Facultad de Ingeniería
Universidad Nacional de Jujuy

Desarrollo Sistemático de Programas

Unidad 2

Análisis de Algoritmos

2da Parte

Ing. Carlos A. Afranllie

Orden de Algoritmos

Introducción:

Supongamos que para cierto problema hemos hallado dos posibles algoritmos que lo resuelven, A y B.

Evaluamos con cuidado sus costos respectivos y obtenemos que $TA(n)=100n$ y $TB(n)=2n^2$.

¿Cuál será conveniente usar, basándose en el criterio de eficiencia

Orden de Algoritmos

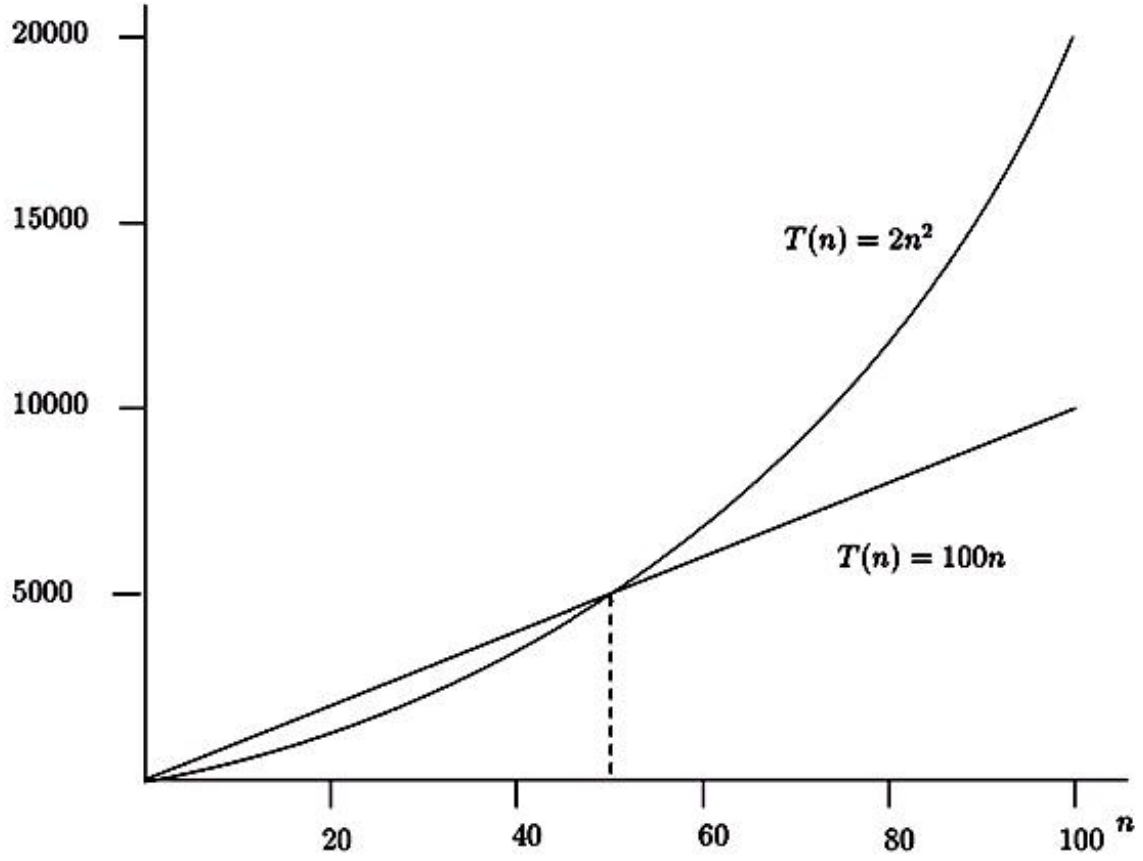


Figura 1: Comparación de los costes $100n$ y $2n^2$

Si n<50, resulta que B es más eficiente que A, pero no mucho más; para n>50 el algoritmo A es mejor que B, y la ventaja de A sobre B se hace mucho mayor cuánto mayor es n. Para entradas de tamaño n=100, A es el doble de rápido que B, pero para entradas de tamaño n=1000, A es veinte veces más rápido que B.

Orden de Algoritmos

Notación O

Esta notación sirve para clasificar el crecimiento de las funciones. Así en vez de decir que el costo T(k) del algoritmo de localización del mínimo es T(k)=4k, diremos que es O(k), lo que informalmente significa que el costo es "alguna constate multiplicada por k" para entradas de tamaño k.

