Análise de Algoritmos

João Paulo Dias de Almeida
jp.dias.almeida@gmail.com

Universidade Federal de Sergipe

O que vamos aprender hoje?

- Relembrar o que é algoritmo
- Entender o que é e para que serve a análise de algoritmos
- Compreender o modelo RAM
- Entender como conduzir uma análise de algoritmo
- Aprender a utilizar a notação O

Glossário

Algoritmo:

Procedimento computacional que recebe um conjunto de dados de entrada e produz uma saída

Custo:

Tempo necessário para executar um algoritmo

Função de custo:

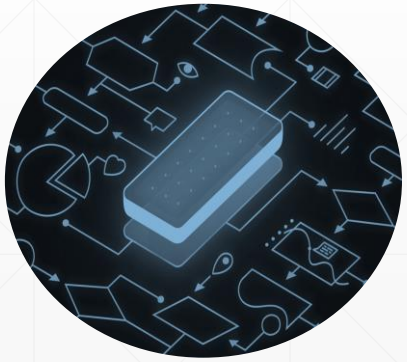
Descreve a relação entre tempo e entrada de dados

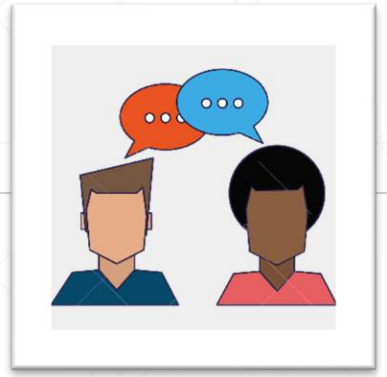
Modelo RAM:

Modelo de computador hipotético no qual instruções simples são executadas com a mesma duração

O que é um algoritmo?

É qualquer procedimento computacional bem definido que toma um conjunto de valores como entrada e produz um conjunto de valores como saída **- Ascencio**





Como vocês fariam a análise de um algoritmo?

Análise de algoritmos

- A análise de um algoritmo pode ser feita por diferentes ângulos:

1. Análise de um algoritmo em particular

   - Qual o custo (tempo de execução) do algoritmo?
   - Quanto de memória o algoritmo necessita?
     - Largura de banda
     - Ou qualquer outro hardware em geral...

Análise de algoritmos

- A análise de um algoritmo pode ser feita por diferentes ângulos:

  2. Análise de uma classe de algoritmos

     - Qual seria o algoritmo de menor custo para resolver um problema?
     - É comum realizar a análise de uma família de algoritmos para identificar o que se adequa melhor à situação

Para analisar um algoritmo poderíamos medir o tempo de execução em um computador e anotar esse tempo?

O tempo de execução do algoritmo é confiável?

PC

Supercomputador

Outras características a se considerar

- Ao medir o tempo de execução de um algoritmo para mensurar o seu custo, você pode ser afetado também por:
  - Compilador
  - Sistema Operacional
  - Especificidades de cada hardware
- Para evitar estas imprecisões, usa-se um modelo hipotético de computador chamado RAM – Random Access Machine

O modelo RAM

- Esse modelo de computador possui todas as instruções de um computador real

- O custo (tempo de execução) individual de instruções simples é irrelevante no nosso caso
  - Instruções simples → operações aritméticas
    - Ou qualquer instrução que exija apenas um passo para ser executada
  - Por isso, vamos considerar que todas as instruções simples demoram um tempo constante

O modelo RAM

- **Loops** e sub-rotinas não são considerados irrelevantes
  - Ordenar 1 000 000 de itens é consideravelmente mais lento do que ordenar 10

https://visualgo.net/pt/sorting

O modelo RAM

O modelo RAM foi idealizado para que a análise do algoritmo seja independente da máquina

O modelo RAM

- Usando o modelo RAM, podemos contar quantos passos cada instrução leva para ser executada
  - Essa quantidade de passos será dependente da entrada
    - Exemplo do loop
- A nossa análise deve comparar dois algoritmos considerando todos os possíveis tamanho de entradas

Função de custo

- O tempo de execução de um algoritmo é representado por uma função de custo T
  - T(n) é o tempo necessário para executar um algoritmo para o problema de tamanho n

- **Exemplo:** Agora, vamos considerar um algoritmo para encontrar o menor elemento de um vetor A de inteiros, com n elementos

