Análisis y Diseño de Algoritmos

Notación Asintótica

DR. JESÚS A. GONZÁLEZ BERNAL
CIENCIAS COMPUTACIONALES
INAOE

Introducción

- ¿Por qué el análisis de algoritmos?
  - Determinar tiempos de respuesta (runtime)
  - Determinar recursos computacionales
- Aproximación teórica
  - Generaliza el número de operaciones que requiere un algoritmo para encontrar la solución a un problema

Introducción

- Ventajas
  - Elección de algoritmos eficientes para resolver problemas específicos
  - No depende de lenguajes de programación ni de hardware
- Desventajas
  - Para muchos casos, en análisis no es trivial

Introducción

- Para realizar el análisis de un algoritmo, es necesario:
  - Conocer la complejidad del problema que resuelve el algoritmo
  - Conocer la dimensión de la entrada (número de elementos)
  - Determinar el número de operaciones a realizar
- La complejidad de un algoritmo se representa a través de una función matemática
  - Polinomios
  - Logaritmos
  - Exponentes…

Introducción

- Funciones
  - $f(n) = cn$ (algoritmos lineales)
  - $f(n) = cn^2$ (algoritmos cuadráticos)
  - $f(n) = cn^3$ (algoritmos cúbicos)
- Un algoritmo puede estar compuesto de dos o más operaciones, por lo que determinar la complejidad depende de identificar la operación más costosa en el algoritmo
  - Por ejemplo, sume 2 matrices e imprima el resultado. ¿de que orden es el problema?

Principio de Invarianza

- A través de un análisis teórico, se pueden obtener funciones que representen el número de operaciones, independientemente de cómo se implementaron
- Análisis "Principio de la Invarianza"
  - Dos implementaciones distintas de un mismo algoritmo no van a diferir en su eficiencia en más de una constante multiplicativa "c"

Análisis Peor Caso – Caso Promedio - Mejor Caso

- El tiempo que requiere un algoritmo para dar una respuesta, se divide generalmente en 3 casos
  - Peor Caso: caso más extremo, donde se considera el tiempo máximo para solucionar un problema
  - Caso promedio: caso en el cual, bajo ciertas restricciones, se realiza un análisis del algoritmo
  - Mejor caso: caso ideal en el cual el algoritmo tomará el menor tiempo para dar una respuesta
- Por ejemplo, ¿Cuál es el peor y mejor caso de el algoritmo de ordenamiento “burbuja”?

Operación Elemental (OE)

- Es aquella operación cuyo tiempo de ejecución se puede acotar superiormente por una constante que solamente dependerá de la implementación particular usada
  - No depende de parámetros
  - No depende de la dimensión de los datos de entrada

Crecimiento de Funciones

- Orden de crecimiento de funciones
  - Caracteriza eficiencia de algoritmos
  - Permite comparar performance relativo de algoritmos
- Es posible en ocasiones calcular el tiempo de ejecución exacto
  - No siempre vale la pena el esfuerzo
  - Las constantes y términos de orden más bajo son dominados por los efectos del tamaño de la entrada

Crecimiento de Funciones

- Diccionario de la Real Academia Española
  - Asintótico, ca (De asíntota).
    - Adj. Geom. Dicho de una curva: Que se acerca de continuo a una recta o a otra curva sin llegar nunca a encontrarla.

Crecimiento de Funciones

- Eficiencia Asintótica de Algoritmos
  - Cuando el tamaño de la entrada es suficientemente grande que sólo el orden de crecimiento del tiempo de ejecución es relevante.
  - Sólo importa cómo incrementa el tiempo de ejecución con el tamaño de la entrada en el límite
    - El tamaño de la entrada incrementa sin frontera
- Usualmente el algoritmo asintóticamente más eficiente es la mejor opción, excepto para entradas muy pequeñas

Notación Asintótica

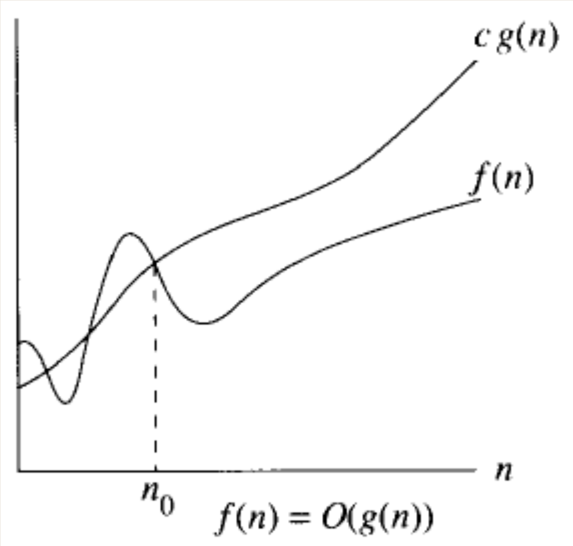
- Eficiencia Asintótica
  - Orden de crecimiento del algoritmo conforme el tamaño de la entrada se acerca al límite (incrementa sin frontera)