Orden de Algoritmos

Notación O

Habitualmente no interesa cuánto tarda exactamente un algoritmo sino que nos interesa saber cuál es la tasa de crecimiento del algoritmo en función de los datos de entrada. Para el estudio de algoritmos existe una notación muy práctica y utilizada universalmente en este tipo de problemas conocida como la gran "O" (Big-Oh notation) que define el orden de un algoritmo.

Orden de Algoritmos

Definición

"El orden de un algoritmo se define como la tasa de crecimiento del tiempo que insume el algoritmo en función de la cantidad o tamaño de los datos de entrada".

Orden de Algoritmos

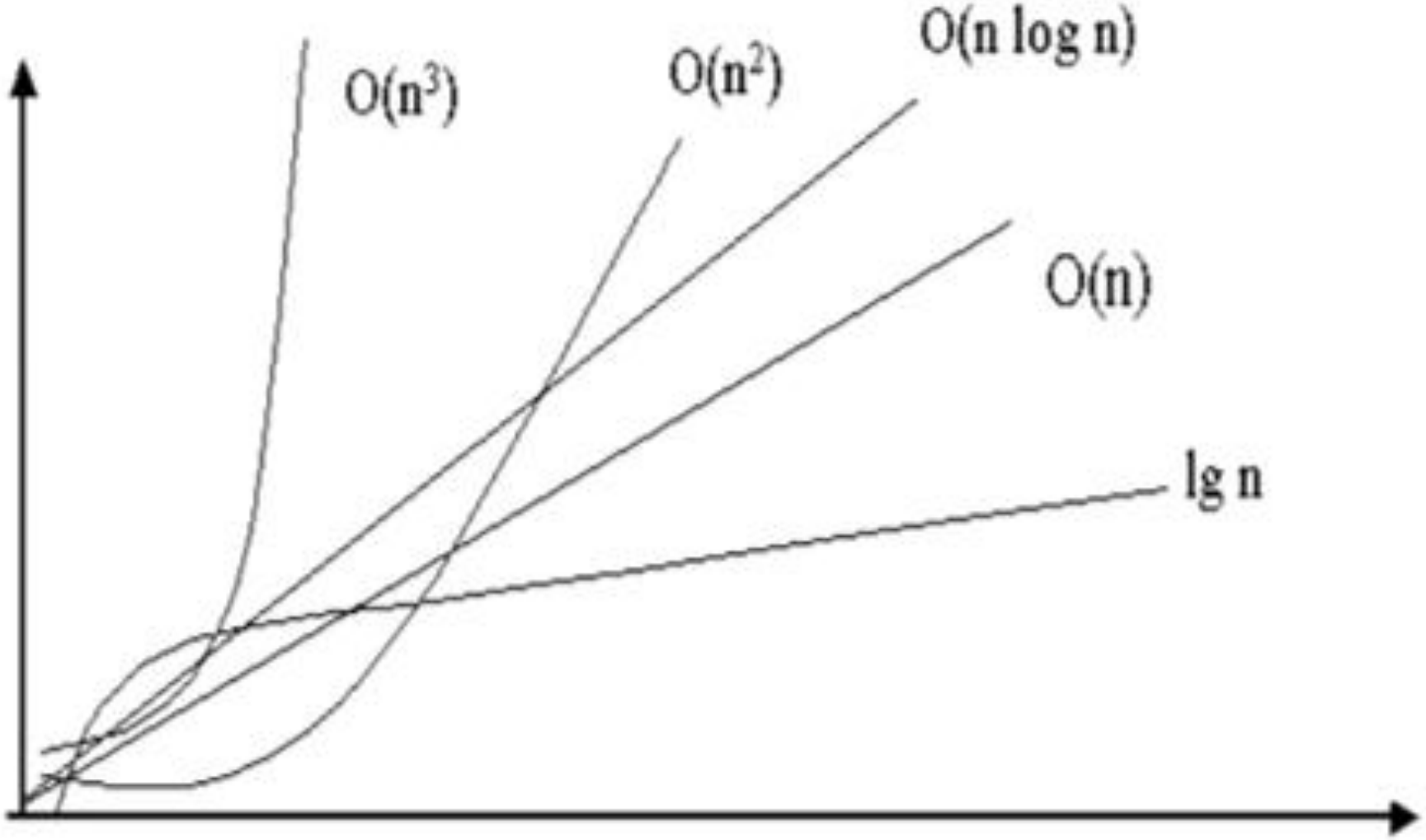
Ordenes más comunes

Orden	Nombre informal
O(1)	Orden constante
O(log n)	Orden logarítmico
O(n)	Orden lineal
O(n2)	Orden cuadrático
O(n3)	Orden cúbico
O(nk)	Orden polinómico
O(2n)	Orden exponencial
O(n!)	Orden factorial