Exemplo

T(n) = n - 1

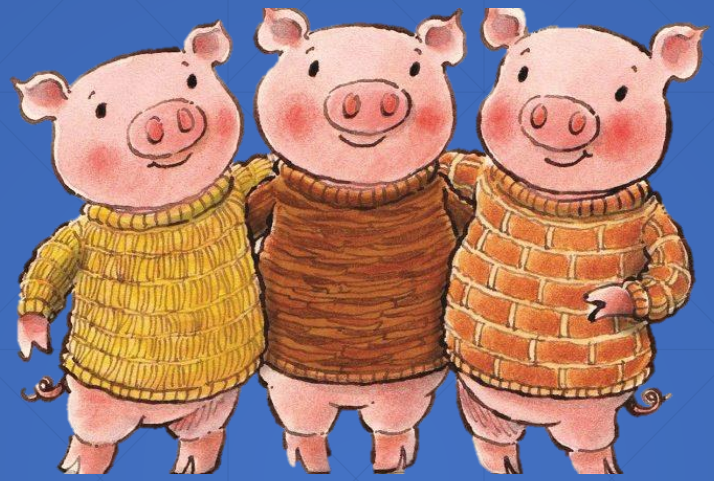
n-1 comparações

```
int calculamenor (int A[], int n)
{
        int i, menor;

        menor = A[0];
        for (i = 1; i < n; i++)
        {
                if (A[i] < menor)
                        menor = A[i];
        }
        return menor;
}
```

Função de custo

- A função T(n) representa a relação entre o tamanho do problema e o tempo necessário para resolvê-lo
  - O tempo de execução depende principalmente do tamanho da entrada
  - Experimente “plotar” o gráfico
- Considere agora o caso de realizar uma busca em um arquivo de dados sequencial
  - Nesse caso, não teremos apenas uma função de custo
  - Serão três análises: o melhor, o médio e pior custo



O melhor, o médio, e o pior caso

Análise de complexidade

- Imagine executar um algoritmo e todas as combinações de dados executadas por ele
- Para o problema da ordenação, o algoritmo compara o valor de cada item com todos os outros
  - O **pior caso** é caso é aquele que representa o número **máximo** de passos que o algoritmo utiliza para realizar uma tarefa de tamanho N
  - O **melhor caso** é aquele que representa o número **mínimo** de passos
  - O **caso médio** é aquele representa a **média** de passos para executar a tarefa

Análise de complexidade

O pior caso é o mais útil das três medidas

Análise de complexidade

- Imagine levar **n** reais para um cassino e fazer algumas apostas
  - **No melhor caso**, você ganha todas as apostas e sai de lá dono do cassino

Análise de complexidade

- Imagine levar **n** reais para um cassino e fazer algumas apostas
  - **No pior caso**, você perde todos n reais.
    - Isso é fácil de calcular e muito provável de acontecer!

