Notação Big O, Omega e Theta

Prof. Me. Lucas Bruzzone

Aula 03

Revisão da Aula Anterior

O que já sabemos:

- Como contar operações básicas
- Complexidade Linear: $O(n)$
- Complexidade Logarítmica: $O(\log n)$
- Complexidade Quadrática: $O(n^2)$
- Análise espacial e temporal básica

Motivação

Problema: Como comparar algoritmos de forma precisa?

Exemplos que geram dúvidas:

- Algoritmo A: $T(n) = 3n + 5$
- Algoritmo B: $T(n) = 2n + 100$
- Algoritmo C: $T(n) = n^2$

Questões Práticas

Perguntas que surgem:

- Qual é melhor para n pequeno?
- Qual é melhor para n grande?
- Como classificar matematicamente?
- Como ignorar detalhes de implementação?
- Como focar no comportamento assintótico?

As notações formais resolvem essas questões!

As Três Notações Fundamentais

Família de Notações Assintóticas

- **Big O (O)**: Limite superior
- **Big Omega (Ω)**: Limite inferior
- **Big Theta (Θ)**: Limite apertado

Cada uma responde uma pergunta diferente!

Quando Usar Cada Notação?

Big O (O):

- Quando queremos garantias de **pior caso**
- "O algoritmo não vai demorar mais que..."
- Útil para análise de limites superiores

Big Omega (Ω):

- Quando queremos garantias de **melhor caso**
- "O algoritmo vai demorar pelo menos..."
- Útil para provar limites inferiores

Big Theta (Θ):

- Quando conhecemos o comportamento **exato**
- "O algoritmo demora exatamente da ordem de..."
- Mais informativo e preciso

Notação Big O - Definição Formal

Definição Matemática:

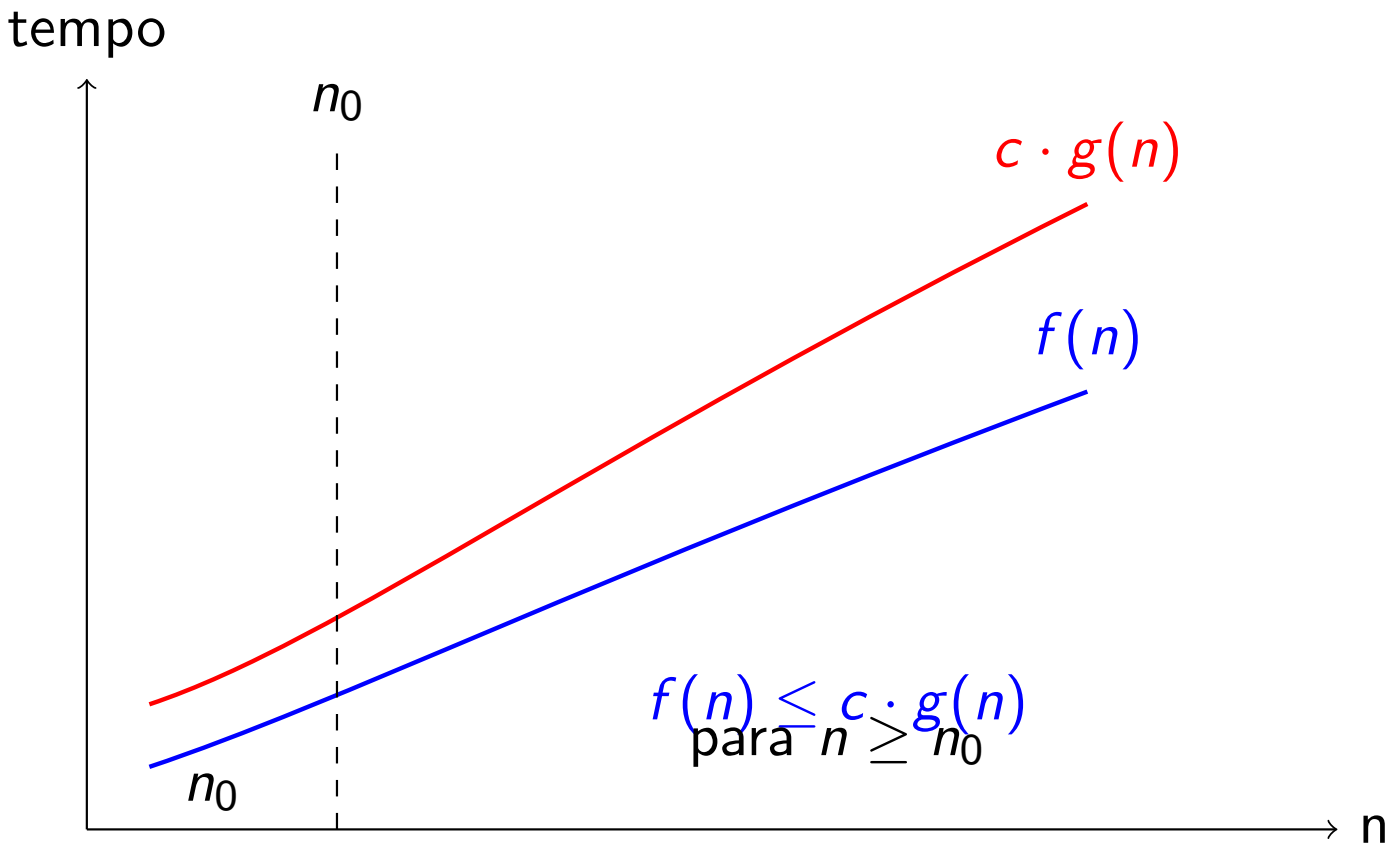
$f(n) = O(g(n))$ se e somente se existem constantes positivas c e n_0 tais que:

