text{ puede ser cierto}$$

3) $p(n) = \sum_{i=0}^d a_i n^i \quad a_0 > 0 \quad p(n) = \Theta(n^d)$

4) $p(n) = an + b \quad a > 0 \quad p(n) = \Theta(n^2)$

$$n_0 = 1 \quad c = a + |b|$$

5) otros

$\checkmark 4n^2 + \Theta(n) = \Theta(n^2) \Rightarrow 4n^2 + (an + b) = \Theta(n^2)$

$\checkmark 4n^2 + \underline{\underline{\Theta(n)}} = \Theta(n^2)$

$\checkmark 4n^2 + c = \Theta(n^2)$

$\checkmark 4n^2 + \log n = \Theta(n^2)$

$$\begin{aligned} &\times \log(n) + \lceil n \rceil = \Theta(n^2) \\ &\times \sum_{i=1}^n \Theta(i) = \Theta(1) + \Theta(2) + \dots + \Theta(n) = \Theta(n) \left. \begin{array}{l} \text{mal} \\ \text{sep.} \end{array} \right\} \\ &\checkmark \sum_{i=1}^n i = \frac{n(n+1)}{2} = \Theta(n^2) \end{aligned}$$

Invariante para ciclo while interno de Insertion Sort

```
i=j-1
while i ≥ 0 and A[i] > llave do
    A[i+1] = A[i]
    i = i - 1
end while
```

i = j-1 0 ≤ i ≤ j-1

Aqui la variable decremente

Encontrar la Invariante:

$A[0, \dots, j-1]$ esta ordenado

$A[0, \dots, i+1] \cancel{\leq} \text{llave} \leq A[i+2, \dots, j-1]$

⇒ no es invariante

$A[i] \cancel{\leq} \text{llave} \leq A[i+1]$

⇒ sobra primera desigualdad

$\boxed{\text{llave} \leq A[i+1]}$

⇒ esta funciona como
invariante

Clase II Septiembre

Tarea: Ejercicio 1:

Ejercicio 2: BubbleSort \rightarrow swap $\Theta(1)$

\rightarrow correctez con invariante de ciclo
para ambos ciclos

Ejercicio 3: Evaluar un polinomio \rightarrow Pseudocódigo Evaluar-Ingenuo

\rightarrow Correctez evaluar Ingenuo

Ejercicio 4: Implementar Regla Horner

Correctez de RH

Tipos de algoritmo:

Incremental: Construye para una parte pequeña de la entrada.

p.ejemplo: insertion sort

Dividir y Conquistar: En algoritmos recursivos:

1) Dividir: Se divide el problema total en subproblema

2) Conquistar: Se lleva a caso base

3) Combinar: subproblema para el original

p.ejemplo: merge-sort

Prog. Dinámica: Similar a D&C. Pueden tener intersección los subproblema

Algoritmos greedy: Busca la mejor solución con minimo esfuerzo por ejemplo local.

Dividir y conquistar: Problema del subarraylo máximo

entrada: arreglo numérico A $n \geq 1$

Salida: tupla (i, j, s) tal que $s = \text{sum } A[i, \dots, j]$ máxima entre
todas las sumas de todos los pos. subarrays

ejemplo [compras venta de bienes]

Fuerza Bruta:

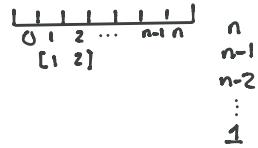
- 1) Probar uno por uno todos los posibles subarreglos de A y calcular los indices del arreglo más grande

$$\binom{n}{2} = \frac{n(n-1)}{2} = \Theta(n^2)$$

- 2) Suma de arreglo

$$\Theta(j-i+1) = \Theta(n)$$

- 3) Total $\sum (n^2) \leq T(n) = \Theta(n^3)$



Dividir y conquistar: Subarreglo

Dividir y Conquistar: 1) Calcular imedio A

Mismo prob.

2) A [0, imedio]

Mismo prob

3) A [imedio+1, A.length-1]

No es el mismo

El problema del subarreglo
máximo que cruce el
punto medio

4) Que cruce imedio.

5) Comparar 3 subarreglos máximos.

