Estructura de Datos y Algoritmos
Departamento de electrónica
Universidad Técnico Federico Santa María

DEPARTAMENTO DE ELECTRONICA

Ayudantía 8: Notación asintótica, árboles de recursividad y método maestro

Objetivo Principal: Conocer cómo analizar y determinar la complejidad de algoritmos a través de los métodos de árbol de recursividad y método maestro.
Autor: Kevin Pizarro Aguirre

---

1. Notación Asintótica

La notación asintótica es aquel lenguaje que nos permitan analizar el tiempo de ejecución de un algoritmo. Permite identificar su comportamiento o tasa de crecimiento, en base al tamaño de la entrada. Es una herramienta poderosa que nos permite abstraernos de los términos menos relevantes y poder comparar fácilmente algoritmos de la misma naturaleza, por ejemplo los algoritmos de ordenamientos. Se tienen 3 formas de expresar la notación asintótica: con notación notación Big-Ω, notación Big-O y Big-Θ.

1.1. Notación Big-Ω

Nos permite denotar la cota inferior de nuestro tiempo de ejecución, es decir representa el mejor caso (no es de interés). Matemáticamente se define como la ecuación (1).

$$\Theta(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq cg(n) \leq f(n)\} \tag{1}$$

Siendo f(n) el tiempo de ejecución real, g(n) la cota inferior y n la cantidad de datos de entrada.

1.2. Notación Big-O

Nos permite denotar la cota superior de nuestro tiempo de ejecución, es decir representa el peor caso (sí es de interés). Matemáticamente se define como la ecuación (2).

$$O(g(n)) = \{f(n) : \exists c > 0, n_0 > 0 \mid 0 \leq f(n) \leq cg(n), \forall n \geq n_0\} \tag{2}$$

1.3. Notación Big-Θ

Nos permite denotar las cotas inferior y superior de nuestro tiempo de ejecución, es decir es la intersección entre la notación Big-Ω y la notación Big-O. Matemáticamente se define como la ecuación (3).

$$\Theta(g(n)) = \{f(n) : \exists c_1, c_2, n_0 > 0 \mid c_1 g(n) \leq f(n) \leq c_2 g(n)\} \tag{3}$$

En la figura (1) se ve gráficamente cómo funciona cada notación asintótica. Dentro de las notaciones asintóticas más usadas tenemos a la Big-Θ y Big-O.

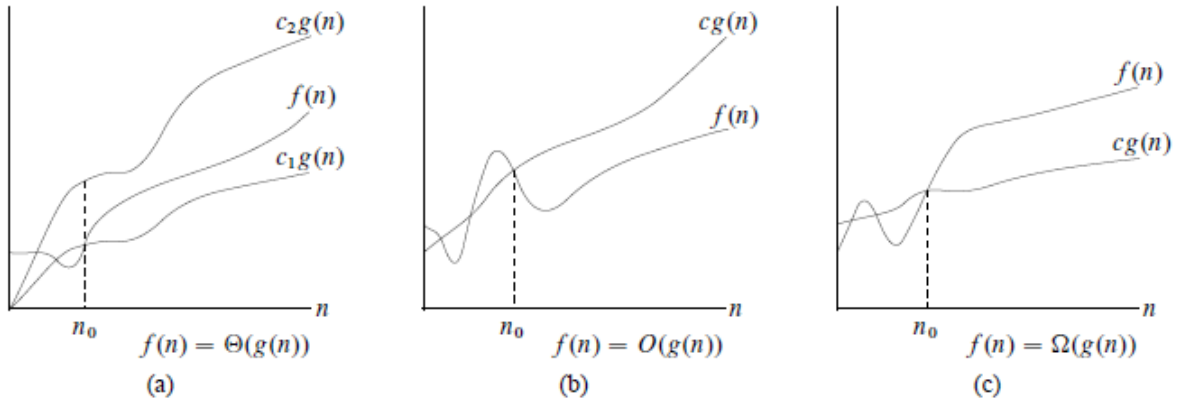


Figura 1: Gráficas de notaciones asintóticas: (a) Big-Θ, (b) Big-O y (c) Big-Ω.

Cuando queremos analizar algoritmos tantos casos diferentes como algoritmos hay en el mundo. Aún así hay intuiciones que nos permiten construir principios básicos y otros que serán más elaborados, algunas de ellas se muestran en la siguiente tabla.[1]

Función/Algoritmo	**Descripción**	**Notación asintótica**
A(n)	Compuesto por 1 bucle que se repite n veces.	O(n) o Θ(n)
B(m)	Compuesto por 1 bucle que se repite m veces.	O(n) o Θ(n)
C(n,m) = A(n)+B(m)	Suma de dos bucles que se repiten n y m veces respectivamente.	Si n≥ m: O(n) o Θ(n) Si m>n: O(m) o Θ(m)
C(n,m) = (AoB)(n,m)	"Composición" de los algoritmos. Un bucle m dentro del bucle n.	O(n*m) o Θ(n*m) Si n=m: O(n^2) o Θ(n^2)

Cuadro 1: Análisis de algoritmos.