$$f(n) \leq c \cdot g(n) \text{ para todo } n \geq n_0$$

Interpretação:

- $g(n)$ é um **limite superior** para $f(n)$
- Para valores grandes de n, $f(n)$ não cresce mais rápido que $g(n)$
- As constantes c e n_0 podem ser escolhidas apropriadamente

Big O - Interpretação Visual



Região importante: À direita de n_0, onde $f(n) \leq c \cdot g(n)$

Exemplo 1 - Demonstrando Big O

Provar que $3n + 5 = O(n)$

Precisamos encontrar: c e n_0 tais que $3n + 5 \leq c \cdot n$ para $n \geq n_0$

Estratégia: Escolher c e n_0 apropriados

Tentativa 1: $c = 4$

$$3n + 5 \leq 4n$$

$$5 \leq n$$

Solução: $c = 4$ e $n_0 = 5$

Conclusão: A partir de n $= 5$, meu algoritmo nunca vai demorar mais do que 4 vezes o tempo de um algoritmo linear simples

Verificação do Exemplo 1

Verificação: Para $n \geq 5$: $3n + 5 \leq 3n + n = 4n$

Teste com valores específicos:

- $n = 5$: $3(5) + 5 = 20 \leq 4(5) = 20$
- $n = 10$: $3(10) + 5 = 35 \leq 4(10) = 40$
- $n = 100$: $3(100) + 5 = 305 \leq 4(100) = 400$

Conclusão: $3n + 5 = O(n)$ com $c = 4$ e $n_0 = 5$

Notação Big Omega - Definição Formal

Definição Matemática:

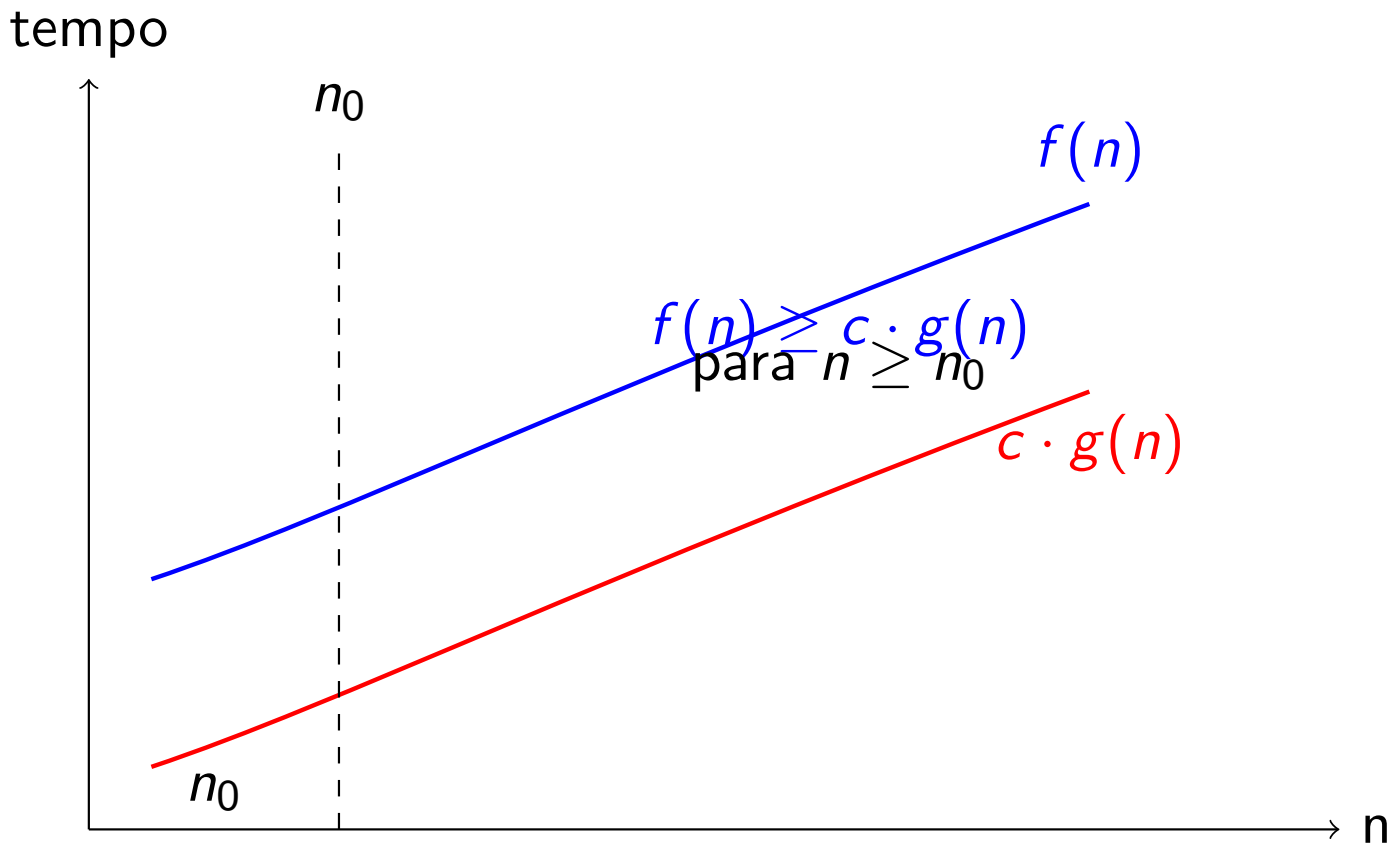
$f(n) = \Omega(g(n))$ se e somente se existem constantes positivas c e n_0 tais que:

$$f(n) \geq c \cdot g(n) \text{ para todo } n \geq n_0$$

Interpretação:

- $g(n)$ é um **limite inferior** para $f(n)$
- Para valores grandes de n, $f(n)$ não cresce mais devagar que $g(n)$
- $f(n)$ é "pelo menos tão rápida"quanto $g(n)$

Big Omega - Interpretação Visual



Região importante: À direita de n_0, onde $f(n) \geq c \cdot g(n)$

Exemplo 2 - Demonstrando Big Omega

Provar que $3n + 5 = \Omega(n)$

Precisamos encontrar: c e n_0 tais que $3n + 5 \geq c \cdot n$ para $n \geq n_0$

Estratégia: Para n grande, o termo dominante é $3n$

Escolha: $c = 3$

$$3n + 5 \geq 3n$$

Solução: $c = 3$ e $n_0 = 1$ (qualquer valor positivo)

