Analisis y Diseño de Algoritmos
Introducción

Juan Gutiérrez

September 2019

1 Motivación y algunas definiciones

Definición 1. *Algoritmo: Procedimiento computacional bien definido que recibe un valor o conjunto de valores como entrada y produce un valor, o conjunto de valores como salida.*

Definición 2. *Problema computacional: relación esperada entre una posible entrada y una posible salida.*

En ese sentido, un algoritmo es un procedimiento que consigue resolver un problema computacional. Veamos el siguiente ejemplo.

Ejemplo 1. *Problema de ordenación.*
Entrada: Secuencia de n números $\langle a_1, a_2, \ldots, a_n \rangle$.
Salida: Una permutación $\langle a'_1, a'_2, \ldots, a'_n \rangle$ de la secuencia de entrada, tal que $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

Definición 3. *Instancia de un problema: caso particular del problema.*

En el ejemplo, una instancia es $\langle 31, 41, 59, 26, 41, 58 \rangle$. Un algoritmo está *correcto* si para cada entrada, devuelve la correspondiente salida. En ese caso, decimos que el algoritmo *resuelve* el problema

Definición 4. *Análisis de Algoritmos: Estudio de la cantidad de recursos computacionales, tiempo y memoria, que los algoritmos consumen.*

Se busca identificar aspectos estructurales comunes a algoritmos diferentes, al estudiar métodos y paradigmas de diseño de algoritmos. Estas técnicas son comunes al resolver problemas mucho más aplicados relacionados al genoma humano, internet, criptografia, programación lineal (optimización), etc.

Aprenderemos distintos paradigmas de diseño. Los principales a estudiar en este curso son:

- División y Conquista,
- Programación Dinámica,

- Algoritmos voraces.

Para problemas NP-dificiles, se realizan técnicas más avanzadas: heurísticas o algoritmos de aproximación. Sin embargo, estas técnicas más avanzadas tienen mucho en común con las técnicas clásicas a ver en el curso.

2 Introducción al análisis de eficiencia

Las métricas básicas para medir la eficiencia de un algoritmo son el tiempo de ejecución y memoria consumidos. En esta sección veremos un análisis informal del tiempo de ejecución de dos algoritmos.

Ejemplo 2. *Tenemos dos computadores A y B con las siguientes características y algoritmos a ejecutar.*

- *Computador A:*
  - *Velocidad:* 10^{10} *instrucciones/s,*
  - *Algoritmo a ejecutar:* InsertionSort*, cuya complejidad es* $2n^2$*. Es decir, con entrada de tamaño* n*, ejecuta* $2n^2$ *instrucciones.*
- *Computador B:*
  - *Velocidad:* 10^7 *instrucciones/s,*
  - *Algoritmo a ejecutar:* MergeSort*, cuya complejidad es* $50n \lg n$*.*

En el ejemplo anterior, suponiendo que ordenamos $n = 10^7$ números, tenemos que

- A demora $\frac{2(10^7)^2 inst}{10^{10} inst/s} = 20000s = 5.55hs$.
- B demora $\frac{50(10^7)\lg 10^7 inst}{10^7 inst/s} = 1162.67s = 0.322hs$.

En el mismo ejemplo, si ordenamos $n = 10^8$ números, A demora 23 días, pero B demora 4 hrs.

Podemos concluir, que, mientras más grande es el valor de n, el algoritmo MergeSort se comporta mucho mejor que el algoritmo de InsertionSort, dejando de tener relevancia la velocidad del procesador y las constantes involucradas en el tiempo de ejecución. Es decir, las expresiones relevantes son $2n^2$ para InsertionSort y $50n \lg n$ para MergeSort.

Ejercicio 1. *Suponga que en una misma computadora se corre Insertion sort y* MergeSort*, donde* InsertionSort *tiene complejidad* $8n^2$ *y* MergeSort *tiene complejidad* $64n \lg n$*. ¿Para que valores de* n *es más eficiente el* InsertionSort*?*

n	$8\lg n$
1	0
2	8
3	12.67
10	26.57
20	34.57
30	39,..
40	42.57
43	43.4
44	43.67
$\ldots$	$\ldots$

Podemos tabular valores y notar que la respuesta es $n \in \{2, \ldots, 43\}$.