Cabe destacar que la notaciones de operaciones pueden no ser correctas, mas lo importante es la intuición y el resultado en la notación asintótica obtenida. Por ejemplo al tener dos bucles anidados, uno dentro del otro, si son del mismo largo entonces se tendrá una complejidad de O(n^2). Otro ejemplo es si se tienen bloques de tiempos diferentes la complejidad estará dada por el de mayor tasa de crecimiento.

Para otros casos de interés, como el análisis de funciones recursivas, se utilizan 2 métodos principalmente: los árboles de recursividad y el método maestro. Ambos abordados a continuación.

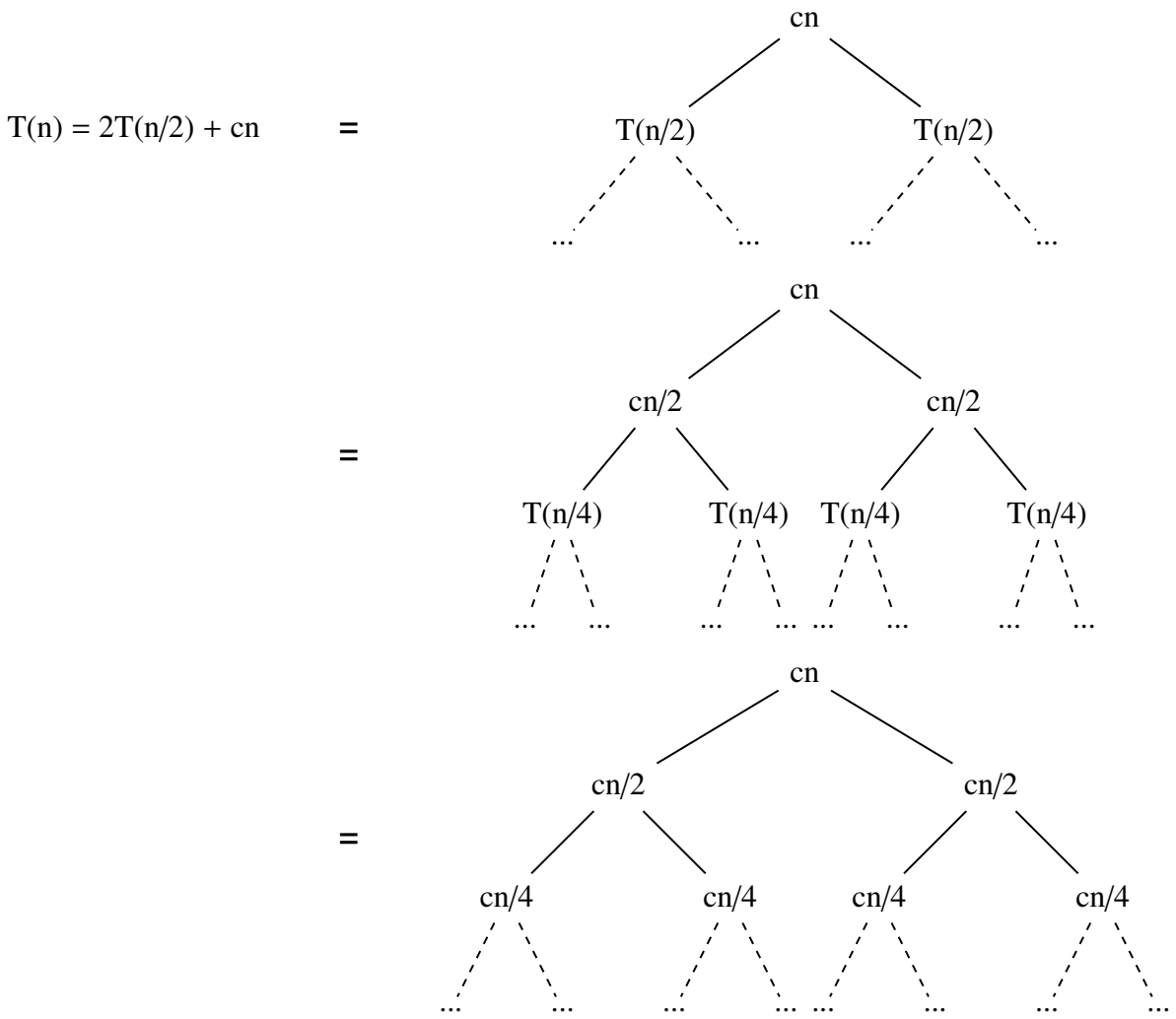
[1]Otros casos, no tan comunes, pueden ser consultados en el libro guía: Introduction to Algorithms, Third Edition, page 53.