- Para determinar la complejidad de un algoritmo, se siguen los siguientes pasos:
  - Se analiza el algoritmo para determinar una función que represente el número de operaciones a realizar por el mismo
  - Se define el orden de la función en términos de funciones matemáticas,
  - Se clasifica de acuerdo a su complejidad

Notación O

- $f(n) = O(g(n))$, $g(n)$ es una cota superior de $f(n)$
- Dada una función $g(n)$, denotamos como $O(g(n))$ al conjunto de funciones tales que:

$O(g(n)) = \{f: N \rightarrow R^+ \mid \exists\ c$ constante positiva y $n_0 \in N : f(n) \leq cg(n),\ \forall\ n \geq n_0\}$



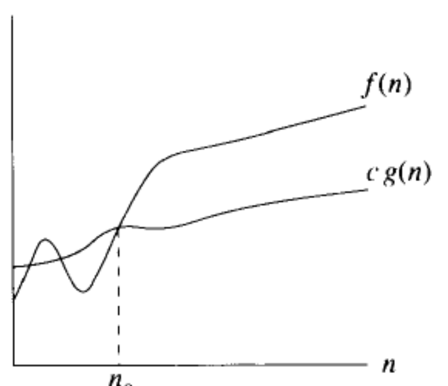
Propiedades de O

1. Para cualquier función de f se tiene que $f \in O(f)$.
2. $f \in O(g) \Rightarrow O(f) \subset O(g)$.
3. $O(f) = O(g) \Leftrightarrow f \in O(g)$ y $g \in O(f)$.
4. Si $f \in O(g)$ y $g \in O(h) \Rightarrow f \in O(h)$.
5. Si $f \in O(g)$ y $f \in O(h) \Rightarrow f \in O(\min(g,h))$.
6. Regla de la suma: Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 + f_2 \in O(\max(g,h))$.
7. Regla del producto: Si $f_1 \in O(g)$ y $f_2 \in O(h) \Rightarrow f_1 \bullet f_2 \in O(g \bullet h)$.
8. Si existe $\lim_{n \to \infty} \frac{f(n)}{g(n)} = k$, dependiendo del valor de k obtenemos:
   a) Si $k \neq 0$ y $k < \infty$ entonces $O(f) = O(g)$.
   b) Si $k = 0$ entonces $f \in O(g)$, es decir, $O(f) \subset O(g)$, pero sin embargo se verifica que $g \notin O(f)$.

Notación Omega: Ω

- $f(n) = \Omega(g(n))$, $g(n)$ es una cota asintótica inferior de $f(n)$
- Dada una función $g(n)$, denotamos al conjunto de funciones $\Omega(g(n))$ de la siguiente forma:

  $\Omega(g(n)) = \{f{:}N \rightarrow R^{+} \mid \exists\, c$ constante positiva y n_0: $0 < cg(n) \leq f(n)$, $\forall\, n \geq n_0\}$