A continuación una demostración mucho más formal. Decimos que una función sobre los números naturales es *creciente* si $f(x) \leq f(x+1)$ para todo x en el dominio de f.

Proposición 1. *La función $f(n) = n/\lg n$ sobre los números naturales a partir de 3 es una función creciente.*

Proof. Demostraremos, por inducción en n, que $f(n) \leq f(n+1)$ para todo número natural $n \geq 3$. Si $n = 3$, tenemos que $f(3) = 3/\lg 3 < 2 = 4/\lg 4 = f(4)$, por lo tanto la proposición es válida. Suponga ahora que $n \geq 4$. Por hipótesis de inducción, tenemos que

$$\frac{n-1}{\lg(n-1)} \leq \frac{n}{\lg n}.$$

Por lo tanto,

$$n = \frac{1}{1 - \frac{n-1}{n}} \leq \frac{1}{1 - \frac{\lg(n-1)}{\lg n}} = \frac{\lg n}{\lg n - \lg(n-1)} = \frac{\lg n}{\lg(n/(n-1))}.$$

Luego,

$$\begin{aligned}
\frac{n}{n+1} &= 1 - \frac{1}{n+1} \\
&\leq 1 - \frac{1}{\frac{\lg n}{\lg(n/(n-1))} + 1} \\
&\leq 1 - \frac{\lg(n/(n-1))}{\lg(n^2/(n-1))} \\
&= \frac{\lg n}{\lg(n^2/(n-1))} \\
&\leq \frac{\lg n}{\lg(n+1+\frac{1}{n-1})} \\
&\leq \frac{\lg n}{\lg(n+1)}.
\end{aligned}$$

$\square$

Proposición 2. *Para todo número natural n, $8n^2 < 64n \lg n$ si y solo si $2 \leq n \leq 43$.*

Proof. Si $n = 1$ entonces $8n^2 = 8 > 64n \lg n$. Supongamos entonces que $n \geq 2$. Por la Proposición 1, $n/\lg n$ es una función creciente. Luego, para $n \leq 43$, tenemos que $\frac{n}{\lg n} \leq \frac{43}{\lg 43} < 8$. Y para $n \geq 44$, tenemos que $\frac{n}{\lg n} \geq \frac{44}{\lg 44} > 8$. Luego, $8n^2 < 64n \lg n$ si y solo si $2 \leq n \leq 43$. $\square$

Un corolario importante de la Proposición 1 es el siguiente.

Corolario 1. *Para todo número natural n, se cumple que $n > \lg n$.*

Proof. Si $n = 1$, tenemos que $n = 1 > 0 = \lg 1$. Si $n \geq 2$, tenemos que, por la Proposición 1, $\frac{n}{\lg n} \geq \frac{2}{\lg 2} = 2$. Luego $n \geq 2 \lg n > \lg n$. $\square$

Observación 1. *Notación de logaritmos: A lo largo del curso usaremos las siguientes notaciones.*

- $\lg x := \log_2 x$
- $\log x := \log_{10} x$

Ejercicio 2. *¿Cual es el menor valor de n tal que un algoritmo con tiempo de ejecucion $100n^2$ es más rapido que uno con tiempo de ejecucion 2^n en la misma máquina?*

$\lg 100n^2$	n
6.64...	1
8.64 ..	2
9.813 ..	3
13.2877..	10
...	...
14.25..	14
14.457..	15

Podemos tomar logaritmo en base 2 a ambas expresiones y tabular los correspondientes valores. Notamos que el valor pedido es $n = 15$. La demostración formal queda como ejercicio propuesto.

Ejercicio 3. *Demuestre formalmente que $2^n/100n^2$ es una función creciente. Concluya la respuesta del ejercicio anterior.*

Ejercicio 4. *Para cada función $f(n)$ y tiempo t en la siguiente tabla, determine el mayor tamaño de n de un problema que puede ser resuelto en tiempo t, suponiendo que el algoritmo para resolver el problema toma $f(n)$ μs*

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
$\lg n$							
$\sqrt{n}$							
n							
$n \lg n$							
n^2							
n^3							
2^n							
$n!$							

Observación 2. *c Propiedades básicas de logaritmos.*

- $\log_a b = x$ *si y solo si* $b = a^x$. *(Definición de logaritmo).*
- $\log_a(x^y) = y \log_a(x)$.
- $\log_a(xy) = \log_a(x) + \log_a(y)$.

