Análise de Algoritmos

Rafael Alves da Costa

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
FATEC Carapicuíba

Aula 2 - Estrutura de Dados

08/2025

Sumário

Notações Assintóticas

Introdução

- **Considerações 1:**
  - Ordem de crescimento do tempo de execução.
  - Comparação da eficiência dos algoritmos.
- **Considerações 2:**
  - Busca-se calcular a função de complexidade $f(n)$.
  - Para valores pequenos de n (entrada), praticamente qualquer algoritmo custa pouco para executar. Logo, a escolha do algoritmo terá pouca influência.

A análise assintótica de algoritmos dedica-se ao estudo do comportamento do algoritmo para valores grandes de **n** (entrada).

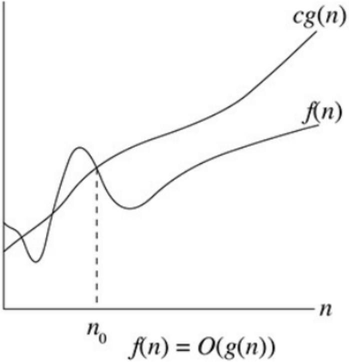
Notações Assintóticas

Vamos estudar mais detalhadamente!

- Notação O-grande (big-O): limite superior assintótico.
- Notação Ω-grande: limite inferior assintótico.
- Notação Θ-grande: limite assintótico tight.

Definição Formal: Notação O-grande (big-O)

- **g(n)** domina assintoticamente **f(n)** se:
  - $f(n) = O(g(n))$: Existe duas constantes c e n_0 tais que $|f(n)| \leq c \cdot |g(n)|$ para todo $n \geq n_0$.



Fonte: Algoritmos[1]

Dominação Assintótica: Exemplo

Funções:

- $f(n) = (n+1)^2$
- $g(n) = n^2$

Dominação Assintótica de g(n) sobre f(n) :

- $|f(n)| \leq c|g(n)| \quad \text{para } n \geq n_0$
- $c = 4$ e $n_0 = 1$
- $|(n+1)^2| \leq 4|n^2|, \quad \text{para } n \geq 1$

Dominação Assintótica de f(n) sobre g(n) :

- $|g(n)| \leq c|f(n)| \quad \text{para } n \geq n_0$
- $c = 1$ e $n_0 = 1$
- $|n^2| \leq |(n+1)^2|, \quad \text{para } n \geq 0$

Fatec Carapicuíba

Notação O Grande (big-O)

Especifica um limite superior para $f(n)$

Dominação Assintótica:

- Escrevemos $f(n) = O(g(n))$ para expressar que $g(n)$ domina assintoticamente $f(n)$.
- Lê-se: $f(n)$ é da ordem de no máximo $g(n)$.

Exemplo:

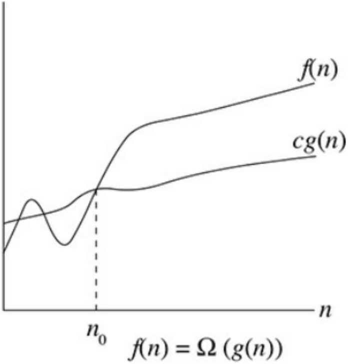
- Quando dizemos que o tempo de execução $T(n)$ de um programa é $O(n^2)$, significa que existem constantes c e n_0 tais que, para valores de $n \geq n_0$, $T(n) \leq cn^2$.

Notação O Grande (big-O): Operações

- $f(n) = O(f(n))$
- $c \cdot O(f(n)) = O(f(n))$, onde c é uma constante
- $O(f(n)) + O(f(n)) = O(f(n))$
- $O(O(f(n))) = O(f(n))$
- $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$
- $O(f(n)) \cdot O(g(n)) = O(f(n) \cdot g(n))$
- $f(n) \cdot O(g(n)) = O(f(n) \cdot g(n))$

Definição Formal: Notação Ω

- **g(n)** é dominada assintoticamente pela função **f(n)** se:
  - $f(n) = \Omega(g(n))$: Existe duas constantes c e n_0 tais que $|f(n)| \geq c \cdot |g(n)|$ para todo $n \geq n_0$.



