Análisis de algoritmos

Parte 2: Crecimiento de funciones

Notación asintótica

- En este tema veremos algunos conjuntos de funciones con las que se suele representar la forma de crecimiento asintótico de un algoritmo
- Practicaremos su uso y calcularemos unas pocas de sus relaciones, pero sobre todo, veremos su significado gráfico

¿Qué es un conjunto?

- Colección de elementos u objetos
- Con el nombre del conjunto en común
- No se admiten elementos repetidos

Algunos conjuntos de funciones

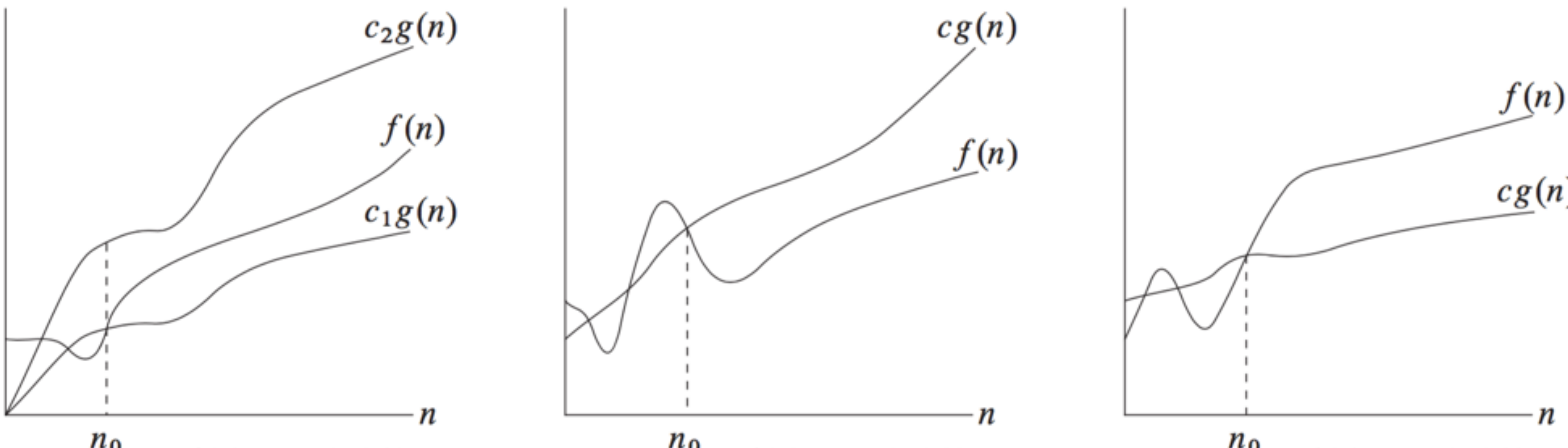
- Rectas: $\{Ax+B: A,B\in\mathbb{R}\}$
- Parábolas: $\{Ax^2+Bx+C: A,B,C\in\mathbb{R}\}$
- Exponenciales: $\{Ae^{Bx}: A,B\in\mathbb{R}\}$
- Logaritmos: $\{A\ln(Bx): A\in\mathbb{R}, B\in\mathbb{R}^+\}$
- Áreas de triángulos con la misma base: $\{bx/2: b\in\mathbb{R}^+\}$

- $O(g(n))=\{f(n):$ existe $n_0>0$ y $c>0$, tal que $0<=f(n)<=cg(n)$ para toda $n>=n_0\}$

- $\Omega(g(n))=\{f(n):$ existe $n_0>0$ y $c>0$, tal que $0<=cg(n)<=f(n)$ para toda $n>=n_0\}$

- $\Theta(g(n))=\{f(n):$ existe $c_1>0, c_2>0, n_0>0,$ tal que $0<=c_1g(n)<=f(n)<=c_2g(n)$ para toda $n>=n_0\}$

gráficamente



$f(n) = \Theta(g(n))$ $\qquad$ $f(n) = O(g(n))$ $\qquad$ $f(n) = \Omega(g(n))$

- El análisis asintótico se refiere a ver el comportamiento de las funciones para valores del parámetro `n` grandes
- Este análisis marca la tendencia de la función, o sea, lo que se puede esperar de ella
- Es el factor **más importante** y en general, **el más útil para analizar un algoritmo**, sin embargo no es el único y a veces las constantes que se “desprecian” son demasiado significativas para no considerarlas.
- Sin embargo, es más o menos fácil identificar cuando estas constantes van a ser significativas.