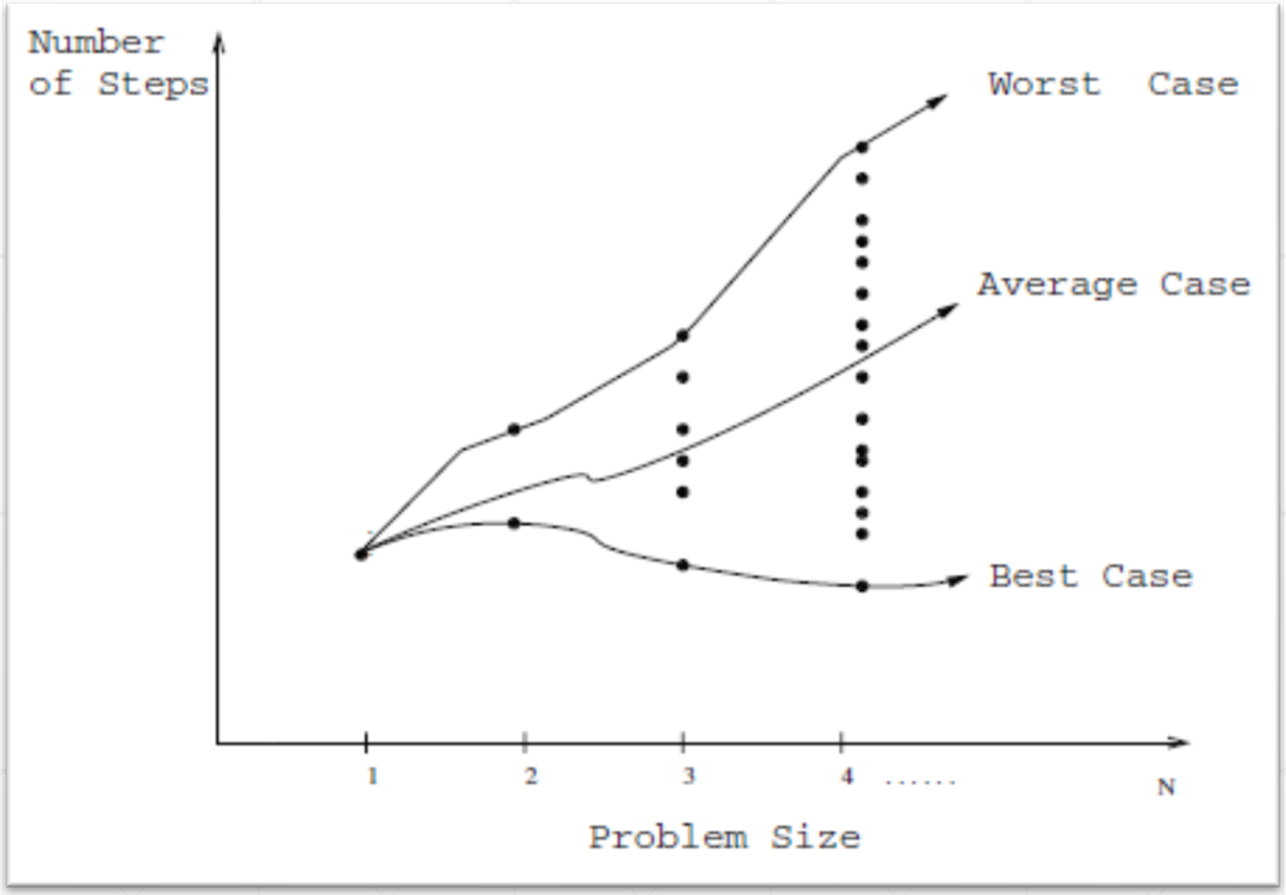
Análise de complexidade

- Imagine levar **n** reais para um cassino e fazer algumas apostas
  - **O caso médio** é que os apostadores perdem 87,32% do dinheiro
  - Isso é difícil de estabelecer e está sujeito a debate:
    - Pessoas inteligentes ganham mais dinheiro?
    - A média foi calculada usando uma pessoa inteligente?
    - Trapaceiros ganham mais que jogadores honestos. E agora?

Análise de complexidade