Verificação: Para $n \geq 1$: $3n + 5 \geq 3n$ sempre

Conclusão: Meu algoritmo SEMPRE vai demorar pelo menos tanto quanto um algoritmo linear simples. Ele nunca será mais rápido que linear

Notação Big Theta - Definição Formal

Definição Matemática:

$f(n) = \Theta(g(n))$ se e somente se existem constantes positivas c_1, c_2 e n_0 tais que:

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ para todo } n \geq n_0$$

Equivalência:

$$f(n) = \Theta(g(n)) \iff f(n) = O(g(n)) \text{ E } f(n) = \Omega(g(n))$$

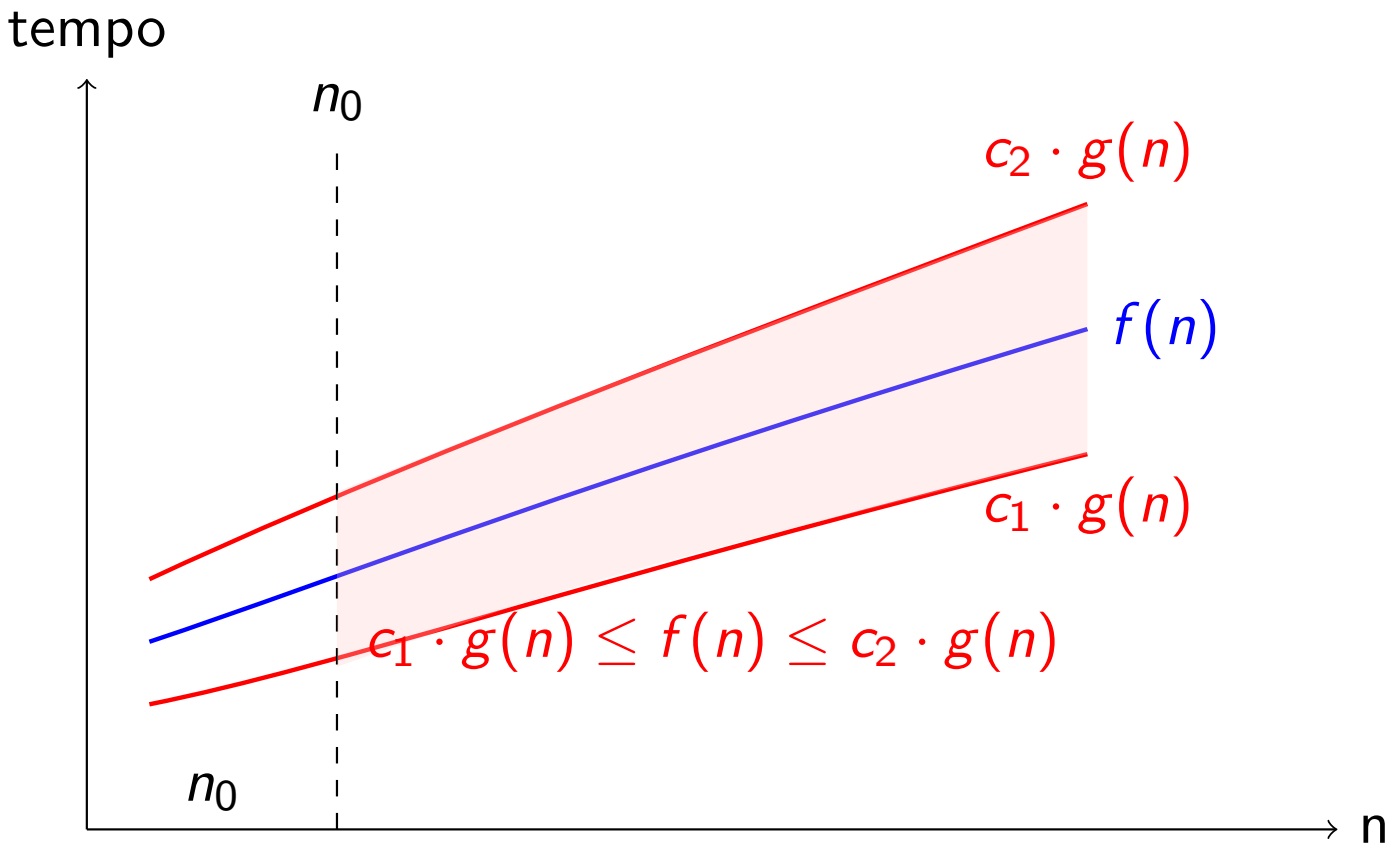
Big Theta - Interpretação

Significado: $g(n)$ é um **limite apertado** para $f(n)$

Características:

- Fornece o crescimento **exato** da função
- Mais informativo que O ou Ω isoladamente
- Indica que $f(n)$ e $g(n)$ crescem na mesma taxa
- Ignora constantes multiplicativas e termos de menor ordem

Big Theta - Interpretação Visual



$f(n)$ **fica "espremida"entre os dois limites**

Exemplo 3 - Demonstrando Big Theta

Provar que $3n + 5 = \Theta(n)$

Método: Usar os resultados anteriores

Já provamos:

- $3n + 5 = O(n)$ com $c_2 = 4$, $n_0 = 5$
- $3n + 5 = \Omega(n)$ com $c_1 = 3$, $n_0 = 1$

Combinando: Para $n \geq 5$:

$$3 \cdot n \leq 3n + 5 \leq 4 \cdot n$$

Conclusão: $3n + 5 = \Theta(n)$ com $c_1 = 3$, $c_2 = 4$, $n_0 = 5$

Regras de Simplificação - Soma

Soma de funções:

$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

Exemplos práticos:

- $O(n) + O(n^2) = O(n^2)$
- $O(\log n) + O(n) = O(n)$
- $O(n^2) + O(n^3) = O(n^3)$

Interpretação: O termo que cresce mais rápido domina a soma.

Vale também para: Ω e Θ

Regras de Simplificação - Produto

Produto de funções:

$$O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$$

Exemplos práticos:

- $O(n) \cdot O(\log n) = O(n \log n)$
- $O(n) \cdot O(n) = O(n^2)$
- $O(\sqrt{n}) \cdot O(n) = O(n\sqrt{n}) = O(n^{3/2})$

Interpretação: Os crescimentos se multiplicam.

Vale também para: Ω e Θ

Regras de Simplificação - Constantes

Constantes multiplicativas:

$$O(c \cdot f(n)) = O(f(n)) \text{ para qualquer constante } c > 0$$

Exemplos:

- $O(5n) = O(n)$
- $O(100n^2) = O(n^2)$
- $O(0.001n \log n) = O(n \log n)$

Justificativa: As constantes não afetam a taxa de crescimento assintótico.

Importante: Isso não vale para exponenciais! $O(2^n) \neq O(3^n)$

Regras de Simplificação - Termos Dominantes

Eliminação de termos de menor ordem:

$$n^3 + n^2 + n + 1 = O(n^3)$$

Regra geral: Em um polinômio, apenas o termo de maior grau importa.

Mais exemplos:

- $5n^4 + 3n^3 + 2n^2 + 7n + 10 = \Theta(n^4)$
- $n^2 + 1000n = \Theta(n^2)$
- $\log n + \sqrt{n} + n = \Theta(n)$

Dica prática: Identifique o termo que cresce mais rápido!

Hierarquia de Crescimento

Ordem crescente de complexidade:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n) < O(n!)$$

Interpretação:

- Cada função cresce mais rápido que a anterior
- Para n suficientemente grande, a diferença se torna dramática
- Algoritmos à esquerda são preferíveis

Comparação Numérica - Parte 1

Crescimento para diferentes valores de n:

n	$\log n$	n	$n \log n$	n^2
10	3.3	10	33	100
100	6.6	100	664	10,000
1,000	10	1,000	9,966	1,000,000
10,000	13.3	10,000	132,877	100,000,000

Observação: Note como a diferença cresce rapidamente!

Comparação Numérica - Parte 2

Crescimento exponencial:

n	n^3	2^n	$n!$
10	1,000	1,024	3,628,800
20	8,000	1,048,576	2.4×10^{18}
30	27,000	1.1×10^9	2.7×10^{32}

Conclusão: Algoritmos exponenciais e fatoriais se tornam impraticáveis rapidamente!

