# Medidas de Eficiência de Algoritmos

Prof. Me. Lucas Bruzzone

Aula 01

# Objetivos da Disciplina

- Analisar complexidade temporal e espacial de algoritmos
- Aplicar técnicas de projeto de algoritmos eficientes
- Desenvolver algoritmos usando divisão e conquista e programação dinâmica
- Implementar algoritmos clássicos de busca
- Entender conceitos básicos de análise assintótica

# A Importância da Eficiência

**Cenários reais:**

- Busca no Google: bilhões de páginas em milissegundos
- GPS: calcular rota em tempo real
- Jogos: renderizar 60 FPS
- Big Data: processar terabytes de dados

**Algoritmo eficiente = diferença entre viável e inviável**

# Exemplo: Busca em Lista

**Problema:** Encontrar elemento em lista de 1 milhão de itens

**Busca Linear:** Até 1.000.000 comparações
**Busca Binária:** Até 20 comparações

Diferença de **50.000x** na velocidade!

# Como Funciona a Busca Binária

**Buscar 11 em:** [1, 3, 5, 7, 9, 11, 13]

**Passo 1:** Compare com o meio (posição 3, valor 7)

- $11 > 7 \rightarrow$ vá para metade direita: [9, 11, 13]

**Passo 2:** Compare com o meio da direita (posição 1, valor 11)

- $11 = 11 \rightarrow$ encontrou!

**Total:** 2 comparações vs 6 da busca linear

**Cada passo elimina metade:** $7 \rightarrow 3 \rightarrow 1$ elemento

# Como Medir Eficiência?

**Critérios principais:**

- **Tempo de execução**: Quanto demora para executar
- **Espaço de memória**: Quanta memória consome
- **Escalabilidade**: Como se comporta com entrada maior

**Abordagens:**

- **Empírica**: Medir tempo real de execução
- **Teórica**: Analisar algoritmo matematicamente

# Problemas da Análise Empírica

- **Dependente de hardware**: Resultados variam
- **Dependente de linguagem**: Implementação influencia
- **Dependente de entrada**: Dados específicos
- **Trabalhosa**: Precisa implementar para testar
- **Limitada**: Só testa casos específicos

**Necessário: análise independente de implementação**

# Modelo de Análise

**Modelo RAM (Random Access Machine):**

- Operações básicas custam 1 unidade de tempo
- Acesso à memória em tempo constante
- Independente de hardware específico

**Operações básicas:**

- Atribuição, comparação, operações aritméticas
- Acesso a array, chamada de função

**Foco:** Crescimento do tempo com tamanho da entrada

# Função de Complexidade

$T(n)$**:** Tempo de execução em função do tamanho da entrada

**Exemplo - Busca Linear:**

- Melhor caso: $T(n) = 1$ (primeiro elemento)
- Pior caso: $T(n) = n$ (último elemento)
- Caso médio: $T(n) = n/2$

**Geralmente analisamos o pior caso**

# Melhor, Pior e Caso Médio

**Melhor caso:**

- Entrada mais favorável ao algoritmo
- Raramente útil na prática

**Pior caso:**

- Entrada que maximiza o tempo
- Garantia de limite superior
- Mais comum na análise

**Caso médio:**

- Comportamento esperado
- Requer conhecimento da distribuição
- Mais difícil de calcular

# Exemplo: Busca em Array

**Problema:** Encontrar um elemento específico

**Melhor caso:** Elemento está na primeira posição

- Apenas 1 comparação
- $T(n) = 1$

**Pior caso:** Elemento está na última posição ou não existe

- Precisa verificar todos os elementos
- $T(n) = n$

**Caso médio:** Elemento em posição aleatória

- Em média, metade do array
- $T(n) = n/2 \approx n$

# Análise de Espaço

$S(n)$**:** Espaço adicional usado pelo algoritmo

**Tipos de espaço:**

- **Entrada**: Espaço dos dados originais
- **Auxiliar**: Espaço adicional usado
- **Total**: Entrada + auxiliar

**Exemplos:**

- Busca linear: $S(n) = 1$ (apenas variáveis de controle)
- Cópia de array: $S(n) = n$ (array auxiliar)
- Recursão: $S(n) =$ profundidade da pilha

# Como Contar Operações

**Exemplo - Soma de Array:**

```
soma = 0                    # 1 operacao
for i in range(n):          # n+1 comparacoes
    soma += arr[i]          # 2n operacoes
return soma                 # 1 operacao
```

**Total:** $T(n) = 1 + (n+1) + 2n + 1 = 3n + 3$

**Dominante:** $3n$ (termo de maior grau)

# Exemplo - Loops Aninhados

**Multiplicação de Matrizes (simplificado):**

```
for i in range(n):
    for j in range(n):
        resultado[i][j] = A[i][j] * B[i][j]
```

**Operações:** Para cada $i$ de 0 a $n-1$, fazemos $n$ multiplicações

**Total:** $T(n) = n \times n = n^2$

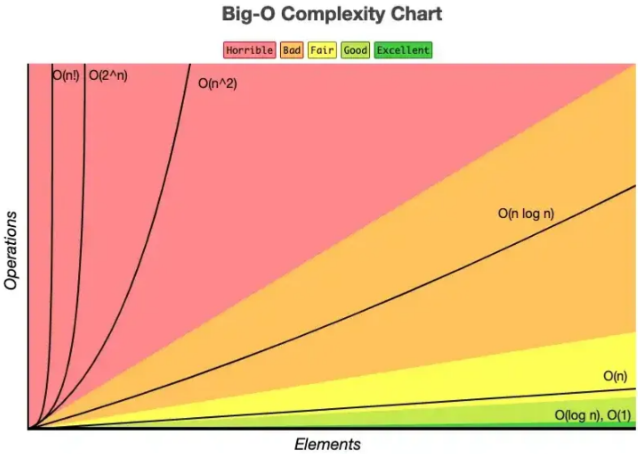
# Dicas para Análise

- **Loops simples**: $O(n)$
- **Loops aninhados**: $O(n^2)$, $O(n^3)$, etc.
- **Divisão pela metade**: $O(\log n)$
- **Recursão**: Resolver recorrência
- **Foco no crescimento**: Ignorar constantes e termos menores

**Hierarquia comum:**

$$1 < \log n < n < n \log n < n^2 < n^3 < 2^n < n!$$

# Visualização das Complexidades



# Próxima Aula

## Notação Big-O e Análise Assintótica

Estudaremos como formalizar a análise de complexidade