Tabla 1: Nombres informales para algunos órdenes comunes

Orden de Algoritmos

Ordenes más comunes



Orden de Algoritmos

Reglas

• **Regla de las sumas:** Consideremos un algoritmo que consta de k partes compuestas secuencialmente, cada una con un costo $T_i(n)$ y donde $T_i(n) = O(f_i(n))$ para i=1, .., k. Entonces

$$T(n) = T_1(n) + \ldots + T_k(n) = O(max\{ f_1(n), \ldots. f_k(n)\}).$$

• **Reglas de los productos:** Si el costo del cuerpo de una iteración es O(f(n)) y el número de iteraciones es O(g(n)) entonces el costo total de la iteración es O(f(n)*g(n)).

Orden de Algoritmos

Regla de las sumas

Supóngase que un algoritmo consta de dos partes. Si la primera tiene coste $O(n^2)$ y la segunda tiene coste $O(n^3)$, entonces el coste del algoritmo sería

$$T(n) = O(n^2) + O(n^3) = O(n^2 + n^3) = O(\max\{n^2, n^3\}) = O(n^3)$$

Para las estructuras iterativas debemos evaluar el costo del cuerpo y el número de iteraciones que se producirán en función del tamaño de la entrada.

Orden de Algoritmos

Regla de los productos

Si en un algoritmo aparecen bucles anidados se analizan uno por uno desde el más interno al más externo. Por ejemplo, el siguiente segmento de un algoritmo inicializa las componentes de una matriz n x n a 0.

```
para i desde 1 hasta n hacer
        para j desde 1 hasta n hacer
                C[i, j] : = 0
        fin_para
fin_para
```

Si tomamos como tamaño de la entrada la dimensión de la matriz, n, el costo del segmento completo es $O(n^2)$.

Orden de Algoritmos

Algoritmos Recursivos

"Un algoritmo recursivo es aquel que en algún momento durante su ejecución se invoca a si mismo".

Por ejemplo el siguiente programa sirve para calcular el factorial de un número.

Algoritmo Fact(n). Calcula el factorial de n en forma recursiva.

```
si n < 2 entonces
        devolver 1
sino
        devolver n*Fact(n-1)
fin_si
```

Orden de Algoritmos

Algoritmos Recursivos

- En general el programar en forma recursiva permite escribir menos código y algunas veces nos permite simplificar un problema difícil de solucionar en forma iterativa.

- En cuanto a la eficiencia un programa recursivo puede ser mas, menos o igual de eficiente que un programa iterativo.

Orden de Algoritmos

Algoritmos Recursivos

• En primer lugar hay que señalar que los programas recursivos consumen espacio en memoria para la pila ya que cada una de las llamadas recursivas al algoritmo son apiladas una sobre otras para preservar el valor de las variables locales y los parámetros utilizados en cada invocación.

• Este aspecto lo vamos a considerar un costo en espacio ya que lo que estamos consumiendo es memoria.

Orden de Algoritmos

Algoritmos Recursivos

• En cuanto al tiempo de ejecución del programa debemos calcularlo de alguna forma para poder obtener el orden del algoritmo.

• Para ello se recurre a plantear el tiempo que insume un programa recursivo de la siguiente forma:

Tiempo de ejecución del factorial

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

Orden de Algoritmos

Algoritmos Recursivos

- Las ecuaciones planteadas definen una recurrencia.

- Una recurrencia es un sistema de ecuaciones en donde una o más de las ecuaciones del sistema están definidas en función de si mismas.

- Para calcular el orden de un algoritmo recursivo es necesario resolver la recurrencia que define el algoritmo.

Algoritmos Recursivos

Recurrencias

Tomemos por ejemplo la siguiente recurrencia muy común en algoritmos recursivos.

$$T(n) = \begin{cases} 1 \\ 2T(\frac{n}{2}) + n \end{cases}$$

Estudiaremos tres técnicas que suelen utilizarse para resolver recurrencias.

- El método de sustitución
- El método iterativo
- El teorema maestro de las recurrencias.

El método de sustitución

En método de sustitución se utiliza cuando estamos en condiciones de suponer que el resultado de la recurrencia es conocido.

En estos casos lo que hacemos es sustituir la solución en la recurrencia y luego demostrar que el resultado es valido por inducción.