Abusando de la notación

¿Cómo denotamos una función que está en estos conjuntos?

$\mathtt{f(n)} \in O\mathtt{(g(n))}$ se escribe $\mathtt{f(n)} = O\mathtt{(g(n))}$

$\mathtt{f(n)} \in \Omega\mathtt{(g(n))}$ se escribe $\mathtt{f(n)} = \Omega\mathtt{(g(n))}$

$\mathtt{f(n)} \in \Theta\mathtt{(g(n))}$ se escribe $\mathtt{f(n)} = \Theta\mathtt{(g(n))}$

El "orden" de una función

- Toda función se puede escribir $f(x)=O(f(x))$
- En efecto, si $f(x)>=0$, tenemos que para $c=1$ y $n_0=1$ tenemos:

$$0<=f(x)<=cf(x)$$

- De manera similar $f(x)=\Omega(f(x))$ y $f(x)=\Theta(f(x))$

- Así, si
$$f(x) = \tfrac{1}{2}x + 3$$
- entonces
$$\tfrac{1}{2}x + 3 = O(\tfrac{1}{2}x + 3)$$
- pero la importancia de esta notación asintótica es su poder para simplificar la notación de funciones

- Por ejemplo, podemos también escribir
$$\tfrac{1}{2}x + 3 = O(x)$$
- lo que demostraremos a continuación…

- Por definición, si queremos demostrar

$$\frac{1}{2}x + 3 = O(x)$$

- entonces tenemos que mostrar que existen

$$c > 0, n_0 > 0$$

- tal que

$$0 \leq f(x) \leq cg(x)$$

- donde

$$f(x) = \frac{1}{2}x + 3, g(x) = x$$

- Así tenemos

$$0 \leq \frac{1}{2}x + 3 \leq cx$$

- De la desigualdad derecha tenemos

$$\frac{1}{2}x + 3 \leq cx$$

- pasamos las x del lado mayor y lo demás del lado menor:

$$3 \leq cx - \frac{1}{2}x$$

- factorizamos x

$$3 \leq (c - \frac{1}{2})x$$

- lo del paréntesis no puede ser cero ni negativo (pues el producto no podría ser mayor que 3), para poder dividir por lo de la izquierda, así escogemos un valor $c > \frac{1}{2}$ por ejemplo $c = 1$, así:

$$x \geq \frac{3}{1-\frac{1}{2}} = \frac{3}{\frac{1}{2}} = 6$$

- Con lo que el valor para n_0 se escoge

$$n_0 = 6$$

- con lo que demostramos que efectivamente

$$\frac{1}{2}x + 3 = O(x)$$

- De la misma manera podemos demostrar que

$$\frac{1}{2}x + 3 = \Omega(x)$$

- así, queremos demostrar que existen

$$c > 0, n_0 > 0$$

- tales que

$$0 \leq cx \leq \frac{1}{2}x + 3$$

- Con la desigualdad derecha observamos que

$$\begin{aligned} cx &\leq \frac{1}{2}x + 3, \\ -3 &\leq \frac{1}{2}x - cx, \\ -3 &\leq (\frac{1}{2} - c)x \end{aligned}$$

- Lo del paréntesis no puede ser cero ni negativo, o al pasar dividiendo, la x estará del lado menor, y queremos que esté del lado mayor. Además la c debe ser positiva, así $0<c<1/2$, tomemos $c=1/4$.

- Con lo que tenemos que $x \geq \dfrac{-3}{\frac{1}{2}-\frac{1}{4}} = \dfrac{-3}{\frac{1}{4}} = -12$
- Si $n_0 = 1$, el primer positivo mayor que -12, todo lo anterior se cumplirá para $x \geq n_0$
- con lo que efectivamente $\frac{1}{2}x + 3 = \Omega(x)$
- Con estas dos pruebas y el siguiente teorema tendremos que $\frac{1}{2}x + 3 = \Theta(x)$
- En efecto, el **Teorema 1** relaciona O, Ω y Θ

$$f(n) = O(g(n)), f(n) = \Omega(g(n)) \iff f(n) = \Theta(g(n))$$

- Probemos primero que
$f(n) = O(g(n)), f(n) = \Omega(g(n)) \implies f(n) = \Theta(g(n))$
- Así sabemos que existen constantes
$$n_0 > 0, m_0 > 0, c_1 > 0, c_2 > 0$$
- tales que
$$0 \leq c_1 g(n) \leq f(n)$$
- y
$$0 \leq f(n) \leq c_2 g(n)$$
- tomemos
$$k_0 = \max\{n_0, m_0\}$$
- con lo que para toda $x > k_0$ se tiene
$$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$