Clase 18 Septiembre

Dividir y conquistar • Problema del subarraylo maximo

Pseudocódigo y complejidad

FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$)

```

1   left-sum = -∞
2   sum = 0
3   for i = mid downto low
4       sum = sum + A[i]
5       if sum > left-sum
6           left-sum = sum
7           max-left = i
8   right-sum = -∞
9   sum = 0
10  for j = mid + 1 to high
11      sum = sum + A[j]
12      if sum > right-sum
13          right-sum = sum
14          max-right = j
15  return (max-left, max-right, left-sum + right-sum)

```

If the subarray $A[low..high]$ contains n entries (so that $n = high - low + 1$), we claim that the call FIND-MAX-CROSSING-SUBARRAY($A, low, mid, high$) takes $\Theta(n)$ time. Since each iteration of each of the two for loops takes $\Theta(1)$ time, we just need to count up how many iterations there are altogether. The for loop of lines 3–7 makes $mid - low + 1$ iterations, and the for loop of lines 10–14 makes $high - mid$ iterations, and so the total number of iterations is

$$(mid - low + 1) + (high - mid) = high - low + 1 = n.$$

FIND-MAXIMUM-SUBARRAY($A, low, high$)

```

1  if high == low
2  return (low, high, A[low])           // base case: only one element
3  else mid = ⌊(low + high)/2⌋
4  (left-low, left-high, left-sum) =
   FIND-MAXIMUM-SUBARRAY(A, low, mid)
5  (right-low, right-high, right-sum) =
   FIND-MAXIMUM-SUBARRAY(A, mid + 1, high)
6  (cross-low, cross-high, cross-sum) =
   FIND-MAX-CROSSING-SUBARRAY(A, low, mid, high)
7  if left-sum ≥ right-sum and left-sum ≥ cross-sum
8      return (left-low, left-high, left-sum)
9  elseif right-sum ≥ left-sum and right-sum ≥ cross-sum
10   return (right-low, right-high, right-sum)
11  else return (cross-low, cross-high, cross-sum)

```

The recursive case occurs when $n > 1$. Lines 1 and 3 take constant time. Each of the subproblems solved in lines 4 and 5 is on a subarray of $n/2$ elements (our assumption that the original problem size is a power of 2 ensures that $n/2$ is an integer), and so we spend $T(n/2)$ time solving each of them. Because we have to solve two subproblems—for the left subarray and for the right subarray—the contribution to the running time from lines 4 and 5 comes to $2T(n/2)$. As we have

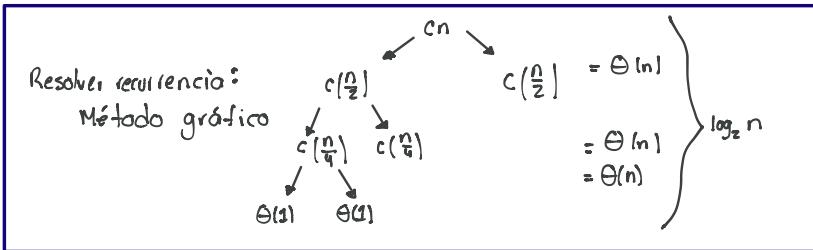
time. Lines 7–11 take only $\Theta(1)$ time. For the recursive case, therefore, we have

$$\begin{aligned} T(n) &= \Theta(1) + 2T(n/2) + \Theta(n) + \Theta(1) \\ &= 2T(n/2) + \Theta(n). \end{aligned} \quad (4.6)$$

Combining equations (4.5) and (4.6) gives us a recurrence for the running time $T(n)$ of FIND-MAXIMUM-SUBARRAY:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1. \end{cases} \quad (4.7)$$

$$T(n) = \Theta(n \lg n).$$



Cada caso base es $\Theta(1)$ $\Rightarrow \Theta(n \log n)$ complejidad del algoritmo

Por inducción probar que $T(n) = \Theta(n \log n)$

Hay una solución lineal $\Theta(n)$ sin DC.

Es muy común $T(n) = aT(n/b) + f(n)$
Dividir y combinar

Analisis de correctez

Prueba de correctez para find max crossing subarray

Invariante de ciclo: es suma maxima

1) Caso base: A tiene un elemento, regresa $(0, 0, A[0])$

2) Supongamos es valido para men Probar para n

Clase 25 septiembre

Problemas, medida y complejidad:

Arreglo máxi:

$$\bullet \Theta(n \log n)$$

Merge sort:

$$\bullet \Theta(n \log n)$$

Insertion sort:

$$\bullet \Theta(n^2)$$

En n pequeñas Insertion Sort

Strassen's algorithm

$$\bullet O(n^{2.81})$$

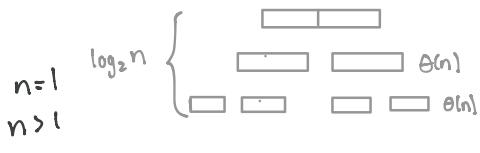
le gana a Merge sort

Square-matrix Multiply (A,B)