Para nuestro ejemplo sabemos que

$$T(n) = (n \log n) + n$$

Algoritmos Recursivos

El método de sustitución

Veamos que pasa cuando n=1

Si n=1 => T(1)=1

1 log 1 + 1 = 0 + 1 = 1 Verifica

Hipótesis inductiva

si n>1 => T(n')=(n' log n')+n' si n'<n

El método de sustitución

Demostración

$$T(n) = 2T(\frac{n}{2}) + n \qquad (1)$$

Como $\frac{n}{2} < n$

Por inducción

$$T(\frac{n}{2}) = (\frac{n}{2}) * \log(\frac{n}{2}) + \frac{n}{2} \qquad (2)$$

El método de sustitución

Reemplazando (2) en (1)

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

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

$$T(n) = n(\log n - \log 2) + 2n$$

$$T(n) = n\log n - n + 2n$$

$$T(n) = n\log n + n$$

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

Por lo que queda demostrado.

Algoritmos Recursivos

El método de sustitución

- El método de sustitución es simple.

- Requiere que uno ya conozca cual es la solución de la recurrencia.

- A veces sin embargo en recurrencias muy complejas para resolver por otros métodos puede llegar a ser conveniente intentar adivinar la solución y luego demostrarla usando este método.

El método iterativo

• El método iterativo es un forma bastante poderosa de resolver una recurrencia sin conocer previamente el resultado.

• En este método se itera sobre la recurrencia hasta que se deduce un cierto patrón que permite escribir la recurrencia usando sumatorias.

• Luego realizando una sustitución apropiada y resolviendo las sumatorias se llega al resultado de la recurrencia.

El método iterativo

Utilizando el método iterativo en nuestro ejemplo observamos lo siguiente:

$$T(1) = 1$$

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

$$T(n) = 2(2T(\frac{n}{4}) + \frac{n}{2}) + n = 4T(\frac{n}{4}) + n + n$$

$$T(n) = 4(2T(\frac{n}{8}) + \frac{n}{4}) + n + n = 8T(\frac{n}{8}) + n + n + n$$

$$T(n) = 2^k * T(\frac{n}{2^k}) + kn$$

El método iterativo

Como T(1)=1 hacemos $\frac{n}{2^k} = 1 \Rightarrow n = 2^k \Rightarrow k = \log_2 n$

Reemplazando

$$T(n) = 2^{\log_2 n} T(\frac{n}{2^{\log_2 n}}) + n \log_2 n$$

$$Sabemos\,que: 2^{\log_2 n} = n^{\log_2 2} = n^1 = n$$

$$T(n) = nT(\frac{n}{n}) + n \log_2 n = nT(1) + n \log_2 n$$

$$T(n) = n + n \log_2 n$$

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

El teorema maestro de las recurrencias

Mediante este teorema podemos contar con una poderosa herramienta para resolver algunas recurrencias. El teorema dice lo siguiente.

Teorema: Sea:

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

Si $a \geq 1$ y $b > 1$

Entonces

- Caso 1 Si $a > b^k \Rightarrow T(n) = \in \Theta(n^{\log_b a})$
- Caso 2 Si $a = b^k \Rightarrow T(n) = \in \Theta(n^k \log n)$
- Caso 3 Si $a < b^k \Rightarrow T(n) = \in \Theta(n^k)$

El teorema maestro de las recurrencias

Por ejemplo para

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

tenemos que:
a = 2, b = 2, k = 1

luego $a=b^k$ y estamos en el caso 2. Por lo que el algoritmo es $O(n^1 \log n)$

Lamentablemente no todas las recurrencias tienen la forma que requiere el teorema maestro por lo que a veces no queda más remedio que aplicar el método iterativo.

Cambio de variables

En algunas recurrencias puede ser conveniente realizar un cambio de variables de forma tal de resolver una recurrencia ya conocida, luego aplicando la inversa del cambio podemos obtener la solución de la recurrencia original.

Supongamos que tenemos la siguiente recurrencia:

$$T(n) = \begin{cases} 1 & \text{si n = 1} \\ T(\sqrt{n}) + 1 & \text{sino} \end{cases}$$

Cambio de variables

Sea m log n. Entonces $n=2^m$. Reemplazando 2^m por n en la recurrencia de T(n) tenemos:

$$T(2^m) = T(2^{m/2}) + 1$$

Llamamos $S(m) = T(2^m)$. Entonces la recurrencia queda:

$$S(m) = S(m/2) + 1$$

Usando el teorema maestro sabemos que S(m)=O(log m). Por lo tanto

$$T(n) = T(2^m) = S(m) = O(lgm)$$
$$T(n) = O(\log \log n)$$