- O importante é perceber que esta análise define uma função numérica representando uma relação entre **tempo** e **tamanho da entrada**

Análise de complexidade



Função de custo

- O importante é perceber que esta análise define uma função numérica representando uma relação entre **tempo** e **tamanho da entrada**
- As funções podem ser bem definidas como:

$$y = x^2 - 2x + 1$$

- Ou uma função temporal do patrimônio de um investidor na bolsa:

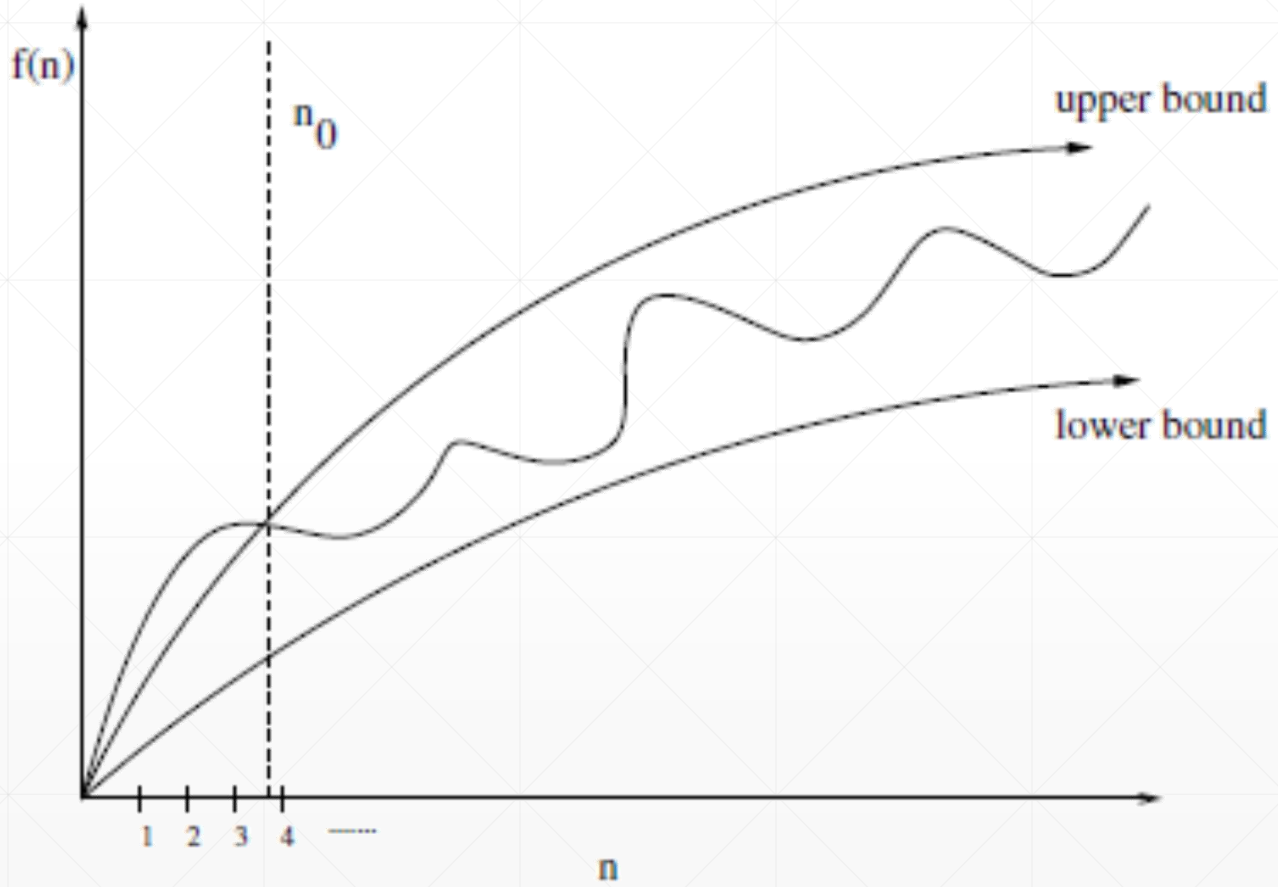
$$W_h(i) = W_i(i) + N_i(i)D_i + (W_i(i) - N_i(i)P_i)r + N_i(i)(P_h - P_i)$$