$f(n) = \Omega(g(n))$

NOTA: $f(n) \in \Omega\,(g(n))$ sí y solo si $g(n) \in O(f(n))$

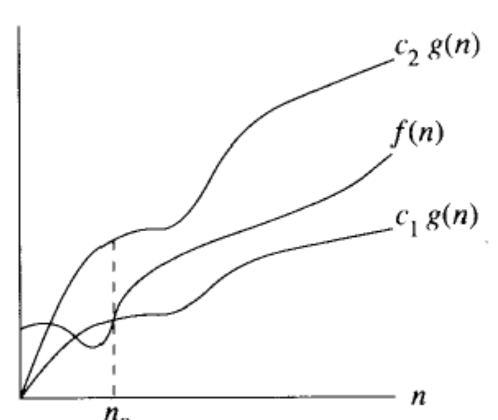
Propiedades de Omega

1. Para cualquier función de f se tiene que $f \in \Omega(f)$.
2. $f \in \Omega(g) \Rightarrow \Omega(f) \subset \Omega(g)$.
3. $\Omega(f) = \Omega(g) \Leftrightarrow f \in \Omega(g)$ y $g \in \Omega(f)$.
4. Si $f \in \Omega(g)$ y $g \in \Omega(h) \Rightarrow f \in \Omega(h)$.
5. Si $f \in \Omega(g)$ y $f \in \Omega(h) \Rightarrow f \in \Omega(\max(g,h))$.
6. Regla de la suma: Si $f_1 \in \Omega(g)$ y $f_2 \in \Omega(h) \Rightarrow f_1 + f_2 \in \Omega(g + h)$.
7. Regla del producto: Si $f_1 \in \Omega(g)$ y $f_2 \in \Omega(h) \Rightarrow f_1 \bullet f_2 \in \Omega(g \bullet h)$.
8. Si existe $\lim_{n\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo del valor de k obtenemos:
   a) Si $k \neq 0$ y $k < \infty$ entonces $\Omega(f) = \Omega(g)$.
   b) Si $k = 0$ entonces $g \in \Omega(f)$, es decir, $\Omega(g) \subset \Omega(f)$, pero sin embargo se verifica que $f \notin \Omega(g)$.

Notación Theta: Θ

- $f(n) = \Theta(g(n))$, $c_2 g(n)$ y $c_1 g(n)$ son las cotas asintóticas de $f(n)$ tanto superior como inferior respectivamente
- Diremos que $f(n) \in \Theta(g(n))$ si $f(n)$ pertenece tanto a $O(g(n))$ como a $\Omega(g(n))$

  $\Theta(g(n)) = \{f{:}N \rightarrow R^+ \mid \exists c_1, c_2$ constantes positivas, n_0: $0 < c_1 g(n) \leq f(n) \leq c_2 g(n),\ \forall\, n \geq n_0\}$



$f(n) = \Theta(g(n))$

Propiedades de Theta

1. Para cualquier función f se tiene que $f \in \Theta(f)$.
2. $f \in \Theta(g) \Rightarrow \Theta(f) \subset \Theta(g)$.
3. $\Theta(f) = \Theta(g) \Leftrightarrow f \in \Theta(g)$ y $g \in \Theta(f)$.
4. Si $f \in \Theta(g)$ y $g \in \Theta(h) \Rightarrow f \in \Theta(h)$.
5. Regla de la suma: Si $f_1 \in \Theta(g)$ y $f_2 \in \Theta(h) \Rightarrow f_1 + f_2 \in \Theta(\max(g,h))$.
6. Regla del producto: Si $f_1 \in \Theta(g)$ y $f_2 \in \Theta(h) \Rightarrow f_1 \bullet f_2 \in \Theta(g \bullet h)$.
7. Si existe $\lim_{n\to\infty} \frac{f(n)}{g(n)} = k$, dependiendo del valor de k obtenemos:
   a) Si $k \neq 0$ y $k < \infty$ entonces $\Theta(f) = \Theta(g)$.
   b) Si $k = 0$ entonces los órdenes exactos de f y g son distintos.

Notación Theta

- Teorema 2.1
  - $f(n) = \Theta(g(n))$ sí y solo si $f(n) = O(g(n))$ y $f(n) = \Omega(g(n))$.

Ejemplo – Theta (1/3)

- Considere la función $f(n) = ½ n^2 - 3n$
  - Debido a que $f(n)$ es un polinomio cuadrático, se deduce que su estructura general tiene la forma $an^2 + bn + c$
  - Para n muy grande, an^2 "domina" al resto de la ecuación
  - Por tanto, se propone una $g(n) = n^2$ de tal forma que se demostrará si $f(n) \in \Theta(n^2)$

Ejemplo – Theta (2/3)

- Para demostrarlo, se debe apelar a la definición de Θ :

  $\Theta(n^2) = \{f(n) \mid \exists c_1, c_2$ constantes positivas, n_0: $0 < c_1 n^2 \leq f(n) \leq c_2 n^2$, $\forall\, n \geq n_0\}$, donde $f(n) = ½\, n^2 - 3n$

  → Se deben encontrar c_1, c_2 y n_0 para los cuales se cumple

  $0 < c_1 n^2 \leq ½\, n^2 - 3n \leq c_2\, n^2$

  → $0 < c_1 \leq ½ - 3/n \leq c_2$

  → Esta ecuación se analiza por casos:

  → $c_1 \leq ½ - 3/n$

  → $½ - 3/n \leq c_2$

Ejemplo – Theta (3/3)

