1. Notaciones Asintóticas

¿Qué es una notación asintótica?

Las notaciones asintóticas permiten expresar cómo crece una función en relación al tamaño de su entrada n, ignorando constantes y términos no dominantes. Sirven para analizar la eficiencia de algoritmos sin depender de la implementación.

1.1. Notación Big O - O(f(n))

La notación Big O representa una cota superior para el tiempo de ejecución o uso de recursos de un algoritmo. Se interpreta como 'el algoritmo tarda como máximo esto'.

Definición formal:

$T(n) = O(f(n))$ si existen constantes $c > 0$ y $n_0 > 0$ tal que $T(n) \leq c \cdot f(n)$ para todo $n \geq n_0$.

1.2. Notación Omega - Ω(f(n))

La notación Omega representa un limite inferior para el tiempo de ejecución. Se interpreta como 'el algoritmo tarda al menos esto'.

Definición formal:

$T(n) = \Omega(f(n))$ si existen constantes $c > 0$ y $n_0 > 0$ tal que $T(n) \geq c \cdot f(n)$ para todo $n \geq n_0$.

1.3. Notación Theta - Θ(f(n))

La notación Theta indica un limite exacto del crecimiento del algoritmo. Se interpreta como 'el algoritmo crece exactamente como f(n)'.

Definición formal:

$T(n) = \Theta(f(n))$ si existen constantes $c1, c2 > 0$ y $n_0 > 0$ tal que $c1 \cdot f(n) \leq T(n) \leq c2 \cdot f(n)$ para todo $n \geq n_0$.

1.4. Notación Asintótica Condicional

Este tipo de notación se usa cuando el comportamiento asintótico depende de una condición externa (como una conjetura matemática).

Ejemplos:

- Problemas de factorización: O(n^{1/4 + ε}) suponiendo la Hipótesis de Riemann.
- Problema SAT: Ω(2^n) si P ≠ NP.
- Quicksort: O(n log n) en el mejor caso, O(n^2) en el peor caso dependiendo del pivote.

1.5 Comparación de Notaciones

Big O: Limite superior - peor caso

Omega: Limite inferior - mejor caso

Theta: Limite exacta - caso promedio

Condicional: Bajo hipótesis externas - criptografía, teoría de la complejidad

Ejemplos de Código

1. O(n):	2. Ω(n):

def buscar_elemento(lista, x):
  for elemento in lista:
  if elemento == x:
  return True
  return False
	def encontrar_mayor(lista):
	  mayor = lista[0]
	  for x in lista:
	  if x > mayor:
	  mayor = x
	  return mayor

3. Θ(n):	

def sumar_lista(lista):
  total = 0
  for x in lista:
  total += x
  return total	