Fundamentos de Análise de Complexidade

Unidade I: Análise de Algoritmos

PUC Minas Virtual

Agenda

- Conceitos básicos da Matemática
- Noções de complexidade
- Aspectos da análise de algoritmos
- Notações Θ, O e Ω
- Exercícios

Conceitos Básicos da Matemática

Exercício Resolvido (1): Resolva as Equações

a) $2^{10} =$

b) $\lg(1024) =$

c) $\lg(17) =$

d) $\lceil \lg(17) \rceil =$

e) $\lfloor \lg(17) \rfloor =$

Exercício Resolvido (1): Resolva as Equações

a) $2^{10} = \mathbf{1024}$

b) $\lg(1024) = \mathbf{10}$

c) $\lg(17) = \mathbf{4{,}08746284125034}$

d) $\lceil \lg(17) \rceil = \mathbf{5}$

e) $\lfloor \lg(17) \rfloor = \mathbf{4}$

Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

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

c) $f(n) = n \times lg(n)$

d) $f(n) = n$

e) $f(n) = sqrt(n)$

f) $f(n) = lg(n)$

Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

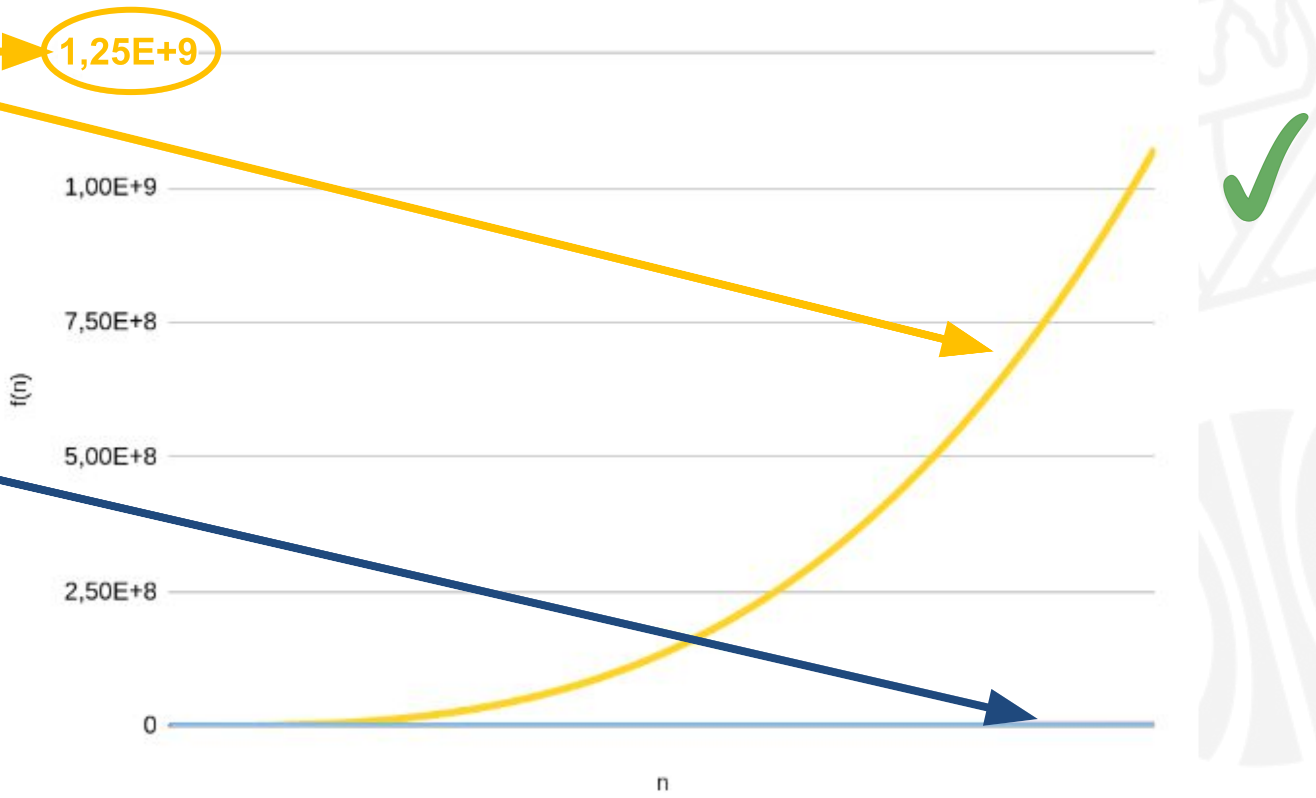
b) $f(n) = n^2$

c) $f(n) = n \times lg(n)$

d) $f(n) = n$

e) $f(n) = sqrt(n)$

f) $f(n) = lg(n)$



Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

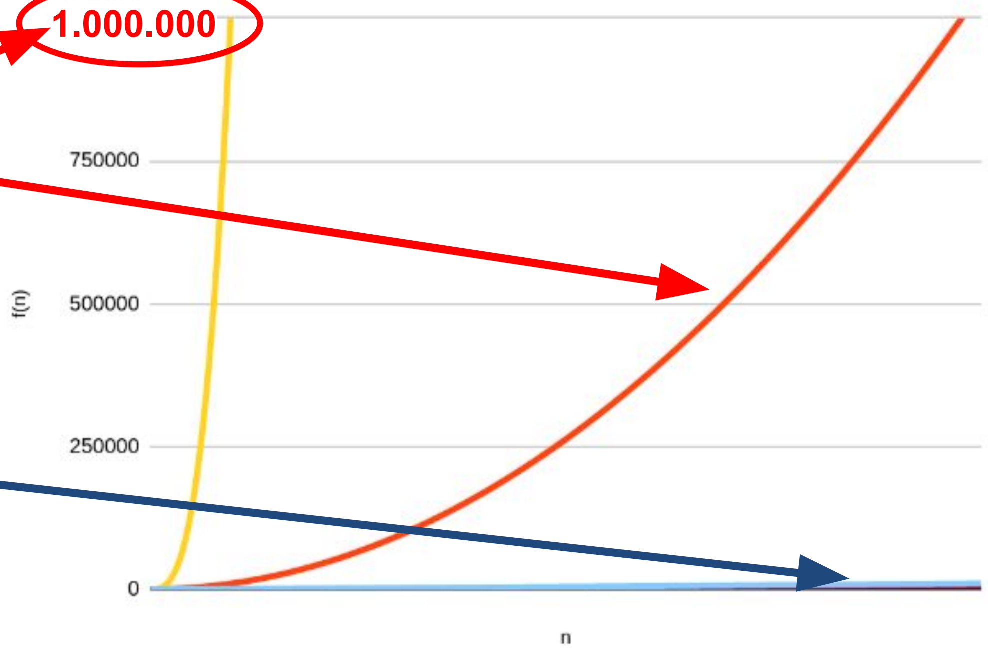
b) $f(n) = n^2$

c) f(n) = n x lg(n)

d) f(n) = n

e) f(n) = sqrt(n)

f) f(n) = lg(n)



Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

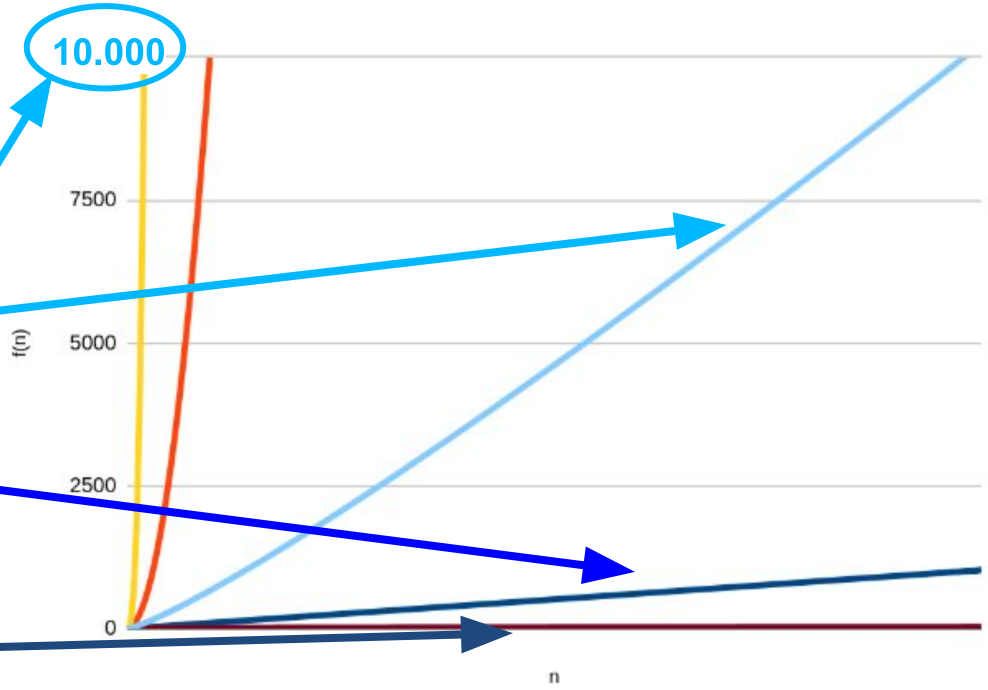
b) $f(n) = n^2$

c) $f(n) = n \times lg(n)$

d) $f(n) = n$

e) $f(n) = sqrt(n)$

f) $f(n) = lg(n)$



Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

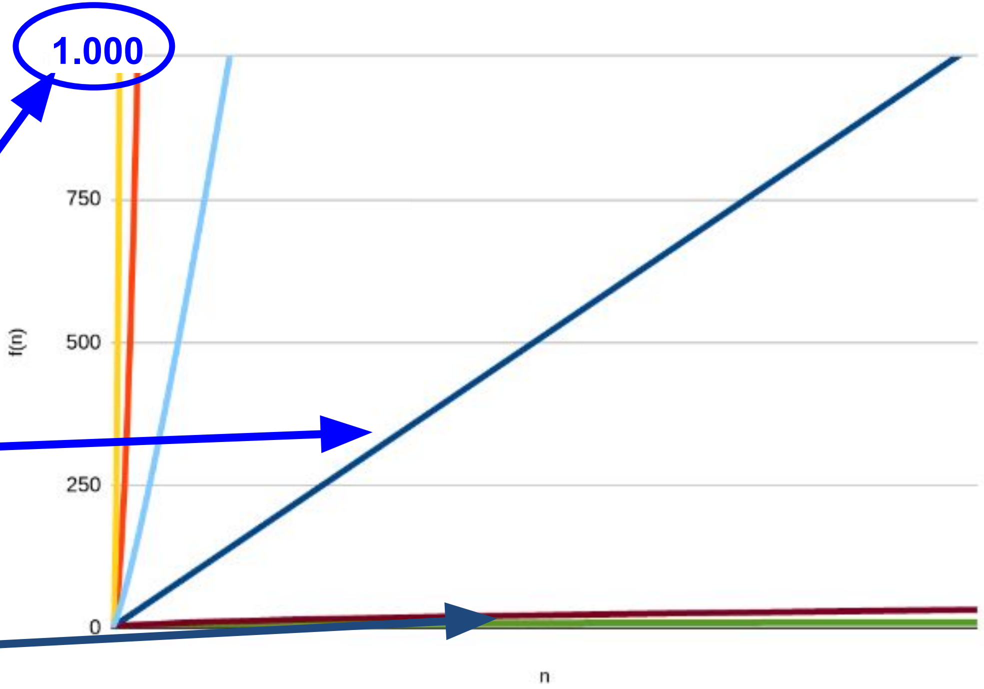
b) $f(n) = n^2$

c) $f(n) = n \times lg(n)$

d) $f(n) = n$

e) $f(n) = sqrt(n)$

f) $f(n) = lg(n)$



Exercício Resolvido (2): Plote os Gráficos

a) $f(n) = n^3$

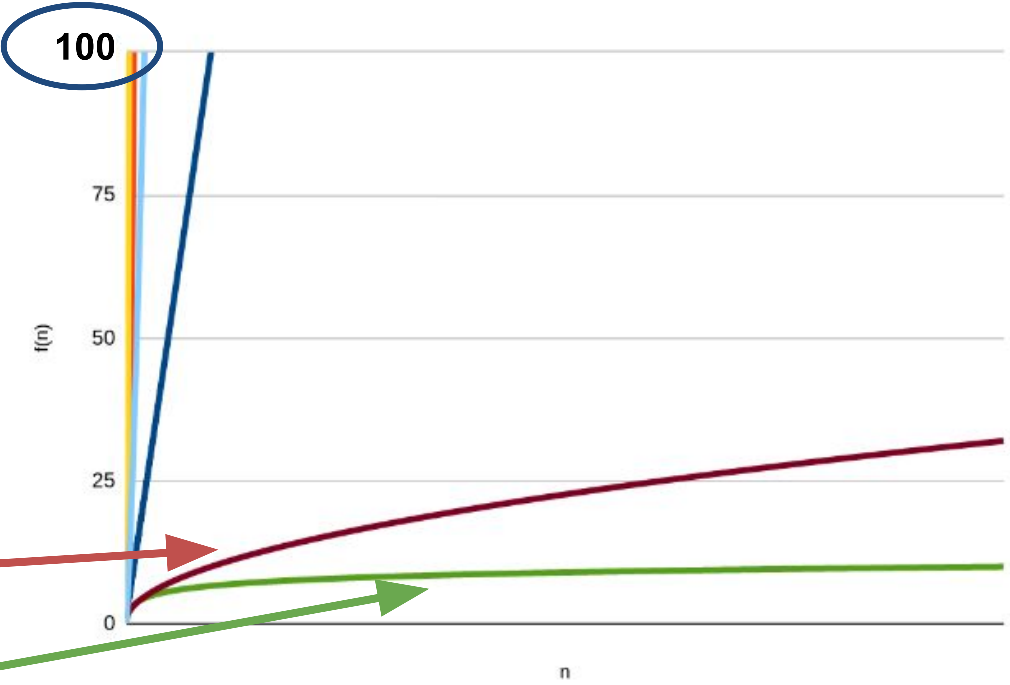
b) $f(n) = n^2$

c) $f(n) = n \times lg(n)$

d) $f(n) = n$

e) $f(n) = sqrt(n)$

f) $f(n) = lg(n)$



Noções de Complexidade

Contagem de Operações

- Trivial em estruturas sequenciais
- Nas estruturas condicionais, consideramos o custo da condição mais ou a lista verdadeira ou a lista falsa
- Nas estruturas de repetição, consideramos o número de repetições que podem ser simples, duplas ou com custo logarítmico
- Podem ter pior, melhor e caso médio

Funções de Complexidade

- Mensuram a quantidade de recursos (como tempo e espaço) que um algoritmo requer à medida que o tamanho da entrada aumenta

- Um exemplo do Algoritmo de ordenação por Seleção:

$$c(n) = \frac{n^2}{2} - \frac{n}{2} \qquad m(n) = 3n - 3$$

  - Supondo que n = 100, temos:

$$c(100) = \frac{100^2}{2} - \frac{100}{2} = 4950 \qquad m(100) = 3 \times 100 - 3 = 997$$

Notação Θ

- Indica a tendência de crescimento de uma função de complexidade
- Ignora as constantes e os termos com menor crescimento das funções de complexidade
- Por exemplo:
