Resumo sobre complexidade de algoritmos

Métricas:

- Tempo, consumo de memória, etc ...;
- Ordem de crescimento da função em relação ao volume de dados de entrada é o que interessa;

Formas de análise:

- melhor caso;
- caso médio;
- pior caso (mais utilizado);

Busca sequencial em um vetor de tamanho n:

- melhor caso: 1 comparação
- caso médio (n + 1)/2 comparações
- pior caso: n comparações

Por que pior caso?

- pior caso ocorre na maior parte do tempo (ex. Busca em vetor que não encontrou);
- o melhor caso é igual para a maioria dos algoritmos;
- determinar o caso médio é difícil ou impossível em muitas situações;
- o pior caso representa o limite superior da função que representa a complexidade de algoritmo: nunca será pior que o pior caso;

Notação O (big O)

- usada para representar o pior caso de um algoritmo;
- Regras básicas:
    - Desprezar as constantes da função;
    - Desprezar multiplicadores;
    - Considerar a maior ordem da função;
- Exemplo:
    - Uma operação tem custo medido em tempo dado pela função:
      $T(n) = 3n^2 + 10n + 10 = O(n^2)$
- Outras propriedades:
    - Constante: $O(c) = O(1)$
    - Multiplicação de constantes: $O(cT) = cO(T) = O(T)$
    - Adição: $O(T_1) + O(T_2) = O(T_1 + T_2) = \max(O(T_1), O(T_2))$
    - Multiplicação: $O(n)*O(n) = O(n^2)$

Por que considerar apenas a maior ordem da função?

- O que interessa na complexidade de um algoritmo é o crescimento da função de complexidade em relação ao crescimento de n. Para valores altos de n, os multiplicadores e constantes da função se tornam pouco significativos:
- Para a função: $T(n) = 3n^2 + 10n + 10$ se calcularmos o percentual de participação de cada parte no resultado temos:
    - Para n = 10:
        - $3n^2$ = 73,2% do resultado;
        - 10n = 24,4% do resultado;
        - 10 = 2,4% do resultado

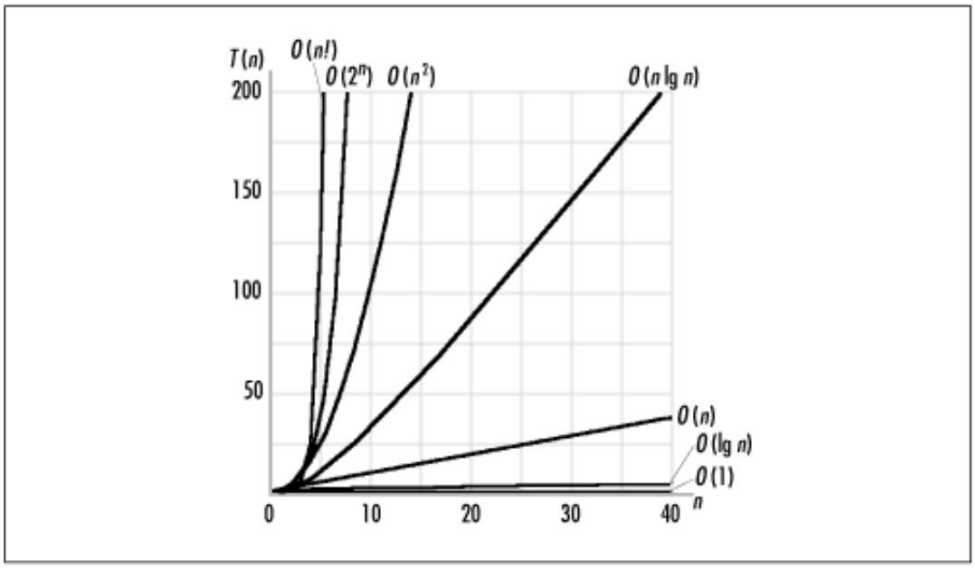
- Para n = 100:
  - $3n^2$ = 96,7% do resultado;
  - 10n = 3,2% do resultado;
  - 10 = 0,1% do resultado

Funções comuns de complexidade de algoritmos:

Table 4.1. Some Situations Wherein Common Complexities Occur

Complexity	Example
$O(1)$	Fetching the first element from a set of data
$O(\lg n)$	Splitting a set of data in half, then splitting the halves in half, etc.
$O(n)$	Traversing a set of data
$O(n \lg n)$	Splitting a set of data in half repeatedly and traversing each half
$O(n^2)$	Traversing a set of data once for each member of another set of equal size
$O(2^n)$	Generating all possible subsets of a set of data
$O(n!)$	Generating all possible permutations of a set of data

Representação gráfica das funções de crescimento:



Outras representações de complexidade:

Θ - caso médio;

Ω - melhor caso;

Imagens retiradas de:
Loudon, Kyle. Mastering Algorithms with C. 1. Ed. O'Reilly, 1999.