Função de custo - desvantagens

- Além disso, é difícil trabalhar com essas funções porque:

1. **Podem existir muitas oscilações:** a busca binária é mais rápida quando o problema tem um tamanho específico

Função de custo com muitas oscilações



Função de custo - desvantagens

- Além disso, é difícil trabalhar com essas funções porque:

2. **Exige muito detalhe:**
   - É necessário contar todas as instruções do algoritmo para o pior caso
   - É dependente de decisões de implementação
     - O programa usa if encadeado ou um switch-case?

Função de custo - desvantagens

- Uma função precisa, como essa abaixo, acrescenta pouca informação útil:

$$T(n) = 12754\, n^2 + 4353\, n + 834\, \lg_2 n + 13546$$

- O importante é perceber que **n** cresce de forma quadrática
- Para focar no que realmente importa, vamos utilizar notações assintóticas

Notações Assintóticas

Notação Assintótica

- O melhor, o médio, e pior caso são funções sob o tamanho do conjunto de dados a ser processado
- **A notação assintótica** serve para comparar estas funções ignorando os detalhes que não são úteis para a nossa análise de comparação entre algoritmos
  - Vamos ignorar as diferenças entre multiplicações de constantes

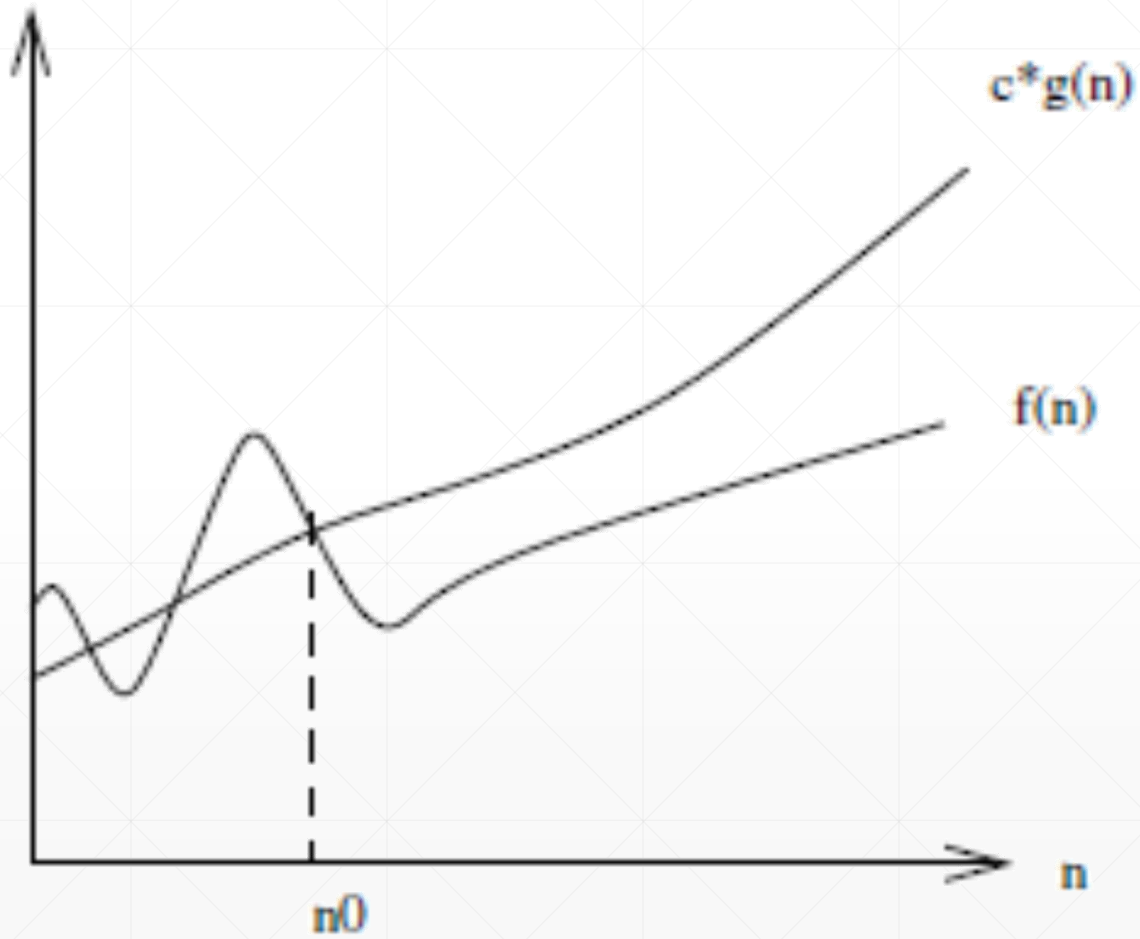
$$f(n) = 2n \qquad g(n) = n$$

Definições – Notação O (*Big Oh*)

- **Limite superior (Upper bound)**
  - Ocorre quando f(n) = O(g(n))
  - Ou seja, **c · g(n)** é limite superior de **f(n)**
  - Sendo assim, existe uma constante c com a qual f(n) será sempre ≤ c · g(n), considerando-se um n grande o suficiente ($n \geq n_0$)

$$f(n) \leq c \cdot g(n),\ n \geq n_0$$

Limite superior



Vamos tentar?

- Verifique se **$3n^2 - 100n + 6 = O(n^2)$**
  - Basta encontrar um valor de **c** que demonstre isso

$c = 3$

$3n^2 > 3n^2 - 100n + 6$

Atenção!

$$f(n) = 3n^2 - 100n + 6$$

$$f(n) = O(n^2)$$

Não significa igualdade

Em Análise de Algoritmos **$f(n) = O(n^2)$** significa que f(n) é $O(n^2)$.

Isso se chama **abuso de notação**.