    - $f(n) = 3n + 2n^2$ operações é $\Theta(n^2)$
    - $f(n) = 5n + 4n^3$ operações é $\Theta(n^3)$
    - $f(n) = lg(n) + n$ operações é $\Theta(n)$

Exercício Resolvido (3)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
if (a - 5 < b - 3){
        i--;
        --b;
        a -= 3;
} else {
        j--;
}
```

PUC Minas Virtual

Exercício Resolvido (3)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
if (a - 5 < b - 3){
        i--;
        --b;
        a -= 3;
} else {
        j--;
}
```

Melhor caso	Pior caso
f(n) = 3, $\Theta(1)$	f(n) = 5, $\Theta(1)$

Exercício Resolvido (4)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
        a--;
        b--;
}
```

Exercício Resolvido (4)

- Calcule o **número de subtrações** que o código abaixo realiza:

✓

```
...
for (int i = 0; i < n; i++){
        a--;
        b--;
}
```

Todos os casos
f(n) = 2n, Θ(n)

Exercício Resolvido (5)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
                a--;
                b--;
                c--;
        }
}
```

Exercício Resolvido (5)

- Calcule o **número de subtrações** que o código abaixo realiza:

```
...
for (int i = 0; i < n; i++){
        for (int j = 0; j < n; j++){
                a--;
                b--;
                c--;
        }
}
```

Todos os casos
$f(n) = 3n^2$, $\Theta(n^2)$

Exercício Resolvido (6)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n; i > 0; i /= 2){
        a *= 2;
}
```

Exercício Resolvido (6)

- Calcule o **número de multiplicações** que o código abaixo realiza:

```
...
for (int i = n; i > 0; i /= 2){
      a *= 2;
}
```

Todos os casos
$f(n) = \lfloor lg(n) \rfloor + 1$, $\Theta(lg\ n)$

Aspectos da Análise de Algoritmos

PUC Minas Virtual

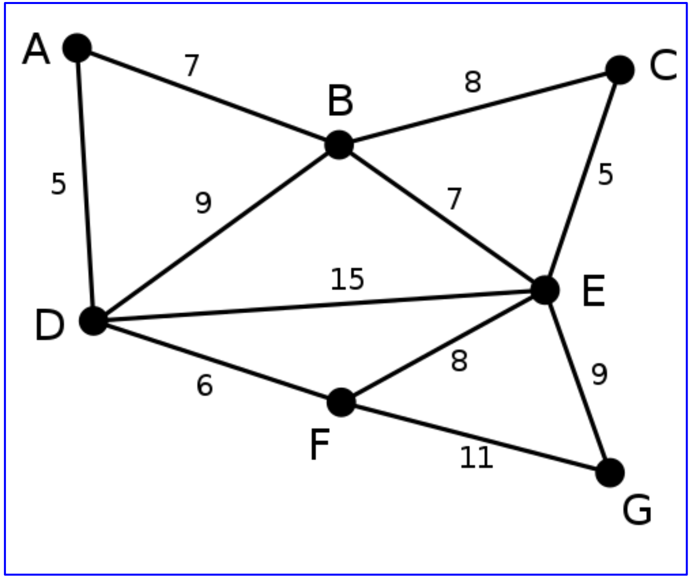
Restrição dos Algoritmos

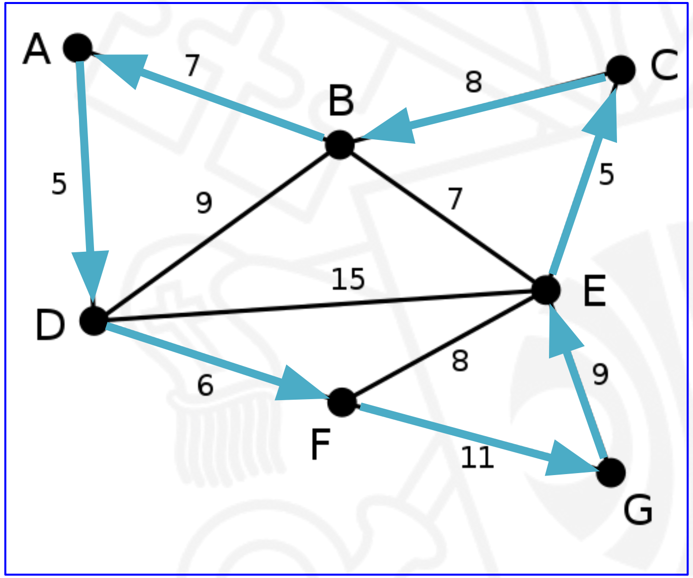
- Nossos algoritmos devem ser implementados em um computador
- Restrições do computador: capacidade computacional e armazenamento
- Logo, devemos analisar a complexidade de se implementar algoritmos

Um algoritmo que leva séculos para terminar é uma opção inadequada

Exemplo (Rascunho) de Algoritmo NP-Completo

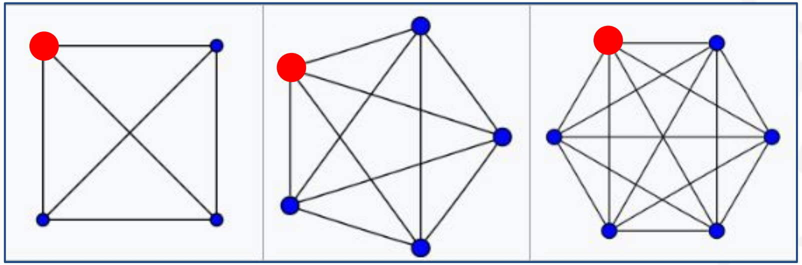
- Problema do Caixeiro Viajante





Exemplo (Rascunho) de Algoritmo NP-Completo

- Problema do Caixeiro Viajante



Número de combinações:

__ __ __	__ __ __ __	__ __ __ __ __
3 2 1	4 3 2 1	5 4 3 2 1

Exemplo (Rascunho) de Algoritmo NP-Completo

- Rascunho do algoritmo força bruta para encontrar a solução ótima do PCV

Número de cidades	Tempo de execução
5	5 s
6	5 x (5s) = 25 s
7	6 x (25s) = 150 s = 2,5 min
8	7 x (2,5 min) = 17,5 min
9	8 x (17,5 min) = 140 min = 2,34 h
10	9 x (2,34 h) = 21 h
11	10 x (21 h) = 210 = 8,75 dias
12	11 x (8,75 dias) = 96,25 dias
13	12 x (96,25 dias) = 1155 = 3,15 anos
14	13 x (3,15 anos) = 41,02 anos
15	14 x (41,02 anos) = 5,74 séculos
16	15 x (5,74 séculos) = 8,6 milênios

Exemplo (Rascunho) de Algoritmo NP-Completo

- Rascunho do algoritmo força bruta para encontrar a solução ótima do PCV

Observação (1): Na verdade, a solução ótima para o PCV é 2x mais rápida que a apresentada, contudo, isso é “indiferente” na tendência de crescimento

Observação (2): Se tivermos um computador 100 vezes mais rápido, isso também será “indiferente” na tendência de crescimento

Métricas para a Análise de Complexidade

- Tempo de execução
- Espaço de memória ocupado
- Energia
- Outros…

Tipos de Análise de Complexidade

- **Análise de um algoritmo particular**: analisamos o custo de um algoritmo específico para um problema específico

- **Análise de uma classe (ou família) de algoritmos**: analisamos o menor custo possível para resolver um problema específico
  - Todo problema tem um nível mínimo de dificuldade para ser resolvido

Como Medir o Custo de um Algoritmo

?

Restrições no Modelo do Cronômetro

- Hardware
- Arquitetura
- Sistema Operacional
- Linguagem
- Compilador

Exemplo de Otimização do Compilador

