Lista 02 unidade 04

Universidade Federal de Goiás

Stephany Resque

1. Dizer que g(n) ∈ O(f(n)) significa que, para valores suficientemente grandes de n, g(n) cresce no máximo tão rápido quanto f(n), multiplicado por uma constante. É uma forma de expressar o limite superior da taxa de crescimento de g(n).
2. Dizer que g(n) ∈ Θ(f(n)) significa que g(n) cresce na **mesma ordem de grandeza** que f(n) para valores suficientemente grandes de n. Isso indica que f(n) é tanto um limite superior quanto inferior para g(n).
3. Dizer que g(n) ∈ Ω(f(n)) significa que g(n) cresce pelo menos tão rápido quanto f(n) para valores suficientemente grandes de n. Em outras palavras, f(n) é um **limite inferior assintótico** para g(n).
4. O algoritmo B (complexidade 2n) seria usado no lugar do A (complexidade n5) apenas se n for muito pequeno, pois 2n cresce muito mais rápido que n5 e se torna inviável para valores grandes.
5. Problemas com complexidade exponencial geralmente envolvem **combinatória** e **pesquisa exaustiva**, onde o número de possibilidades cresce exponencialmente com o tamanho da entrada. Dois exemplos são:
   1. **Problema do Caixeiro Viajante (TSP)**: Um caixeiro precisa visitar várias cidades exatamente uma vez e retornar à cidade de origem, minimizando a distância percorrida. À medida que o número de cidades aumenta, o número de rotas possíveis cresce exponencialmente, tornando o problema difícil de resolver por força bruta.
   2. **Problema da Satisfatibilidade Boolean (SAT)**: Determinar se existe uma atribuição de valores verdadeiro/falso para as variáveis que satisfaça uma fórmula booleana. Para fórmulas grandes, o número de combinações possíveis de valores cresce exponencialmente, tornando o problema intratável em casos gerais.

Esses problemas são comuns em áreas como logística, inteligência artificial e otimização.

1. 1. **Complexidade:** O(n)  
      O termo dominante é 2n, já que o termo constante 10 não influencia assintoticamente.
   2. **Complexidade:** O(n2)  
      O termo dominante é n2.
   3. **Complexidade:** O(n2)  
      O termo dominante é n2.
   4. **Complexidade:** O(n4)  
      O termo dominante é n4.
   5. **Complexidade:** O(n)  
      O termo dominante é 7n.
   6. **Complexidade:** O(n!)  
      O crescimento de n! é muito mais rápido que n2 ou termos constantes.
   7. **Complexidade:** O(n)  
      O termo dominante é 5000n, as constantes são ignoradas.
   8. **Complexidade:** O(1)  
      Essa é uma constante, portanto sua complexidade é constante.
2. A complexidade no pior caso do código é: O(N3).
3. A função de custo para o número de operações é: C(N)=N−1. A **ordem de complexidade** no pior caso é: O(N).
4. **Função de custo no pior caso:** C(N)=⌈log2(N)⌉. **Ordem de complexidade no pior caso:** O(logN). **Ordem de complexidade no melhor caso:** Ω(1)