- Para el caso $c_1 \leq ½ - 3/n$
  - Como c_1 es constante positiva, entonces

    $0 < ½ - 3/n$

    $\Rightarrow n > 6$
  - Por tanto, si $n_0 = 7$, entonces $c_1 \leq ½ - 3/7$, lo que es igual a $c_1 \leq 1/14$. Sea $c_1 = 1/14$
- Para el caso $½ - 3/n \leq c_2$, cuando $n \rightarrow \infty$ entonces $½ - 3/n \rightarrow ½$. Por tanto, $c_2 = 1/2$
- Para $c_1 = 1/14$, $c_2 = ½$ y $n_0 = 7$ se cumple que $f(n) \in \Theta(n^2)$

Ejemplo O (1/2)

- Considere la función $f(n) = 2n^2 + 3n + 1$
  - Debido a que $f(n)$ es un polinomio cuadrático, se deduce que su estructura general tiene la forma $an^2 + bn + c$
  - Para n muy grande, an^2 "domina" al resto de la ecuación
  - Por tanto, se propone una $g(n) = n^2$ de tal forma que se demostrará si $f(n) \in O(n^2)$

Ejemplo O (2/2)

- Para demostrarlo, se debe apelar a la definición de O:

  $O(n^2) = \{f(n) \mid \exists\, c$ constante positiva, n_0: $0 < f(n) \leq c\, n^2,\ \ \forall\, n \geq n_0\}$,
  donde $f(n) = 2n^2 + 3n + 1$

→ Se deben encontrar c y n_0 para los cuales se cumple

$$0 < 2n^2 + 3n + 1 \leq c\, n^2$$

→ $0 < 2 + 3/n + 1/n^2 \leq c$

Notemos que si $n \rightarrow \infty$, $2 + 3/n + 1/n^2 \rightarrow 2$

Si $n = 1$ entonces $2 + 3/n + 1/n^2 = 6$

Por tanto, para $c = 6$ y $n_0 = 1$, se demuestra que $f(n) \in O(n^2)$

Notación *o*

- $f(n) = o(g(n))$, $g(n)$ es una cota superior de $f(n)$ que no es asintóticamente justa (*tight*)

Notación ω

- $f(n) = \omega(g(n))$, $g(n)$ es una cota inferior de $f(n)$ que no es asintóticamente justa (*tight*)