Las siguientes propiedades pueden demostrarse a partir de las anteriores.

- $\log_a b = 1/\log_b a$.
- $\log_a b = \log_a c \cdot \log_c a$.
- $\log_a(x/y) = \log_a(x) - \log_a(y)$.
- $a^{\log_b c} = c^{\log_b a}$.
- $a^{\log_a b} = b$.

3 La técnica de inducción

Es una técnica de demostración muy usada para demostrar propiedades sobre secuencias discretas. A continuación veremos tres aplicaciones de esta técnica.

Propiedad 1. *Para todo número natural n,* $1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$.

Proof. Por inducción en n. Cuando $n = 1$, tenemos que $1 + 2 + \ldots + n = 1 = \frac{1 \cdot 2}{2} = n(n+1)/2$ (caso base).

Cuando $n > 1$, por hipótesis de inducción, tenemos que

$$1 + 2 + \cdots + n - 1 = \frac{(n-1)n}{2}.$$

Luego, $1+2+\ldots+n = (1+2+\ldots+n-1)+n = \frac{(n-1)n}{2}+n = n(\frac{n-1}{2}+1) = \frac{n(n+1)}{2}$. □

Propiedad 2. *Para todo número natural n, $1 + 2 + 2^2 + \cdots 2^n = 2^{n+1} - 1$.*

Proof. Por inducción en n. Cuando $n = 1$, tenemos que $1 + 2 + 2^2 + \ldots + 2^n = 1 + 2 = 3 = 2^2 - 1 = 2^{n+1} - 1$ (caso base). Cuando $n > 1$, por hipótesis de inducción, tenemos que

$$1 + 2 + 2^2 + \cdots 2^{n-1} = 2^n - 1.$$

Luego, $1 + 2 + 2^2 + \cdots + 2^n = (1 + 2 + 2^2 + \cdots 2^{n-1}) + 2^n = (2^n - 1) + 2^n = 2^n + 2^n - 1 = 2(2^n) - 1 = 2^{n+1} - 1$. □

Propiedad 3. *Para todo número natural n, $\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = \frac{n}{n+1}$..*

Proof. Por inducción en n. Cuando $n = 1$, tenemos que $\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{n(n+1)} = \frac{1}{2} = \frac{n}{n+1}$ (caso base). Cuando $n > 1$, por hipótesis de inducción, tenemos que

$$\frac{1}{2} + \frac{1}{6} + \cdots + \frac{1}{(n-1)n} = \frac{n-1}{n}$$

Luego, $\frac{1}{2}+\frac{1}{6}+\cdots+\frac{1}{n(n+1)} = (\frac{1}{2}+\frac{1}{6}+\cdots+\frac{1}{(n-1)n})+\frac{1}{n(n+1)} = \frac{n-1}{n}+\frac{1}{n(n+1)} = \frac{(n-1)(n+1)+1}{n(n+1)} = \frac{(n^2-1)+1}{n(n+1)} = \frac{n^2}{n(n+1)} = \frac{n}{n+1}$. □

Propiedad 4. *Para todo número natural $n \geq 1$, $\lg n \leq n$.*

Proof. El enunciado equivale a probar que $n \leq 2^n$ para todo número natural $n \geq 0$. Lo probaremos por inducción en n. Cuando $n = 0$, tenemos que $0 = n \leq 2^0 = 1$. (caso base). Cuando $n > 0$, por hipótesis de inducción, tenemos que

$$n - 1 \leq 2^{n-1}.$$

Además, es claro que $2^{n-1} \geq 1$ (si queremos ser más precisos, deberíamos probar este enunciado también por inducción), luego