Faça você mesmo

- Verifique se **$3n^2 - 100n + 6 = O(n)$**

  Não é O(n) porque $c \times n < 3n^2$, **quando n > c**

Mais uma vez

- Verifique se **$3n^2 - 100n + 6 = O(n^3)$**

  $c = 1$

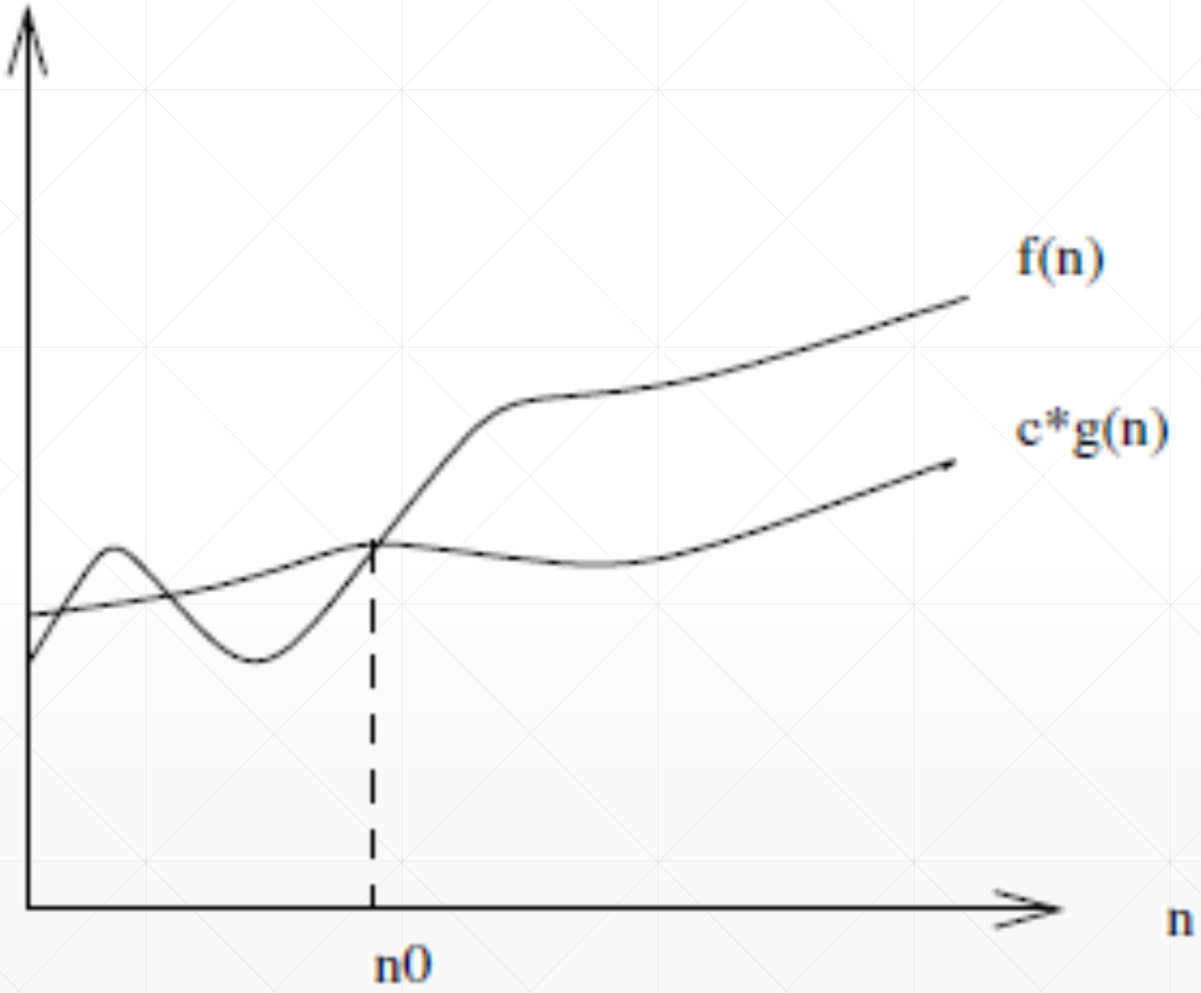
  $n^3 > 3n^2 - 100n + 6$, quando $n > 3$

Definições – Notação Ω

- **Limite inferior (lower bound)**
  - Ocorre quando $f(n) = \Omega(g(n))$
  - Ou seja, **$c \cdot g(n)$** é limite inferior de **$f(n)$**
  - Sendo assim, existe uma constante c com a qual f(n) será sempre $\geq c \cdot g(n)$, considerando-se um n grande o suficiente ($n \geq n_0$)

$$f(n) \geq c \cdot g(n),\ n \geq n_0$$

Limite inferior



Vamos tentar?

- Verifique se **$3n^2 - 100n + 6 = \Omega(n^2)$**
  - Basta encontrar um valor de **c** que demonstre isso

  c = 2

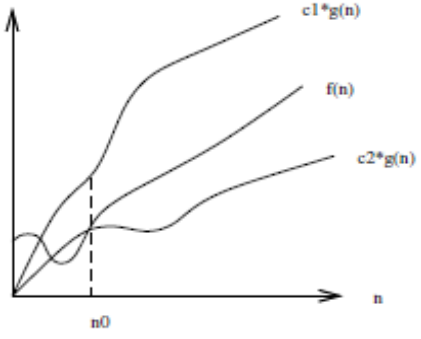
  $2n^2 < 3n^2 - 100n + 6$, quando $n > 100$

Definições – Notação Θ

- **$f(n) = \Theta(g(n))$** significa que **$c_1 \cdot g(n)$** é limite superior de f(n) e **$c_2 \cdot g(n)$** é limite inferior de f(n)
- Sendo assim, existe uma constante **c_1** e uma constante **c_2** que quando multiplicadas a g(n) "envolvem" f(n)

$$c_1 \cdot g(n) \geq f(n) \geq c_2 \cdot g(n),\ n \geq n_0$$

Notação Θ



Vamos tentar?

