Complejidad computacional (repaso)

Tecnología Digital V: Diseño de Algoritmos

Universidad Torcuato Di Tella

UNIVERSIDAD TORCUATO DI TELLA

Complejidad computacional: Las reglas del juego

Problema

En el contexto de la teoría de complejidad computacional, llamamos problema a la descripción de los datos de entrada y la respuesta a proporcionar para cada dato de entrada.

Instancia

Una instancia de un problema es un juego válido de datos de entrada.

- Ejemplo:

  1. **Entrada:** Un número n entero no negativo.
  2. **Salida:** ¿El número n es primo?

- En este ejemplo, una instancia está dada por un número entero no negativo.

Complejidad computacional: Las reglas del juego

- Suponemos una Máquina RAM (*random access memory*).
  1. La memoria está dada por una sucesión de celdas numeradas. Cada celda puede almacenar un valor de b bits.
  2. Supondremos habitualmente que el tamaño b en bits de cada celda está fijo, y suponemos que todos los datos individuales que maneja el algoritmo se pueden almacenar con b bits.
  3. Se tiene un programa imperativo no almacenado en memoria, compuesto por asignaciones y las estructuras de control habituales.
  4. Las asignaciones pueden acceder a celdas de memoria y realizar las operaciones estándar sobre los tipos de datos primitivos habituales.

Complejidad computacional: Las reglas del juego

- Cada instrucción tiene un tiempo de ejecución asociado.
  1. El acceso a cualquier celda de memoria, tanto para lectura como para escritura, es *constante*.
  2. Las asignaciones y el manejo de las estructuras de control se realiza en *constante*.
  3. Las operaciones entre valores lógicos son *constantes*.

- Las operaciones entre enteros/reales dependen de b:
  1. Las sumas y restas *trabajan bit a bit* y requieren tiempo lineal en b.
  2. Las multiplicaciones y divisiones requieren algoritmos más sofisticados (más detalles en las próximas slides!).

Repaso: Notación O

Definición

Dadas dos funciones $f, g : \mathbb{N} \to \mathbb{R}$, decimos que

- $f(n) = O(g(n))$ si existen $c \in \mathbb{R}_+$ y $n_0 \in \mathbb{N}$ tales que $f(n) \leq c\ g(n)$ para todo $n \geq n_0$,
- $f(n) = \Omega(g(n))$ si existen $c \in \mathbb{R}_+$ y $n_0 \in \mathbb{N}$ tales que $f(n) \geq c\ g(n)$ para todo $n \geq n_0$,
- $f(n) = \Theta(g(n))$ si $f = O(g(n))$ y $f = \Omega(g(n))$.

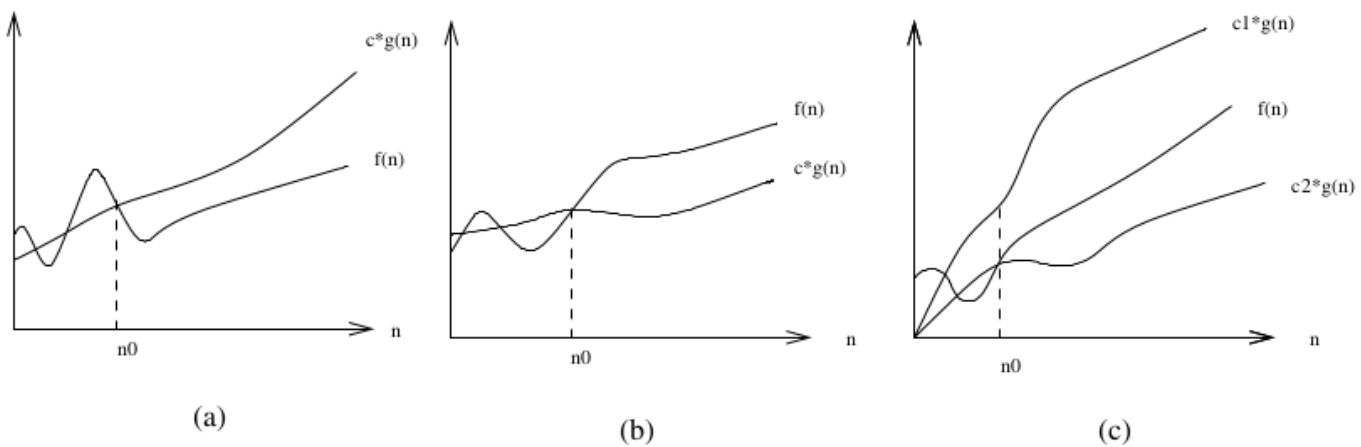

Figure 2.3: Illustrating the big (a) O, (b) Ω, and (c) Θ notations

Complejidad computacional: Las reglas del juego

Tiempo de ejecución de un algoritmo A

$T_A(I)$ = suma de los tiempos de ejecución de las instrucciones realizadas por el algoritmo con la instancia I.

Tamaño de una instancia

Dada una instancia I, definimos $|I|$ como la cantidad de bits necesarios para almacenar los datos de entrada de I.