Luego, $n \leq 2^{n-1} + 1 \leq 2^{n-1} + 2^{n-1} = 2^n$. □

Propiedad 5. *Para todo número natural $n \geq 44$, $8 \lg n \leq n$.*

Proof. El enunciado equivale a probar que $n \leq 2^{n/8}$ para todo número natural $n \geq 44$. Lo probaremos por inducción en n. Cuando $n = 44$, tenemos que $n^8 \leq 2^n$. (caso base). Cuando $n > 44$, por hipótesis de inducción, tenemos que

$$(n-1) \leq 2^{(n-1)/8}.$$

Además, como $n \geq 45$ tenemos que $n/8 \geq 5$, por lo tanto, $2^{-n/8} \leq 2^{-5}$. Luego,

$$\begin{aligned} n &\leq 2^{(n-1)/8} + 1 \\ &\leq 2^{n/8}(2^{-1/8} + 2^{-n/8}) \\ &\leq 2^{n/8}(2^{-1/8} + 2^{-5}) \\ &\leq 2^{n/8}, \end{aligned}$$

como queríamos demostrar. □

4 Sumatorias

Cuando un algoritmo tiene una instrucción iterativa (for o while), podemos expresar su tiempo de ejecución como la suma de los tiempos en cada iteración. Es importante conocer conceptos de sumatorias y encontrar límites superiores para ellas (upper bounds).

4.1 Fórmulas y propiedades básicas

Definición 5. *Dada una secuencia $a_1, a_2, \ldots, a_n$ de números, donde n es un entero no negativo, podemos escribir la suma finita $a_1 + a_2 + \cdots + a_n$ como*

$$\sum_{k=1}^{n} a_k.$$

Si $n = 0$ entonces el valor de la sumatoria es definida como 0.

Definición 6. *Dada una secuencia infinita $a_1, a_2, \ldots$ de números, podemos escribir la suma infinita $a_1 + a_2 + \cdots$ como*

$$\sum_{k=1}^{\infty} a_k = \lim_{n \to \infty} \sum_{k=1}^{n} a_k.$$

Si el límite no existe, la serie diverge, si no, esta converge.

Propiedad de Linealidad

Para cada número real c y cualesquier secuencias $a_1, a_2, \ldots, a_n$ y $b_1, b_2, \ldots, b_n$, tenemos que

$$\sum_{k=1}^{n} (ca_k + b_k) = c \sum_{k=1}^{n} a_k + \sum_{k=1}^{n} b_k.$$

Series aritméticas

Son series en donde la resta de cada dos términos consecutivos es la misma. Ejemplo:

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}.$$

Suma de cuadrados y cubos

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6}.$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4} = (\frac{n(n+1)}{2})^2.$$

Series geométricas

Son series en donde la división de cada dos términos consecutivos es la misma.

Por ejemplo, para cada número real $x \neq 1$,

$$\sum_{k=0}^{n} x^k = 1 + x + x^2 + \cdots + x^n = \frac{x^{n+1} - 1}{x - 1}.$$

Note que cuando $x = 2$, tenemos

$$1 + 2 + 2^2 + \ldots + 2^n = 2^{n+1} - 1.$$

(O también $2^n = 1 + 2 + 2^2 + \cdots + 2^{n-1} + 1$).

Cuando $|x| < 1$, se cumple

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}.$$

De lo anterior, podemos concluir que (ejercicio)

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1 - x)^2}.$$

(Suponemos que $0^0 = 1$).

Serie ármónica

Es la serie

$$H_n = \sum_{k=1}^{n} \frac{1}{k} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n}.$$

$H(n)$ es llamado *número armónico.* Se sabe que $H(n) = \ln n + c$ para una constante $c \approx 0.5772$.

Series telescópicas

Para cada secuencia $a_0, a_1, \ldots, a_n$,

$$\sum_{k=1}^{n}(a_k - a_{k-1}) = a_n - a_0.$$

También,

$$\sum_{k=0}^{n-1}(a_k - a_{k-1}) = a_0 - a_n.$$

Por ejemplo,

$$\sum_{k=1}^{n-1}\frac{1}{k(k+1)} = \sum_{k=1}^{n-1}(\frac{1}{k} - \frac{1}{k+1}) = 1 - \frac{1}{n}.$$

Convirtiendo productorias a sumatorias

Podemos escribir el producto finito $a_1 a_2 \cdots a_n$ como $\prod_{k=1}^{n} a_k$ (cuando $n = 0$ el valor de producto es definido como 1).