$\square$

- Probemos ahora que
  $$f(n) = \Theta(g(n)) \Longrightarrow f(n) = O(g(n)), f(n) = \Omega(g(n))$$
- así, sabemos que existen
  $$n_0 > 0, c_1 > 0, c_2 > 0$$
- tales que para toda $x > n_0$ se tiene
  $$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$$
- para toda $n > n_0$, por lo que

- de la 1a y 2a desigualdad (desde la izquierda) tenemos que
  $$f(n) = \Omega(g(n))$$

- para de la 1a y la 3a desigualdad tenemos que
  $$f(n) = O(g(n))$$

❒

- Antes de ver más ejemplos y resultados, veamos la interpretación de estos conjuntos de funciones, en adelante simplemente las llamaremos "notación asintótica" en lo que se ha visto hasta ahora de la complejidad espacio-temporal de algoritmos, en particular veremos los ejemplos de inserción y mezclas.

Complejidad temporal del algoritmo de inserción

- En el algoritmo de ordenamiento por inserción vemos que tiene el ciclo (que empieza en la línea 1) que controla la variable `j` y dentro tiene un ciclo (que empieza en la línea 4), que controla la variable `i`.

- La variable `j` toma valores desde `1` hasta `len(A)-1`, por lo que produce `n-1` iteraciones.

- A su vez, el ciclo de la variable `i` produce entre `1` y `j-1` iteraciones.

- El mejor caso (la lista está ordenada) es cuando el ciclo de `i` produce `1` iteración por cada iteración de `j`, o sea `n-1` **iteraciones en total.**

- Mientras que en el peor caso es cuando se producen `j-1` iteraciones por cada iteración de `j`.

- Como la cantidad total de iteraciones se puede calcular con la suma de los enteros desde `1` hasta `n-1`, esta cuenta nos da `(n-2)(n-1)/2` **iteraciones en total** en el peor caso (cuando la lista está ordenada al revés).

```
inserción(A):
Para j de 1 a len(A)-1
    pivote = A[j] # carta a acomodar
    i = j - 1 # posición a la izquierda del pivote
    Mientras i >= 0 y A[i] > pivote
        A[i+1] = A[i]
        i = i - 1
    A[i + 1] = pivote
```

- Así, podemos escribir las funciones para los casos mejor y peor de esta forma:
  $f_{mejor}(n) = n\text{-}1$
  $f_{peor}(n) = n(n\text{-}1)/2=(n^2\text{-}n)/2$

- Ej. Ver que $f_{mejor}(n)=\Theta(n)$ y $f_{peor}(n)=\Theta(n^2)$

- Para la mayoría de los algoritmos se prefiere hablar de una sóla complejidad temporal, en este caso la peor. Sin embargo en este caso, no podríamos decir que $\Theta(n^2)$ es lo que inserción se va a tardar para **todas las entradas posibles** (si el arreglo está ordenado, esto sería $\Theta(n)$)

- Sin embargo, con estas notaciones, podemos decir que inserción se tarda entre $\Omega(n)$ y $O(n^2)$. Es decir, existen entradas para las cuales se tarda n y otras para las cuales se tarda n^2

- Así, Ω nos dice la **cota inferior**, o el tiempo en el **mejor caso**, mientras que O nos dice la **cota superior**, o el tiempo en el **peor caso**. Como O nos dice el tiempo en el peor caso, por lo general ese es el que se suele decir que cubre este algoritmo.

- Si $a_n \neq 0$
  $$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 = \Theta(x^n)$$

Para cotas más justas existen…

- o(g(n))={f(n): para toda c>0, existe n_0>0 tal que 0<=f(n)<=cg(n) para toda n>=n_0}

- ω(g(n))={f(n): para toda c>0, existe n_0>0 y, tal que 0<=cg(n)<=f(n) para toda n>=n_0}

- estas denotan “cotas justas”