Recursividad

Merge Sort:

$$T(n) = \begin{cases} \Theta(1) & n=1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n) & n>1 \end{cases}$$



Versión recursiva de Búsqueda lineal

$$T(n) = T(n-1) + \Theta(1)$$

$$T(n) = \Theta(n)$$
 Revisar

Si $a[n-1] = v$ Termino

Si no, `busquedaLineal[A, n-1, v]`

Multiplicación de matriz

Inequalities in recurrences:

Upper bound on $T(n)$: $O(n)$ $T(n) \leq 2T\left(\frac{n}{2}\right) + \Theta(n)$

Lower bound on $T(n)$: $\Omega(n)$ $T(n) \geq 2T\left(\frac{n}{2}\right) + \Theta(n)$

Métodos para resolver: $T(n) = aT(\frac{n}{b}) + f(n)$

- $\bullet T(n)$ is constant for small n
- \bullet omit floor, ceiling and boundary

• Por sustitución (inducción)

- $\bullet T(n) = 2T(\frac{n}{2}) + n$

Sup. cierto para $m < n$ P.D para n :

$$\leq 2T\left(\frac{n}{2}\right) + n \quad c > 0$$

$$\leq 2\left(C_2 \frac{n}{2} \log \frac{n}{2}\right) + n$$

$$= C_2 n \log \frac{n}{2} + n$$

$$c > 1$$

$$n > 0$$

$$= C_2 n \log n - C_2 n \log 2 + n$$

$$= C_2 n \log n - \underline{(C_2 - 1)n}$$

$$\leq Cn \log n$$

- $\bullet T(n) = T(n-1) + n$

$$T(n) = \Theta(n)$$

Sup cierto para $m < n$

$$T(n) = T(n-1) + n$$

$$\leq C(n-1) + n$$

No funciona

$$= Cn - C + n$$

$$= n(C+1) - C$$

$$\leq n(C+1)$$

- $\bullet T(n) = T(n-1) + n^2$

$$T(n) = \Theta(n^2)$$

Sup cierto $m < n \Rightarrow$

$$\leq C(n-1)^2 + n$$

$$= C(n^2 - 2n + 1) + n$$

Funciona

$$= Cn^2 - 2Cn + C + n$$

$$= Cn^2 \cdot (2C-1)n + C$$

$$= \Theta(n^2)$$

Teorema maestro

$a \geq 1$ $b \geq 1$ constant $f(n)$ función $T(n)$ on the nonnegative integers

$$T(n) = aT(\frac{n}{b}) + f(n)$$

$\frac{n}{b}$ either $\lfloor \frac{n}{b} \rfloor$, $\lceil \frac{n}{b} \rceil$ $T(n)$ has:

• if $f(n) = O(n^{\log_b a - \epsilon})$ for $\epsilon > 0$ Then $T(n) = \Theta(n^{\log_b a})$

• if $f(n) = \Theta(n^{\log_b a})$ then $T(n) = \Theta(n^{\log_b a} \log n)$

• if $f(n) = \Omega(n^{\log_b a + \epsilon})$ $\epsilon > 0$ if $\underline{af(\frac{n}{b}) \leq cf(n) < 1}$ n large $T(n) = \Theta(f(n))$

Regularidad

$$T(n) = 2T(\frac{n}{2}) + \Theta(n)$$

$$\log_b a = \log_2 2 = 1$$

$$f(n) = \Theta(n) \quad n^{\log_b a} = n$$

$$\text{aplicamos caso 2: } T(n) = \Theta(n \log n)$$

$$T(n) = 8T(\frac{n}{2}) + \Theta(n^2)$$

$$a=8 \quad b=2 \quad f(n) = \Theta(n^2)$$

$$\log_2 8 = 3 \Rightarrow n^{\log_2 8} = n^3$$

Caso 3 $\Omega(n^{3+\epsilon}) \Rightarrow$ no aplica

Caso 1 $O(n^{3-\epsilon}) \Rightarrow$ aplica

$$T(n) = 7T(\frac{n}{2}) + \Theta(n^2)$$

$$\log_b a = \log_2 7 \approx 2.81$$

$$n^{\log_b a} \approx n^{2.81} \quad f(n) = \Theta(n^{2.81-\epsilon}) \quad \epsilon = .8$$