```
for (int i = 0; i < 20 ; i++){
        array[i] = i;
}
```

Qual é a vantagem de cada um dos códigos?

```
array [0] = 0;
array [1] = 1;
...
array [19] = 19;
```

Ainda sobre Otimização de Compiladores ...

- Frequentemente, alunos fazem otimizações desnecessárias em termos de eficiência

- Por exemplo, frequentemente, o compilador gera o mesmo código objeto para if-else-if e switch-case; for e while; entre outros...

Como Medir o Custo de um Algoritmo

?

Como Medir o Custo de um Algoritmo

Modelo

Matemático

Modelo Matemático para Contar Operações

- Determinamos e contamos as operações relevantes
- O custo total de um algoritmo é igual a soma do custo de suas operações
- Desconsideramos sobrecargas de gerenciamento de memória ou E/S
- A menos que dito o contrário, consideramos o pior caso
- Precisamos definir a função de complexidade

Algoritmo Ótimo

- Algoritmo cujo custo é igual ao menor custo possível

Exercício Resolvido (7): Pesquisa Sequencial

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Este algoritmo é ótimo?

Exercício Resolvido (7): Pesquisa Sequencial

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
        if (array[i] == x){
                resp = true;
                i = n;
        }
}
```

Este algoritmo é ótimo?

Pause!

Exercício Resolvido (7): Pesquisa Sequencial

- Apresente a função de complexidade de tempo (número de **comparações entre elementos do *array***) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Melhor caso: elemento desejado na primeira posição

$t(n) = 1$

Pior caso: elemento desejado não está no array ou está na última posição

$t(n) = n$

Exercício Resolvido (7): Pesquisa Sequencial

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Melhor caso: elemento desejado na primeira posição

$t(n) = 1$

Pior caso: elemento desejado não está no array ou está na última posição

$t(n) = n$

Este algoritmo é ótimo?

Exercício Resolvido (7): Pesquisa Sequencial

- Apresente a função de complexidade de tempo (número de comparações entre elementos do *array*) da pesquisa sequencial no melhor e no pior caso

```
boolean resp = false;

for (int i = 0; i < n; i++){
    if (array[i] == x){
        resp = true;
        i = n;
    }
}
```

Melhor caso: elemento desejado na primeira posição

$$t(n) = 1$$

Pior caso: elemento desejado não está no array ou está na última posição

$$t(n) = n$$

Este algoritmo é ótimo? Sim porque temos que testar todos os elementos para garantir nossa resposta

Exercício Resolvido (8)

- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

Exercício Resolvido (8)

- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

Pause!

Exercício Resolvido (8)

- Um aluno deve procurar um valor em um *array* de números reais. Ele tem duas alternativas. Primeiro, executar uma pesquisa sequencial. Segundo, ordenar o *array* e, em seguida, aplicar uma pesquisa binária. O que fazer?