Podemos hacer la siguiente conversión:

$$\lg(\prod_{k=1}^{n} a_k) = \sum_{k=1}^{n} \lg a_k.$$

4.2 Acotando sumatorias

Existen muchas técnicas para acotar sumatorias. Estas nos servirán al acotar las sumatorias que expresan los tiempos de ejecución de un algoritmo.

Inducción

Probaremos por inducción en n que

$$\sum_{k=1}^{n} k \leq \frac{1}{2}(n+1)^2.$$

Es fácil ver que la propiedad es cierta para $n = 1$ (caso base). Asumiremos por hipótesis de inducción que funciona para $n-1$. Osea, $\sum_{k=1}^{n-1} k \leq \frac{1}{2}(n-1+1)^2$.

Luego

$$
\begin{aligned}
\sum_{k=1}^{n} &= \sum_{k=1}^{n-1} + n \\
&\leq \frac{1}{2}n^2 + n \text{ (por hipótesis de induccion)} \\
&\leq \frac{1}{2}n^2 + n + \frac{1}{2} \\
&= \frac{1}{2}(n+1)^2.
\end{aligned}
$$

Probaremos por inducción en n que

$$\sum_{k=0}^{n} 3^k \leq c3^n$$

para alguna constante c. Es fácil ver que cuando $n = 0$, tenemos que $\sum_{k=0}^{n} 3^k = 1$. Luego para cualquier $c \geq 1$ la proposición es cierta. Por hipótesis de inducción, tenemos que existe c tal que $\sum_{k=0}^{n-1} 3^k \leq c3^{n-1}$. Luego, cuando $c \geq 3/2$, tenemos que

$$
\begin{aligned}
\sum_{k=0}^{n} 3^k &= \sum_{k=0}^{n-1} 3^k + 3^n \\
&\leq c3^{n-1} + 3^n \\
&= (\frac{1}{3} + \frac{1}{c})c3^n \\
&\leq c3^n.
\end{aligned}
$$

Portanto, basta tomar $c = 3/2$.

Observación: si bien es cierto, la manera de demostrar anterior es correcta, es recomendable organizar la demostración dando directamente el valor de la constante, de la siguiente manera.

Probaremos por inducción en n que

$$\sum_{k=0}^{n} 3^k \leq \frac{3}{2}3^n.$$

Cuando $n = 0$, tenemos que $\sum_{k=0}^{n} 3^k = 1 \leq \frac{3}{2} \cdot 3^0$ y por lo tanto la afirmación es válida. Cuando $n > 0$, por hipótesis de inducción, tenemos que

$\sum_{k=0}^{n-1} 3^k \leq \frac{3}{2} \cdot 3^{n-1}$. Luego,

$$\begin{aligned}
\sum_{k=0}^{n} 3^k &= \sum_{k=0}^{n-1} 3^k + 3^n \\
&\leq \frac{3}{2} \cdot 3^{n-1} + 3^n \\
&= (\frac{1}{3} + \frac{2}{3})\frac{3}{2} \cdot 3^n \\
&\leq \frac{3}{2} \cdot 3^n.
\end{aligned}$$

Acotando términos

Podemos acotar superiomente cada término de una serie, por ejemplo

$$\sum_{k=1}^{n} k \leq \sum_{k=1}^{n} n = n^2$$

En general, sea $a_{\max} = \max_{1 \leq k \leq n} a_k$, tenemos que

$$\sum_{k=1}^{n} a_k \leq \sum_{k=1}^{n} a_{\max} = n \cdot a_{\max}$$

Supongamos que $a_{k+1}/a_k \leq r$ para todo $k \geq 0$, con $0 < r < 1$. Tenemos que

$$\begin{aligned}
\sum_{k=0}^{n} a_k &\leq \sum_{k=0}^{\infty} a_0 r^k \\
&= a_0 \sum_{k=0}^{\infty} r^k \\
&= a_0 \cdot \frac{1}{1-r}.
\end{aligned}$$

Por ejemplo, acotaremos $\sum_{k=1}^{\infty}(k/3^k) = \sum_{k=0}^{\infty}((k+1)/3^{k+1})$. Tenemos

$$a_0 = 1/3,$$

$$\frac{(k+2)/3^{k+2}}{(k+1)/3^{k+1}} = \frac{1}{3} \cdot \frac{k+2}{k+1} \leq \frac{2}{3} = r.$$