Fonte: Algoritmos[1]

Notação Ω

Especifica um limite inferior para $f(n)$

Definição: Uma função $f(n)$ é $\Omega(g(n))$ se:

- Existem duas constantes positivas c e n_0 tais que, para $n \geq n_0$, temos $|f(n)| \geq c|g(n)|$.

Exemplo 1:

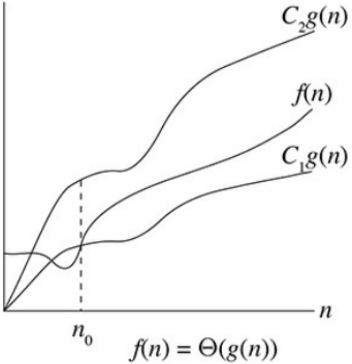
- Quando dizemos que o tempo de execução $T(n)$ de um programa é $\Omega(n^2)$, significa que existem constantes c e n_0 tais que, para valores de $n \geq n_0$, $T(n) \geq cn^2$.

Exemplo 2:

- Para mostrar que $f(n) = 3n^3 + 2n^2$ é $\Omega(n^3)$ basta fazer c $= 1$, e então $f(n) = 3n^3 + 2n^2 \geq n^3$ para $n \geq 0$.

Definição Formal: Notação Θ

- $f(n) = \Theta(g(n))$: $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.



Fonte: Algoritmos[1]

- Para uma função ser $\Theta(g(n))$, ela deve ser $O(g(n))$ e $\Omega(g(n))$.

Notação Θ

Especifica um limite assintótico firme para $f(n)$

Definição: Uma função $f(n)$ é $\Theta(g(n))$ se:

- Existem três constantes positivas c_1, c_2 e n_0, tais que, para $n \geq n_0$, temos: $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$.

Isto é:

- Para todo $n \geq n_0$, a função $f(n)$ é igual a $g(n)$ a menos de uma constante.

Uso Correto de Notações

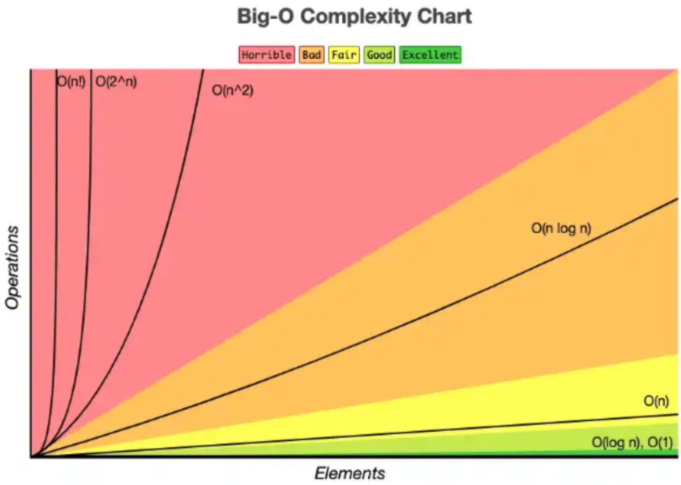
- Use notação assintótica para indicar limites precisos.
- Evite confundir O-notação com Θ-notação.
- Exemplos de uso correto e incorreto de notações.

Funções Comuns na Análise de Algoritmos

- Polinômios: n, n^2, n^3, etc.
- Logaritmos: $\log n$, $n \log n$.
- Exponenciais: 2^n.
- **Relação de dominância de funções tipicamente utilizadas:**

$$\boxed{n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg \log n \gg 1}$$

Visão por Gráfico: Considerando Notação Big-O



Fonte: Análise de Algoritmos uma Introdução

Fatec Carapicuíba

Comparando Taxas de Crescimento

- Comparação entre polinômios e logaritmos.
- Exemplo: n^2 vs. $n \log n$.
- Importância da ordem de crescimento para grandes valores de n.

