Algoritmos y estructuras de datos II
Complejidad 2

Carlos Gustavo Lopez Pombo
(Charlie)

Departamento de Computación,
Facultad de ciencias exactas y naturales,
Universidad de Buenos Aires

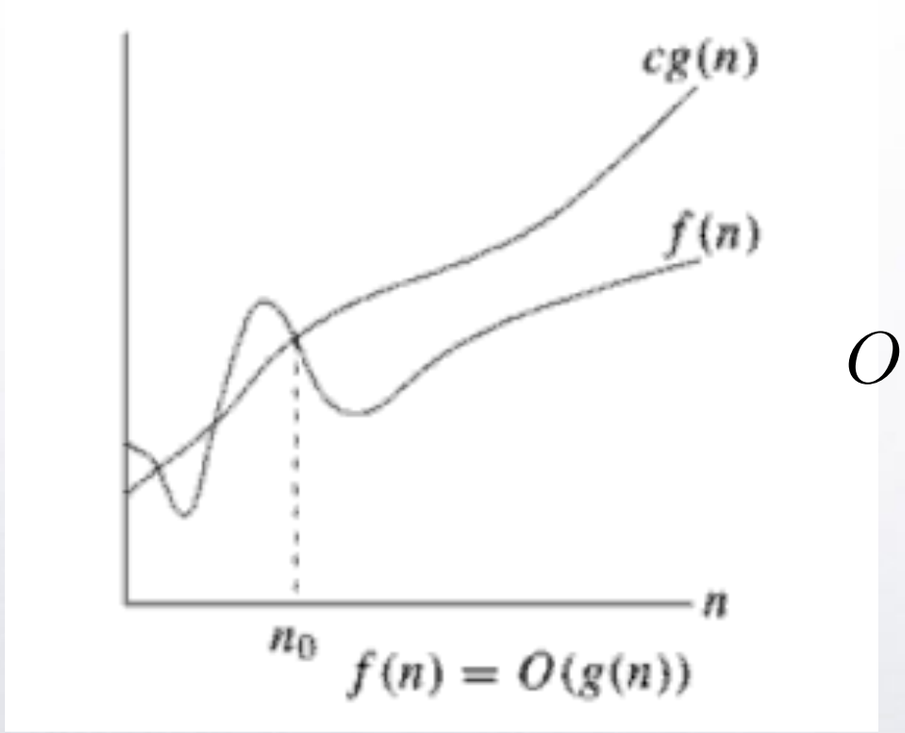
Punteo...

- Aritmética de ordenes de complejidad
- Ecuaciones de recurrencia
- Método de sustitución
- Método de árbol de recursión
- Método maestro: su uso

¿Se acuerdan de esta definición?

Complejidad

La clase $O(g(n))$



$$O(g(n)) = \{f(n)|(\exists c, x_o)(\forall x_0 \leq x)(f(x) < c * g(x))\}$$

Complejidad

¿Y de este algoritmo?

```
void max_min (int *datos, int cant, int &max, int &min){
    max = datos[0];                                          O(1)
    for (int i = 1; i < cant; i++)                           cant*O(1)
      if (max < datos[i]) then max = datos[i];               cant*O(1)
    min = datos[0];                                          O(1)
    for (int i = 1; i < cant; i++)                           cant*O(1)
      if (min > datos[i]) then min = datos[i];               cant*O(1)
}
```

Aritmética de órdenes de complejidad

La pregunta entonces es cómo hacemos para que esas expresiones de complejidad temporal que aparecen al costado del algoritmo

O(1)+cant*O(1)+cant*O(1)+O(1)+cant*O(1)+cant*O(1))

se conviertan en una única expresión O(f(n)) que caracterice su costo...