Luego

$$\sum_{k=1}^{\infty}(k/3^k) \leq \frac{1}{3} \cdot \frac{1}{1-2/3} = 1$$

Dividiendo sumatorias

Acotaremos $\sum_{k=1}^{n} k$. Note que (asuma que n es par)

$$\begin{aligned}\sum_{k=1}^{n} k &= \sum_{k=1}^{n/2} k + \sum_{k=n/2+1}^{n} k \\ &\geq \sum_{k=1}^{n/2} 0 + \sum_{k=n/2+1}^{n} (n/2) \\ &= (n/2)^2.\end{aligned}$$

Para cualquier constante $k_0 > 0$, tenemos que

$$\begin{aligned}\sum_{k=0}^{n} a_k &= \sum_{k=0}^{k_0-1} a_k + \sum_{k=k_0}^{n} a_k \\ &= c + \sum_{k=k_0}^{n} a_k.\end{aligned}$$

donde c es una constante.

Acotaremos $\sum_{k=0}^{\infty} k^2/2^k$. Note que cuando $k \geq 3$, $\frac{(k+1)^2/2^{k+1}}{k^2/2^k} = \frac{(k+1)^2}{2k^2} \leq \frac{8}{9}$. Luego

$$\begin{aligned}\sum_{k=0}^{\infty} k^2/2^k &= \sum_{k=0}^{2} k^2/2^k + \sum_{k=3}^{\infty} k^2/2^k \\ &\leq \sum_{k=0}^{2} \frac{k^2}{2^k} + 9/8 \sum_{k=0}^{\infty} (8/9)^k \\ &= \sum_{k=0}^{2} \frac{k^2}{2^k} + 9/8 \cdot (1/(1-8/9)) \\ &= c\end{aligned}$$

para alguna constante c.

Acotaremos $H_n = \sum_{k=1}^{n} \frac{1}{k}$. Note que

$$\begin{aligned}\sum_{k=1}^{n} \frac{1}{k} &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i + j} \\ &\leq \sum_{i=0}^{\lfloor \lg n \rfloor} \sum_{j=0}^{2^i-1} \frac{1}{2^i} \\ &= \sum_{i=0}^{\lfloor \lg n \rfloor} 1 \\ &\leq \lg n + 1\end{aligned}$$

4.3 Árboles binarios

Un *árbol binario* T es definido recursivamente según:

- si T no tiene nodos entonces es un árbol binario
- si T tiene nodos, está compuesto por un nodo *raíz*, un árbol binario llamado *subárbol izquierdo* y un árbol binario llamado *subárbol derecho*

El árbol binario sin nodos es llamado *nulo* o *vacío* y denotado por NIL. Si el subárbol izquierdo es no vacío, su raíz es llamada *hijo izquierdo*. Análogamente definimos *hijo izquierdo*.

Si cada nodo tiene exactamente dos hijos, un árbol binario es denominado *lleno*. Si todas las hojas tienen la misma profundidad, dicho árbol es denominado *completo*.

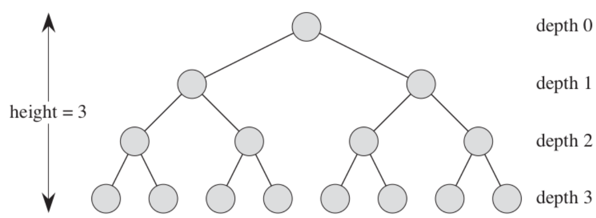

Figure B.8 A complete binary tree of height 3 with 8 leaves and 7 internal nodes.

Figure 1: Tomada del libro Cormen, Introduction to Algorithms

La *altura* de un nodo es el tamaño máximo de un camino que va de manera descendente desde dicho nodo hacia alguna hoja. La *altura* de un árbol es igual a la altura de la raíz del dicho árbol.

Propiedad 6. *La altura de un árbol binario completo con k hojas es* $\lg k$.

Propiedad 7. *Propiedad: La altura de un árbol binario completo con n vértices es* $\lg(n+1) - 1$