O aluno deve escolher a primeira opção, pois a pesquisa sequencial tem custo $\Theta(n)$. A segunda opção tem custo $\Theta(n \times \lg n)$ para ordenar mais $\Theta(\lg n)$ para a pesquisa binária

✓

Notações Θ, O e Ω

Notações Θ, O e Ω

- Regras gerais
- Operações
- Definições

Regras Gerais para das Notações Θ, O e Ω

- Indica a tendência de crescimento de uma função de complexidade
- Ignora as constantes e os termos com menor crescimento das funções de complexidade
- Por exemplo:
  - $f(n) = 3n + 2n^2$ operações é $\Theta(n^2)$, $O(n^2)$ e $\Omega(n^2)$
  - $f(n) = 5n + 4n^3$ operações é $\Theta(n^3)$, $O(n^3)$ e $\Omega(n^3)$
  - $f(n) = lg(n) + n$ operações é $\Theta(n)$, $O(n)$ e $\Omega(n)$

Diferença entre as Notações Θ, O e Ω

- Θ é o limite justo
- O é o limite superior
- Ω é o limite inferior

Diferença entre as Notações Θ, O e Ω

- **O é o limite superior**, logo, se um algoritmo é O(f(n)), ele também será O(g(n)) para toda função g(n) tal que “g(n) é **maior** que f(n)”

- ***Ω* é o limite inferior**, logo, se um algoritmo é Ω(f(n)), ele também será Ω(g(n)) para toda função g(n) tal que “g(n) é **menor** que f(n)”

- **Θ é o limite justo**, logo, g(n) é O(f(n)) *and* Ω(f(n)) se e somente se g(n) é Θ(f(n))

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:
  a) $3n^2 + 5n + 1$ é $O(n)$:
  b) $3n^2 + 5n + 1$ é $O(n^2)$:
  c) $3n^2 + 5n + 1$ é $O(n^3)$:
  d) $3n^2 + 5n + 1$ é $\Omega(n)$:
  e) $3n^2 + 5n + 1$ é $\Omega(n^2)$:
  f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:
  g) $3n^2 + 5n + 1$ é $\Theta(n)$:
  h) $3n^2 + 5n + 1$ é $\Theta(n^2)$:
  i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:

a) $3n^2 + 5n + 1$ é $O(n)$:

b) $3n^2 + 5n + 1$ é $O(n^2)$:

c) $3n^2 + 5n + 1$ é $O(n^3)$:

d) $3n^2 + 5n + 1$ é $\Omega(n)$:

e) $3n^2 + 5n + 1$ é $\Omega(n^2)$:

f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:

g) $3n^2 + 5n + 1$ é $\Theta(n)$:

h) $3n^2 + 5n + 1$ é $\Theta(n^2)$:

i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

Pause!

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:
  a) $3n^2 + 5n + 1$ é $O(n)$:
  b) **$3n^2 + 5n + 1$ é $O(n^2)$: verdadeira**
  c) $3n^2 + 5n + 1$ é $O(n^3)$:
  d) $3n^2 + 5n + 1$ é $\Omega(n)$:
  e) **$3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira**
  f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:
  g) $3n^2 + 5n + 1$ é $\Theta(n)$:
  h) **$3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira**
  i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:
    a) $3n^2 + 5n + 1$ é $O(n)$:
    b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
    c) **$3n^2 + 5n + 1$ é $O(n^3)$: verdadeira**
    d) **$3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira**
    e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
    f) $3n^2 + 5n + 1$ é $\Omega(n^3)$:
    g) $3n^2 + 5n + 1$ é $\Theta(n)$:
    h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
    i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:
  a) **$3n^2 + 5n + 1$ é $O(n)$: falsa**
  b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
  c) $3n^2 + 5n + 1$ é $O(n^3)$: verdadeira
  d) $3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira
  e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
  f) **$3n^2 + 5n + 1$ é $\Omega(n^3)$: falsa**
  g) $3n^2 + 5n + 1$ é $\Theta(n)$:
  h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
  i) $3n^2 + 5n + 1$ é $\Theta(n^3)$:

✓

Exercício Resolvido (9)

- Responda se as afirmações são verdadeiras ou falsas:
  a) $3n^2 + 5n + 1$ é $O(n)$: falsa
  b) $3n^2 + 5n + 1$ é $O(n^2)$: verdadeira
  c) $3n^2 + 5n + 1$ é $O(n^3)$: verdadeira
  d) $3n^2 + 5n + 1$ é $\Omega(n)$: verdadeira
  e) $3n^2 + 5n + 1$ é $\Omega(n^2)$: verdadeira
  f) $3n^2 + 5n + 1$ é $\Omega(n^3)$: falsa
  g) $3n^2 + 5n + 1$ é $\Theta(n)$: falsa
  h) $3n^2 + 5n + 1$ é $\Theta(n^2)$: verdadeira
  i) $3n^2 + 5n + 1$ é $\Theta(n^3)$: falsa