- Verifique se $\mathbf{3n^2 - 100n + 6 = \Theta(n^2)}$

É verdade! Vimos que f(n) é $O(n^2)$ **e também** é $\Omega(n^2)$

Crescimento e Relações de dominância

Taxas de crescimento

- Vimos que é possível descartar constantes multiplicativas
- As funções abaixo são tratadas de forma idêntica:

$f(n) = 0{,}001n^2$ $g(n) = 1000n^2$

Taxa de crescimento de funções comuns (ns)

1 ns = 0,001 μs

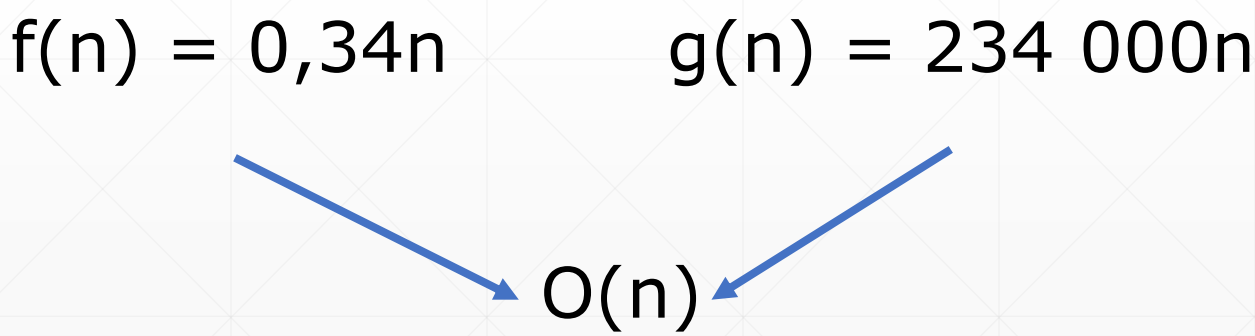
n $f(n)$	$\lg n$	n	$n \lg n$	n^2	2^n	$n!$
10	0.003 μs	0.01 μs	0.033 μs	0.1 μs	1 μs	3.63 ms
20	0.004 μs	0.02 μs	0.086 μs	0.4 μs	1 ms	77.1 years
30	0.005 μs	0.03 μs	0.147 μs	0.9 μs	1 sec	8.4×10^{15} yrs
40	0.005 μs	0.04 μs	0.213 μs	1.6 μs	18.3 min	
50	0.006 μs	0.05 μs	0.282 μs	2.5 μs	13 days	
100	0.007 μs	0.1 μs	0.644 μs	10 μs	4×10^{13} yrs	
1,000	0.010 μs	1.00 μs	9.966 μs	1 ms		
10,000	0.013 μs	10 μs	130 μs	100 ms		
100,000	0.017 μs	0.10 ms	1.67 ms	10 sec		
1,000,000	0.020 μs	1 ms	19.93 ms	16.7 min		
10,000,000	0.023 μs	0.01 sec	0.23 sec	1.16 days		
100,000,000	0.027 μs	0.10 sec	2.66 sec	115.7 days		
1,000,000,000	0.030 μs	1 sec	29.90 sec	31.7 years		

Taxa de crescimento

- Todos os algoritmos utilizam basicamente o mesmo tempo quando n = 10
- Qualquer algoritmo com **n!** é inútil quando **n ≥ 20**
  - **2^n** é ineficiente quando **n > 40**
- Algoritmos quadráticos cujo custo de execução é **n^2** são úteis até **n = 10 000**
  - Deterioram rápido → **n > 1 000 000 é ineficiente**
- Algoritmos lineares e **n lg n** são aplicáveis para entradas de **1 bilhão de itens**
- Um algoritmo **O(lg n)** não tem dificuldade para lidar com nenhum tamanho de entrada **(n)**

Relações de dominância

- A notação O agrupa as funções em conjuntos de classes
  - Todas as funções de uma mesma classe são consideradas equivalentes



Relações de dominância

- Dizemos que uma função com taxa de crescimento mais rápido domina a mais lenta
  - f(n) = O(g(n)) → g ≫ f
- Algumas poucas classes são utilizadas no curso introdutório de Análise de Algoritmos
  - **Funções constantes, f(n) = 1:**

    É o custo de realizar uma operação simples (e.g. somar dois números, exibir valor na tela). Esta função não depende do valor de n

Relações de dominância

- **Funções logarítmicas, f(n) = log n**

  Funções que crescem lentamente a medida que o valor de n cresce. O algoritmo da busca binária está nesta classe.

- **Funções lineares, f(n) = n**

  São funções que medem o custo de olhar cada item em n pelo menos uma vez (ou duas, ou dez vezes...). Algoritmos que buscam pelo maior ou menor valor em um vetor, e aqueles que calculam o valor médio.