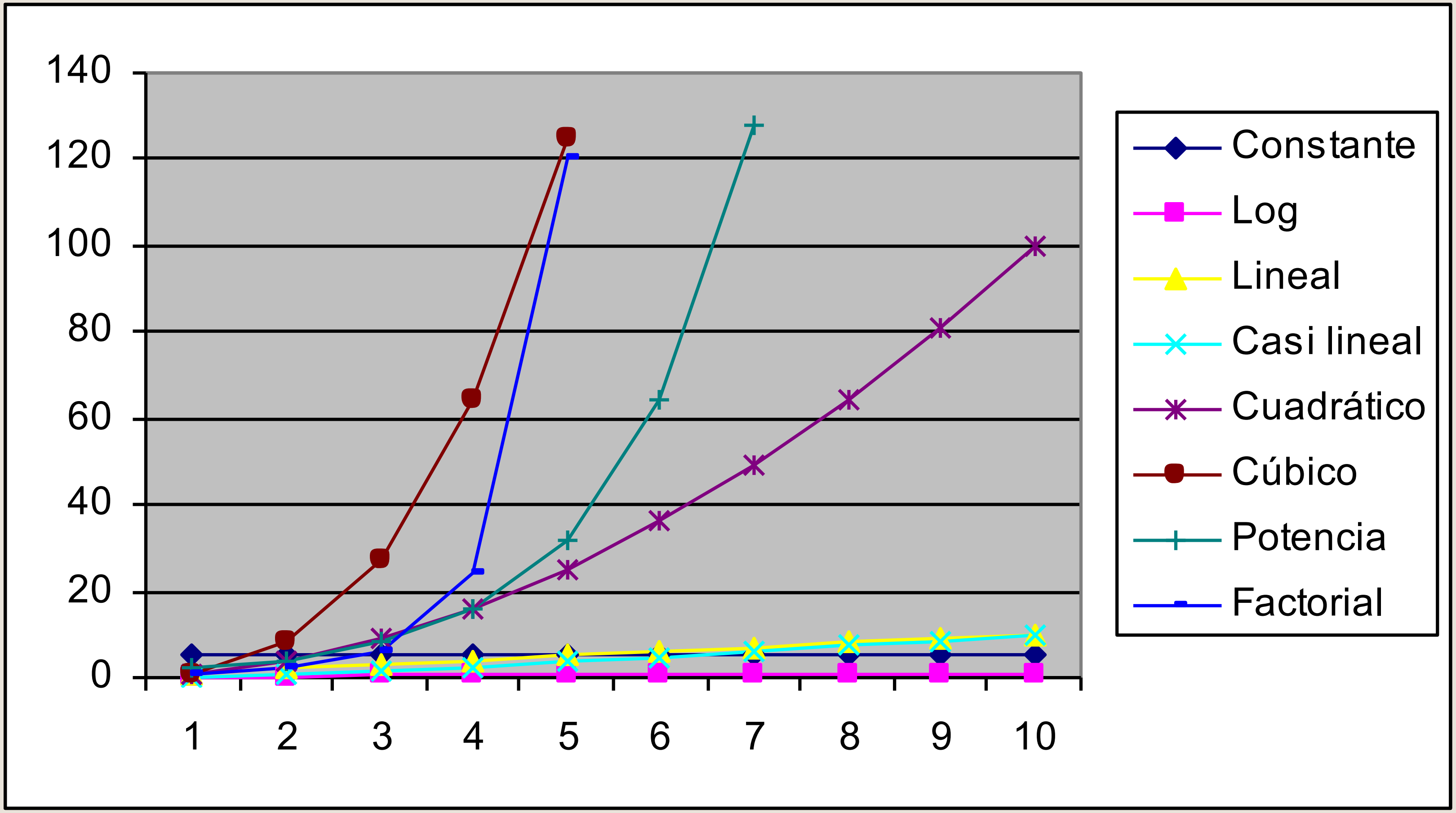
Orden de Complejidad

- La familia $O(f(n))$, $\Omega(f(n))$, $\Theta(f(n))$ define un orden de complejidad
  - Se elige como representante del orden de complejidad a la función $f(n)$ más sencilla de la familia
- Se identifican diferentes familias de orden de complejidad

Orden de Complejidad

- $O(c)$: Orden constante
- $O(\log n)$: orden logarítmico
- $O(n)$: orden lineal
- $O(n \log n)$: orden “casi lineal”
- $O(n^2)$: Orden cuadrático
- $O(n^3)$: Orden cúbico
- $O(n^c)$: Orden polinómico de grado “c”
- $O(2^n)$: Orden exponencial
- $O(n!)$: Orden factorial

Orden de Complejidad



Consejos

- Consejos para Identificar $f(n)$ que Represente el Número de Operaciones Elementales (OE) de un Algoritmo

Consejo 1

- Se asume que el tiempo de una OE es de orden 1
  - La constante "*c*" del principio de la invarianza se asume, por fines prácticos, como 1
- El tiempo de ejecución de una secuencia de instrucciones (elementales o no elementales), se obtiene sumando los tiempos de ejecución de cada instrucción

Consejo Instrucción "case"

- El tiempo de ejecución de una instrucción "switch(n) – case 1: S_1, ..., case k: S_k es:

$$f(n) = f(c) + \max\{f(S_1), \ldots, f(S_k)\}$$

  - $f(c)$ considera el tiempo de comparación de "n" con cada uno de los casos case 1 ... case k

Consejo Instrucción "if"

- El tiempo de ejecución de una instrucción "if C then S_1 else S_2" es:

$$f(n) = f(C) + \max\{f(S_1), f(S_2)\}$$

```
if (n == 0)
  for (i = 0; i<n; i++)
    r += i;
else
  r = 2;
```

Consejo Instrucción "while"

- Tiempo de ejecución de la instrucción:

```
while (c) {
    S
}
```

es definido por:

$$f(n) = f(c) + (\#\text{iteraciones})*(f(c)+f(s))$$

- Nota: las instrucciones for, repeat, loop son equivalentes a una instrucción while

Consejo Instrucción “while”


```
for (i = 0; i <= n; i++)
  {
    S;
  }
```

```
i = 1;
while (i <= n)
  {
    S;
    i++;
  }
```

Consejo Llamado a Funciones NO Recursivas

- El tiempo de ejecución de una llamada a una función $F\,(A_1, \ldots, A_s)$ es:
  - 1, por el llamado a la función, más
  - Tiempo de evaluación de los parámetros $A_1, \ldots, A_s$
  - Tiempo de ejecución de la función (s)

$$f(A_1, \ldots, A_s) = 1 + f(A_1) + \ldots + f(A_s) + f(s)$$

Consejo para Funciones Recursivas

- El tiempo de ejecución de una función recursiva se calcula a través de la solución de funciones de recurrencia (siguiente tema)

Tarea

- Ejercicios 2.1-3, 2.2-2 (del Cormen, Leiserson, Rivest, Stein)
- ¿Cuáles de las siguientes afirmaciones es cierta?
  - $n^2 \in O(n^3)$
  - $2^{n+1} \in O(2^n)$
  - $n^2 \in \Omega(n^3)$