Exercício 1 - Classificação

Classifique as seguintes funções usando Big O:

a) $f(n) = 5n + 3$
b) $f(n) = n^2 + 100n$
c) $f(n) = 3n^3 + 2n^2 + n + 5$
d) $f(n) = \log(n) + n$
e) $f(n) = 2^n + n^{100}$

Dica: Identifique o termo que cresce mais rapidamente.

Exercício 2 - Demonstração

Prove formalmente que $2n^2 + 3n = O(n^2)$

Passos necessários:

1. Encontrar uma constante c apropriada
2. Encontrar um valor n_0 apropriado
3. Mostrar que $2n^2 + 3n \leq c \cdot n^2$ para $n \geq n_0$
4. Verificar a demonstração com valores específicos

Exercício 3 - Análise de Casos

Para o algoritmo de busca binária, determine:

- **Melhor caso**: Complexidade e quando ocorre
- **Pior caso**: Complexidade e quando ocorre
- **Caso médio**: Complexidade esperada

Dica: Lembre-se que a busca binária divide o espaço de busca pela metade a cada iteração.

Respostas - Exercício 1

Classificações corretas:

a) $5n + 3 = O(n) \rightarrow$ Linear

b) $n^2 + 100n = O(n^2) \rightarrow$ Quadrática

c) $3n^3 + 2n^2 + n + 5 = O(n^3) \rightarrow$ Cúbica

d) $\log(n) + n = O(n) \rightarrow$ Linear

e) $2^n + n^{100} = O(2^n) \rightarrow$ Exponencial

Princípio: O termo de maior ordem domina a complexidade.

Respostas - Exercício 2

Demonstração de $2n^2 + 3n = O(n^2)$:

Passo 1: Para $n \geq 1$, temos $3n \leq 3n^2$
Passo 2: Portanto:

$$2n^2 + 3n \leq 2n^2 + 3n^2 = 5n^2$$

Passo 3: Escolhemos $c = 5$ e $n_0 = 1$
Passo 4: Verificação para $n = 10$:

- $2(10)^2 + 3(10) = 200 + 30 = 230$
- $5(10)^2 = 500$
- $230 \leq 500$

Conclusão: $2n^2 + 3n = O(n^2)$ com $c = 5$ e $n_0 = 1$

Respostas - Exercício 3

Busca Binária - Análise de casos:

Melhor caso: $\Theta(1)$

- Ocorre quando o elemento está no meio do array
- Encontramos na primeira tentativa

Pior caso: $\Theta(\log n)$

- Elemento não existe ou está em uma das extremidades
- Precisamos dividir até sobrar 1 elemento: $\log_2 n$ divisões

Caso médio: $\Theta(\log n)$

- Em média, fazemos cerca de $\log_2 n - 1$ comparações

Conceitos-Chave para Fixar

Pontos fundamentais:

- O: "no máximo", Ω: "no mínimo", Θ: "exatamente"
- Constantes multiplicativas não importam assintoticamente
- Termo dominante determina a complexidade
- Demonstrações requerem encontrar c e n_0 apropriados
- Diferentes casos podem ter complexidades diferentes

Algoritmos de Ordenação

Focaremos em dois algoritmos fundamentais:

- **Merge Sort**: Estratégia "dividir para conquistar"
- **Quick Sort**: Estratégia baseada em pivotamento

O que estudaremos:

- Implementação e funcionamento dos algoritmos
- Análise detalhada de complexidade (melhor, pior, médio)
- Complexidade espacial (uso de memória)
- Comparação entre as duas abordagens

Próxima Aula - Parte 2

Análise de complexidade que faremos:

Merge Sort:

- Todos os casos: $\Theta(n \log n)$
- Complexidade espacial: $\Theta(n)$
- Estável e previsível

Quick Sort:

- Melhor/Médio caso: $\Theta(n \log n)$
- Pior caso: $\Theta(n^2)$
- Complexidade espacial: $\Theta(\log n)$ em média

Exemplos práticos:

- Quando usar cada algoritmo
- Casos reais de aplicação
- Otimizações possíveis