Relações de dominância

- **Funções superlineares, $f(n) = n \lg n$:**

  Crescem um pouco mais rápido do que as lineares. Esta classe de funções surge em algoritmos como **Quicksort** e **Mergesort**.

- **Funções quadráticas, $f(n) = n^2$:**

  As funções desta classe medem o custo de olhar em todos os pares de itens em n (ou quase todos os pares). Aparece nos algoritmos **insertion sort** e **selection sort**.

Relações de dominância

- **Funções cúbicas, $f(n) = n^3$:**

  Estas funções ocorrem quando enumeramos todas as triplas de itens possíveis em n. Ocorre em alguns algoritmos de programação dinâmica.

- **Funções exponenciais, $f(n) = c^n$, $c > 1$:**

  Ocorre quando enumera-se todos os subconjuntos possíveis de n-itens em n. Algoritmos desta classe se tornam inúteis rápido.

- **Funções fatoriais, $f(n) = n!$:**

  Ocorre quando é gerada todas as permutações de n itens. É a função que cresce mais rapidamente entre as que vamos estudar.

Relações de dominância

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

Atividade

Operações com a notação O

Somando funções

- A soma de duas funções é governada pela função dominante:

$$O(f(n)) + O(g(n)) \rightarrow O(\mathbf{max(}f(n), g(n)\mathbf{)})$$

Exemplo

$$n^3 + n^2 + n + 1 = \mathbf{O(n^3)}$$

Multiplicando funções

- Multiplicar é realizar uma mesma soma repetidas vezes
- Multiplicar uma função por um valor constante não afeta o seu comportamento assintótico
  - Considerando que o valor constante é maior do que zero ($c > 0$)

$$O(c \cdot f(n)) \rightarrow O(f(n))$$

$$\Omega(c \cdot f(n)) \rightarrow \Omega(f(n))$$

$$\Theta(c \cdot f(n)) \rightarrow \Theta(f(n))$$

Multiplicando funções

$$O(f(n)) * O(g(n)) \rightarrow O(f(n) * g(n))$$

$$\Omega(f(n)) * \Omega(g(n)) \rightarrow \Omega(f(n) * g(n))$$

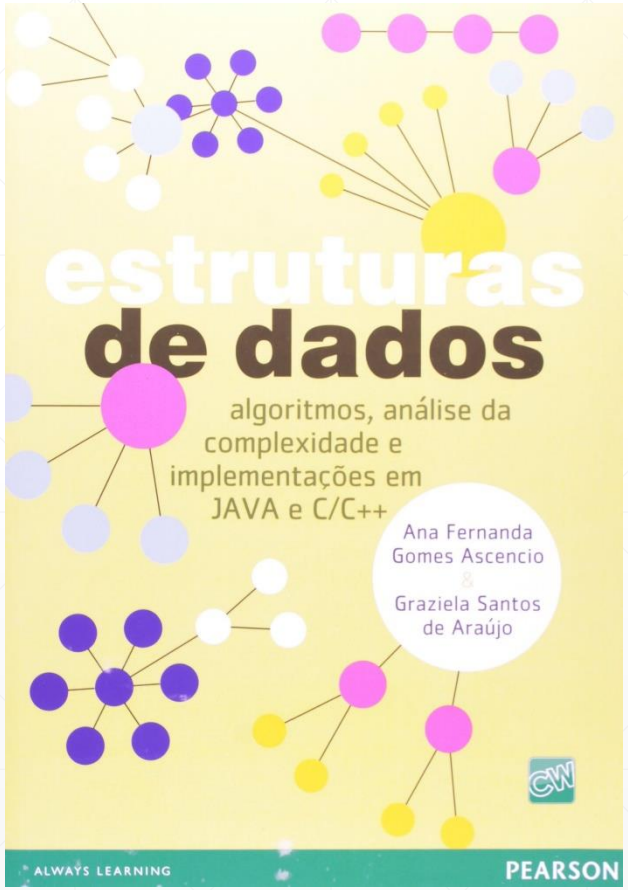
$$\Theta(f(n)) * \Theta(g(n)) \rightarrow \Theta(f(n) * g(n))$$

Resumo

- Podemos conduzir a análise de um algoritmo em particular ou de uma classe de algoritmos
- Usamos a palavra custo para se referir ao tempo de execução de um algoritmo
- Usamos uma função matemática para descrever o custo de um algoritmo a partir de uma determinada entrada
- A notação O é utilizada para comparar funções de custo e remover detalhes pouco úteis para a comparação de algoritmos

Dúvidas?

Referências Bibliográficas



Referências Bibliográficas