$$T(n) = \Theta(n^{2.81})$$

Clase 2 octubre

Programación dinámica

Resuelve los problemas al dividirlos en subproblemas

- No son ajenos (comparten)
- Trade-off Tiempo - Memoria

Multiplicación matrices (Modelo básico)

$$(A_1(A_2A_3)) = 32,000 \text{ M}$$

$A_1 \ 100 \times 20$

$p=100, 20, 20, 20$

$$((A_1A_2)A_3) = 16,000 \text{ M}$$

$A_2 \ 20 \times 20$

$A_3 \ 20 \times 200$

Entrada: Enteros $\langle p_0, \dots, p_n \rangle$ con seq $\langle A_0, \dots, A_n \rangle$

Salida: Asociar matrices

¿Cuántas formas podemos poner paréntesis?

i) $P(n) = \# \text{ formas posibles}$

ii) Calcular costo.

$$n=1 \quad P(1)=1$$

$n > 1$ encierra k matrices A_0, \dots, A_{n-1}

$$P(n) = \begin{cases} 1 & n=1 \\ \sum_{k=1}^{n-1} P(k) P(n-k) & n > 1 \end{cases}$$

$$P(n) = \Omega(2^n) \text{ exponencial}$$

Opción asintótica de $P(n)$

$$P(n) = \Omega(2^n)$$

Por sustitución

$$\Rightarrow P(n) = \Omega(2^n)$$

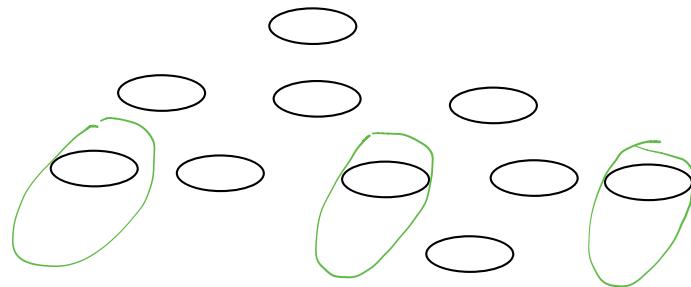
$$n=1 \quad \checkmark$$

$$n > 1$$

$$\begin{aligned} P(n) &\geq \sum_{k=1}^{n-1} P(k) P(n-k) \\ &\geq \sum_{k=1}^{n-1} (C2^k)(C2^{n-k}) \\ &= C^2 \sum 2^{k+n-k} \\ &= C^2 \sum 2^n = C^2 2^n (n-1) \\ &= C^2 2^n (n-1) \end{aligned}$$

$$\geq C2^n$$

$$C \geq 1 \quad n > 1$$



Subproblema en común

$$\begin{array}{c} i=1 \quad \checkmark \\ i>1 \quad A_i \dots A_j \left\{ \begin{array}{l} A_i \dots A_k \\ A_{k+1} \dots A_j \end{array} \right. \quad i \leq k \leq j \end{array}$$

¿Suponiendo que solución para $A_{i \dots k}$ es óptima y la de $A_{k+1 \dots j}$ es óptima ¿Estas componen una solución óptima para $A_{i \dots j}$?

- Suponemos que hay una solución óptima para $A_{i \dots j}$ que usa otra forma de poner paréntesis a $A_{i \dots k}$

Clase 16 octubre

P.D. Paso 2: Ecuación de costo de soluciones

$$C: s \rightarrow t \quad c: s \rightarrow v_n \rightarrow +$$

$$w(s,t) = \begin{cases} 0 & s=t \\ p(\vec{s}t) & \vec{s}t \in A(p) \\ w(s, v_n) + w(v_n, t) & v_n \text{ en } C \end{cases}$$

Suponemos que conocemos v_n

$$\begin{array}{ll} \vec{s} \xrightarrow{\quad} t & \vec{s}t \in A(0) \\ \max \{ w(s, v_n) + w(v_n, t) \} & v_n \in M(p) \end{array}$$

Si $w(v, u) = -\infty$ si no hay camino de v a u

Camino simple más largo : PD Paso 3

- Correctez
- Complejidad

$$\begin{aligned} T(n) &\leq (n-1)T(n-1) + \Theta(1) & f(n) = \Theta(1) \\ &\leq (n-1)(n-2)\dots(n-2) \\ &\leq \Theta(n!) & \Rightarrow \text{con sustitución se puede} \\ && \text{demostrar.} \\ && \Rightarrow \text{no eficiente (se tarda mucho)} \\ && \text{por ejemplo } K_n \end{aligned}$$