2. Árboles de recursividad

Los árboles de recursividad nos proveen una herramienta para poder analizar algoritmos recursivos, primero gráficamente y luego matemáticamente hasta llegar a la notación asintótica.

Sea una función T(n) que representa el tiempo de ejecución de nuestro algoritmo, definida por la ecuación (4), en otra ayudantía se verá que corresponde a el algoritmo *Merge Sort* en particular. Notar que por cómo está definida se trata de una función recursiva, se llama a si misma una cantidad X de veces. Por lo cual se puede representar a gráficamente a través del árbol de recursividad que se muestra a continuación.

$$T(n) = 2T(n/2) + \Theta(n) = 2T(n/2) + cn \tag{4}$$



Traduciendo la información gráfica a términos matemáticos se llegan a los siguientes resultados.

1. La cantidad de niveles en el árbol está dado por $\log_2(n)$. ¿cuántas veces puedo dividir a la mitad algo de largo n?
2. La cantidad de hojas del árbol es n. Si divido algo de largo n a la mitad X veces hasta tener su mínima expresión, ¿cuántas hojas tendremos?
3. La sumatoria de todos los valores de los nodos es $cnlog_2n$. Tip: Realizar sumatoria de los nodos por **nivel** y luego la sumatoria de los resultados obtenidos.

Como la sumatoria de todos los nodos es en realidad T(n), entonces podemos expresar el resultado final a través de notación asintótica definida en (5).

$$T(n) = \Theta(n \log(n)) \tag{5}$$

3. Método maestro

El método maestro[2] es una "receta" para poder analizar los algoritmos del tipo $T(n) = aT(n/b) + f(n)$, donde $a \geq 1$, $b > 1$ y $f(n)$ es positiva asintóticamente (positiva para todo $n \geq n_0$). A continuación los casos y sus resultados.

1. Si $f(n) = O(n^{\log_b(a)-\epsilon})$ para algún $\epsilon > 0$, entonces $T(n) = \Theta(n^{\log_b(a)})$.
2. Si $f(n) = \Theta(n^{\log_b(a)} \log^k(n))$ para algún $k \geq 0$, entonces $T(n) = \Theta(n^{\log_b(a)} \log^{k+1}(n))$.
3. Si $f(n) = \Omega(n^{\log_b(a)+\epsilon})$ para algún $\epsilon > 0$, y además $af(n/a) \leq (1 - \epsilon')f(n)$ con $\epsilon' > 0$, entonces $T(n) = \Theta(f(n))$.

Ejemplo: Para Merge Sort, dada la ecuación (4), tenemos que $a = 2 \geq 1$ y $b = 2 > 1$ por lo cual $n^{\log_2 2} = n$ y también sabemos que $f(n) = \Theta(n)$, por lo tanto podemos aplicar el método maestro. Luego analizamos los casos para ver a cual pertenece.

1. Caso 1: Se descarta puesto que no existe algún $\epsilon > 0$ que nos permita lograr $f(n) = O(n^{1-\epsilon}) = \Theta(n)$. El valor de ϵ que cumple es 0 y está fuera de la restricción.
2. Caso 2: Sí corresponde puesto que se puede elegir $k = 0$. Finalmente $T(n) = \Theta(n \log(n))$.
3. Caso 3: Se descarta puesto que no existe algún $\epsilon > 0$ que nos permita lograr $f(n) = O(n^{1+\epsilon}) = \Theta(n)$. El valor de ϵ que cumple es 0 y está fuera de la restricción.

Se llega al mismo resultado que utilizando árbol de recursividad. El problema es que no siempre se puede ocupar método maestro porque no calzan con las condiciones para aplicar el método.

[2]Para más detalles revisar el libro guía: Introduction to Algorithms, Third Edition, page 93.

4. Ejercicios

4.1. Análisis asintótico

a) Analizar $T(n) = 9T(n/3) + \Theta(\sqrt{n^5})$

b) Analizar $T(n) = 3T(n/2) + \Theta(n^2)$

c) Analizar $T(n) = 4T(n/2) + \Theta(n^2 ln^2(n))$

d) Analizar $T(n) = T(2n/3) + \Theta(1)$

e) Analizar $T(n) = T(n/10) + T(9n/10) + \Theta(n)$

f) Analizar $T(n) = 4T(n/2) + n^2\sqrt{n}$