Aritmética de órdenes de complejidad

$f(m) + O(g(n)) = O(f(m)+g(n))$

$f(m) * O(g(n)) = O(f(m)*g(n))$

$O(f(m)) + O(g(n)) = O(f(m)+g(n))$

$O(f(m)) * O(g(n)) = O(f(m)*g(n))$

Aritmética de órdenes de complejidad

O(1)+cant*O(1)+cant*O(1)+O(1)+cant*O(1)+cant*O(1) =

= 2*O(1)+4*(cant*O(1)) =

= 2*O(1)+4*O(cant) =

= 2*O(1)+O(4*cant) =

= O(2)+O(4*cant) =

= O(1)+O(cant) =

= O(1+cant) = **O(cant)**

Aritmética de órdenes de complejidad

Cuestión importante sobre los logaritmos... son todos iguales a los efectos del cálculo de complejidad temporal

O(log(a)(n)) =

= O(log(b)(n) / log(b)(a)) =

= O(1/log(b)(a) * log(b)(n)) =

= O(log(b)(n))

Ecuaciones de recurrencia

Se denomina **ecuación de recurrencia** a una igualdad de la forma:

$$T(n) = f(n)*T(g(n))+h(n),$$

denotando que la resolución de un problema de tamaño n requiere resolver f(n) sub problemas de tamaño g(n) tal que la integración de sus soluciones cuesta h(n).

Ecuaciones de recurrencia

Entonces, nos gustaría, dada una ecuación de recurrencia, poder dar una función f(n) tal que

$$T(n) <= f(n)$$

Es decir, poder aportar una expresión en n (i.e., f(n)) que acote por encima al término T(n) y así poder decir que T(n) pertenece a O(f(n)).

Ecuaciones de recurrencia

Existen tres métodos para resolver ecuaciones de recurrencia:

- Método de sustitución
- Método de árbol de recursión
- Método maestro

Método de sustitución

El **método de sustitución** se basa en: **a)** "adivinar" una función y luego **b)** probar por inducción que efectivamente esta acota por encima a T(n)...

... veamos un ejemplo; sea la ecuación:

$$T(1)=1 \text{ y } T(n)=2T(n/2)+n,$$

probemos que T(n) pertenece a O(n*log(n)).

Método del árbol de recursión

Normalmente el **método del árbol de recursión** es menos compacto que el **método de sustitución**, resuelve el problema de "adivinar" una solución reemplazando esto por una derivación que permite llegar a la función que caracteriza la complejidad temporal del algoritmo...

Método del árbol de recursión

En muchos casos este método se puede utilizar para construir un candidato que luego se puede demostrar usando inducción de la misma forma que en el caso del método de sustitución...

... veamos un ejemplo, sea la ecuación:

$$T(n)=3T(n/4)+n^2$$

Método maestro

El **método maestro** es el "recetario" para resolver ecuaciones de recurrencia de la forma:

$$T(n) = aT(n/b)+f(n),$$

denotando que la resolución de un problema de tamaño n requiere resolver a subproblemas de tamaño n/b tal que la integración de sus soluciones cuesta f(n).

Método maestro

Teorema Maestro:

Sean $a \geq 1$ y $b > 1$ constantes, sea $f(n)$ una función, y sea $T(n)$ la recurrencia que sigue definida sobre los enteros no negativos

$$T(n) = aT(n/b) + f(n),$$

Entonces $T(n)$ puede ser acotado asintóticamente como sigue:

1. Si $f(n) \in O(n^{log_b(a)-\epsilon})$ para $\epsilon > 0$, entonces $T(n) \in \Theta(n^{log_b(a)})$,
2. Si $f(n) \in \Theta(n^{log_b(a)})$, entonces $T(n) \in \Theta(n^{log_b(a)} log(n))$, y
3. Si $f(n) \in \Omega(n^{log_b(a)+\epsilon})$ para $\epsilon > 0$ y $af(n/b) \leq cf(n)$ para alguna constante $c > 1$ y un n suficientemente grande, entonces $T(n) \in \Theta(f(n))$.

Método maestro

La **intuición** del teorema maestro recae en comparar qué parte del algoritmo domina el costo del problema, es decir, comparar **a)** la base del árbol de recursión en dónde se resuelven los casos base, o **b)** la función que integra las soluciones parciales en una solución final. Esta es la razón por la cual se compara f(n) con la cantidad de casos bases (recordar el árbol de recursión) $n^{log_b(a)}$

Método maestro

Hay una salvedad importante para hacer:
Como se trabaja con cotas asintóticas, las diferencias que se están caracterizando cuando se realiza la comparación son polinomiales en n y por lo tanto, si esto no se cumple el teorema maestro no da respuesta a la caracterización de la complejidad temporal.

Lo mismo ocurre con la condición de regularidad del caso **3.**

Método maestro

Ejemplo: $T(n) = 9*T(n/3)+n$

¡Es todo por hoy!