Conclusão

- Importância da análise assintótica na teoria dos algoritmos.
- Ferramentas para caracterizar tempos de execução.
- Relevância para a comparação e escolha de algoritmos.

Análise de Algoritmos: Análise assintótica

Eficiência Assintótica

- Estudo da eficiência de algoritmos com base na ordem de crescimento.
- Foco no comportamento do algoritmo à medida que o tamanho do input aumenta.
- Importância de simplificar a análise assintótica.
- Vamos exemplificar estudando o problema de ordenação!!!

Problema de ordenação e pseudocódigo

Problema de Ordenação:

- **Entrada:** Uma sequência de n números $\langle a_1, a_2, \ldots, a_n \rangle$.
- **Saída:** Uma permutação (reordenação) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Chaves:

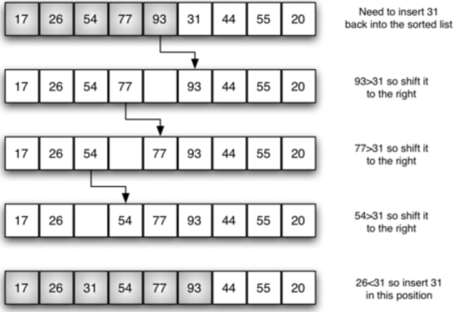
- Os números que desejamos ordenar são conhecidos como chaves.
- Embora conceitualmente estejamos ordenando uma sequência, a entrada é dada na forma de um arranjo com n elementos.

Pseudocódigo:

- Algoritmos são descritos como programas escritos em pseudocódigo.
- Semelhante a linguagens como C, C++, Java, Python ou Pascal.
- Emprega métodos expressivos claros e concisos, inclusive em linguagem comum.
- Não se preocupa com questões de engenharia de software como abstração de dados, modularidade e tratamento de erros.

Exemplo: Ordenação Por Inserção

A ordenação por inserção (Diminuir e Conquistar) mantém uma **sublista ordenada** nas posições inferiores da lista. Cada novo item é "inserido" na sublista anterior de modo que a sublista ordenada fique com um item a mais.[1]



Quinta passagem da lista em detalhe

- **Vamos ver uma animação!!!**

Fatec Carapicuíba

Exemplo: Ordenação Por Inserção (pseudocódigo)

```
INSERTION-SORT(A)
  for j = 2 to A.comprimento
      chave = A[j]
      // Inserir A[j] na sequência ordenada A[1.. j − 1].
      i = j − 1
      while i > 0 e A[i] > chave
          A[i + 1] = A[i]
          i = i − 1
      A[i + 1] = chave
```

Fonte: Algoritmos[1]

Motivação para o estudo da Ordenação Por Inserção

- **Análise de Algoritmos**: Fundamental para prever desempenho.
- **Inserção Direta**: Exemplo clássico para estudo.
- **Medir Tempo de Execução**: Implementação, máquina, compilador, tarefas concorrentes influenciam.
- **Necessidade de Análise Teórica**: Predizer tempo de execução para diferentes entradas e contextos.

Análise do Algoritmo

- **Custo de Cada Linha**: Tempo constante c_k por linha.
- **Número de Execuções**: Contar quantas vezes cada linha é executada.
- **Fórmula Precisa**: Combinação dos custos e execuções resulta em fórmula complexa.
- **Notação Simples**: Usar notação assintótica (Big O (ou O - grande)) para simplificação.