- Si b está fijo y la entrada ocupa n celdas de memoria, entonces $|I| = bn = O(n)$.

Complejidad de un algoritmo A

$f_A(n) = \max_{I:|I|=n} T_A(I)$.

Repaso: Notación O

- Cada instrucción tiene un tiempo de ejecución asociado.
  1. El acceso a cualquier celda de memoria, tanto para lectura como para escritura, es $O(1)$.
  2. Las asignaciones y el manejo de las estructuras de control se realiza en $O(1)$.
  3. Las operaciones entre valores lógicos son $O(1)$.

- Las operaciones entre enteros/reales dependen de b:
  1. Las sumas y restas son $O(b)$.
  2. Las multiplicaciones y divisiones son $O(b \log b)$[1].

$\Rightarrow$ Si b está fijo, estas operaciones son $O(1)$. En cambio, si no se puede suponer esto, entonces hay que contemplar que el costo de estas operaciones depende de b.

[1] David Harvey y Joris van der Hoeven, *Integer multiplication in time $O(n \log n)$*. Annals of Mathematics 193-2 (2021) 563–617.

Repaso: Notación O

- Si un algoritmo es $O(n)$, se dice lineal.
- Si un algoritmo es $O(n^2)$, se dice cuadrático.
- Si un algoritmo es $O(n^3)$, se dice cúbico.
- Si un algoritmo es $O(n^k)$, $k \in \mathbb{N}$, se dice polinomial.
- Si un algoritmo es $O(\log n)$, se dice logarítmico.
- Si un algoritmo es $O(d^n)$, $d \in \mathbb{R}_+$, se dice exponencial.

- Cualquier función exponencial es *peor* que cualquier función polinomial: Si $k, d \in \mathbb{N}$ entonces k^n no es $O(n^d)$.

- La función logarítmica es *mejor* que la función lineal (no importa la base), es decir $\log n$ es $O(n)$ pero no a la inversa.

Repaso: Trabajando con la notación O

- **Suma de funciones**.

$$O(f(n)) + O(g(n)) \quad \longrightarrow \quad O(f(n) + g(n))$$

- **Multiplicación de funciones**. Sea $c \in \mathbb{R}$,

$$c * O(f(n)) \quad \longrightarrow \quad O(f(n))$$
$$O(f(n)) \times O(g(n)) \quad \longrightarrow \quad O(f(n) \times g(n))$$

Ejercicio 1

Dados dos algoritmos

- $A \in O(n)$
- $B \in O(n^2)$

¿Cuál es la complejidad ejecutar B inmediatamente después de A?

Ejercicio 2

Demostrar que las relaciones O son transitivas. Si

- $f(n) = O(g(n))$, y
- $g(n) = O(h(n))$,

entonces $f(n) = O(h(n))$.

Problemas bien resueltos

Definición

Decimos que un problema está bien resuelto si existe un algoritmo de complejidad polinomial que lo resuelve (en forma exacta).

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$
$O(n)$	0.01 ms	0.02 ms	0.03 ms	0.04 ms	0.05 ms
$O(n \log n)$	0.01 ms	0.03 ms	0.04 ms	0.06 ms	0.08 ms
$O(n^2)$	0.10 ms	0.40 ms	0.90 ms	1.60 ms	2.50 ms
$O(n^3)$	1.00 ms	8.00 ms	27.00 ms	64.00 ms	0.12 sg
$O(2^n)$	1.02 ms	1.04 sg	17.90 min	12 días	35 años
$O(3^n)$	0.59 sg	58 min	6 años	3855 siglos	2×10^8 siglos!

Problemas bien resueltos

Instancia más grande que se puede resolver en 1 minuto en función de la velocidad de la computadora disponible:

	92 kIPS IBM 4004 (1971)	2.66 MIPS Intel 286 (1982)	9.7 MIPS Pentium IV (2000)	177 MIPS Intel Core i7 (2011)	
$O(n)$	5520	$1.5\ 10^5$	$5.8\ 10^5$	$1.06\ 10^7$	(1923x)
$O(n \log n)$	4141	$1.0\ 10^5$	$2.7\ 10^5$	$6.19\ 10^6$	(1495x)
$O(n^2)$	74	399	762	3258	(43x)
$O(n^3)$	17	54	83	219	(12x)
$O(2^n)$	12	17	19	23	(1.88x)
$O(3^n)$	7	10	12	14	(1.88x)

Problemas bien resueltos

Convención

Los algoritmos polinomiales se consideran satisfactorios (cuanto menor sea el grado, mejor), y los algoritmos supra-polinomiales se consideran no satisfactorios.

No obstante ...

- Si los tamaños de instancia son pequeños, ¿es tan malo un algoritmo exponencial?
- ¿Cómo se comparan $O(n^{85})$ con $O(1.001^n)$?
- ¿Puede pasar que un algoritmo de peor caso exponencial sea eficiente en la práctica?
- ¿Puede pasar que en la práctica sea *el mejor*?
- ¿Qué pasa si no encuentro un algoritmo polinomial?