Tempo de Execução para Ordenação Por Inserção

	INSERTION-SORT(A, n)	*cost*	*times*
1	**for** $i = 2$ **to** n	c_1	n
2	$key = A[i]$	c_2	$n-1$
3	// Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
4	$j = i - 1$	c_4	$n-1$
5	**while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6	$A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7	$j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8	$A[j+1] = key$	c_8	$n-1$

Exemplo de análise

- **Melhor Caso**: Entrada já ordenada.
  - Teste de loop enquanto (linha 5) executa uma vez por iteração.
  - $T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$
  - **Resultado**: $O(n)$
- **Pior Caso**: Entrada em ordem inversa.
  - Teste de loop enquanto executa i vezes por iteração.
  - $T(n) =$ $(c_5 + c_6 + c_7)n^2/2 + (c_1 + c_2 + c_4 + c_5 - c_6 - c_7 + c_8)n/2 - (c_2 + c_4 + c_5 + c_8)$
  - **Resultado**: $O(n^2)$

Fatec Carapicuíba

Tempo de Execução para Ordenação Por Inserção

INSERTION-SORT(A, n)	*cost*	*times*
1 **for** $i = 2$ **to** n	c_1	n
2 $key = A[i]$	c_2	$n-1$
3 // Insert $A[i]$ into the sorted subarray $A[1:i-1]$.	0	$n-1$
4 $j = i - 1$	c_4	$n-1$
5 **while** $j > 0$ and $A[j] > key$	c_5	$\sum_{i=2}^{n} t_i$
6 $A[j+1] = A[j]$	c_6	$\sum_{i=2}^{n} (t_i - 1)$
7 $j = j - 1$	c_7	$\sum_{i=2}^{n} (t_i - 1)$
8 $A[j+1] = key$	c_8	$n-1$

Exemplo de análise

- **Melhor Caso**: Entrada já ordenada.
  - Teste de loop enquanto (linha 5) executa uma vez por iteração.
  - $T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$
  - **Resultado**: $O(n)$
- **Pior Caso**: Entrada em ordem inversa.
  - Teste de loop enquanto executa i vezes por iteração.
  - $T(n) =$ $(c_5 + c_6 + c_7)n^2/2 + (c_1 + c_2 + c_4 + c_5 - c_6 - c_7 + c_8)n/2 - (c_2 + c_4 + c_5 + c_8)$
  - **Resultado**: $O(n^2)$
  - Vamos analisar dois exemplos no notebook!!!

Fatec Carapicuíba

Ordem de Crescimento

- **Importância da Ordem de Crescimento**:
  - Ignorar termos de ordem inferior e coeficientes constantes.
  - Foco no termo dominante para grandes entradas.
- **Os exemplos estudados no notebook possuem**:
  - **Exercicio1**: $O(n)$ - cresce linearmente com n
  - **Exercicio2**: $O(n^2)$ - cresce quadraticamente com n

Complexidade de pior e melhor caso

- **Utilizando as Notações:**
- Insertion Sort: $O(n^2)$ no pior caso e $\Theta(n^2)$ no pior caso.
- Insertion Sort: $O(n)$, $\Omega(n)$ no melhor caso e $\Theta(n)$ no melhor caso.

Complexidade de pior e melhor caso

- **Utilizando as Notações:**
- Insertion Sort: $O(n^2)$ no pior caso e $\Theta(n^2)$ no pior caso.
- Insertion Sort: $O(n)$, $\Omega(n)$ no melhor caso e $\Theta(n)$ no melhor caso.
- **Vamos ver no notebook!!!**

Considerações

Considerações

- **Recordando!!!**
  - Notação assintótica.
  - Análise assintótica.
  - Pior caso.
  - Exercícios.

Fatec Carapicuíba

Considerações

- **Recordando!!!**
  - Notação assintótica.
  - Análise assintótica.
  - Pior caso.
  - Exercícios.
- **Referências utilizadas!!!**

  - [1]Cormen, T. H. et al. Algoritmos: Teoria e prática, 3a edição. Elsevier, 2012.
  - Cormen, T. H. et al. Introduction to algorithms. Cambridge: MIT press, 2009.
  - Apresentação do Prof. Reinaldo Fortes