✓

Operações com as Notações Θ, O e Ω

1) $f(n) = \Theta(f(n))$
2) $c \times \Theta(f(n)) = \Theta(f(n))$
3) $\Theta(f(n)) + \Theta(f(n)) = \Theta(f(n))$
4) $\Theta(\Theta(f(n))) = \Theta(f(n))$
5) $\Theta(f(n)) + \Theta(g(n)) = \Theta(\text{máximo}(f(n),g(n)))$
6) $\Theta(f(n)) \times \Theta(g(n)) = \Theta(f(n) \times g(n))$
7) $f(n) \times \Theta(g(n)) = \Theta(f(n) \times g(n))$

***) As mesmas propriedades são aplicadas para Ω e O**

Exercício Resolvido (10)

- Sabendo que o Algoritmo de Seleção faz $\Theta(n^2)$ comparações entre registros, quantas dessas comparações temos no código abaixo? Justifique

```
for (int i = 0; i < n; i++){
        seleção();
}
```

Exercício Resolvido (10)

- Sabendo que o Algoritmo de Seleção faz $\Theta(n^2)$ comparações entre registros, quantas dessas comparações temos no código abaixo? Justifique

```
for (int i = 0; i < n; i++){
        seleção();
}
```

Pause!

Exercício Resolvido (10)

- Sabendo que o Algoritmo de Seleção faz $\Theta(n^2)$ comparações entre registros, quantas dessas comparações temos no código abaixo? Justifique

```
for (int i = 0; i < n; i++){
        seleção();
}
```

✓

RESPOSTA: Neste caso, executamos o Seleção n vezes: $n \times \Theta(n^2) = \Theta(n^3)$

Exercício Resolvido (11)

- Dado $f(n) = 3n^2 - 5n - 9$, $g(n) = n.lg(n)$, $l(n) = n.lg^2(n)$ e $h(n) = 99n^8$, qual é a ordem de complexidade das operações abaixo (use a notação Θ):

a) $h(n) + g(n) - f(n)$

b) $\Theta(h(n)) + \Theta(g(n)) - \Theta(f(n))$

c) $f(n)$ *x* $g(n)$

d) $g(n)$ *x* $l(n) + h(n)$

e) $f(n)$ *x* $g(n)$ *x* $l(n)$

f) $\Theta(\Theta(\Theta(\Theta(f(n)))))$

Exercício Resolvido (11)

- Dado $f(n) = 3n^2 - 5n - 9$, $g(n) = n.lg(n)$, $l(n) = n.lg^2(n)$ e $h(n) = 99n^8$, qual é a ordem de complexidade das operações abaixo (use a notação Θ):

a) $h(n) + g(n) - f(n)$

b) $\Theta(h(n)) + \Theta(g(n)) - \Theta(f(n))$

c) $f(n) \; x \; g(n)$

d) $g(n) \; x \; l(n) + h(n)$

e) $f(n) \; x \; g(n) \; x \; l(n)$

f) $\Theta(\Theta(\Theta(\Theta(f(n)))))$

Pause!

Exercício Resolvido (11)

- Dado $f(n) = 3n^2 - 5n - 9$, $g(n) = n.lg(n)$, $l(n) = n.lg^2(n)$ e $h(n) = 99n^8$, qual é a ordem de complexidade das operações abaixo (use a notação Θ):

    a) $h(n) + g(n) - f(n) \Rightarrow [99n^8] + [n.lg(n)] - [3n^2\text{-}5n\text{-}9] \Rightarrow \Theta(n^8)$

    b) $\Theta(h(n)) + \Theta(g(n)) - \Theta(f(n)) \Rightarrow \Theta(n^8) + \Theta(n.lg(n)) - \Theta(n^2) \Rightarrow \Theta(n^8)$

    c) $f(n)\ x\ g(n) \Rightarrow \Theta(n^2) \text{ x } \Theta(n.lg(n)) \Rightarrow \Theta(n^3.lg(n))$

    d) $g(n)\ x\ l(n) + h(n) \Rightarrow \Theta(n.lg(n)) \text{ x } \Theta(n.lg^2(n)) + \Theta(n^8) \Rightarrow \Theta(n^8)$

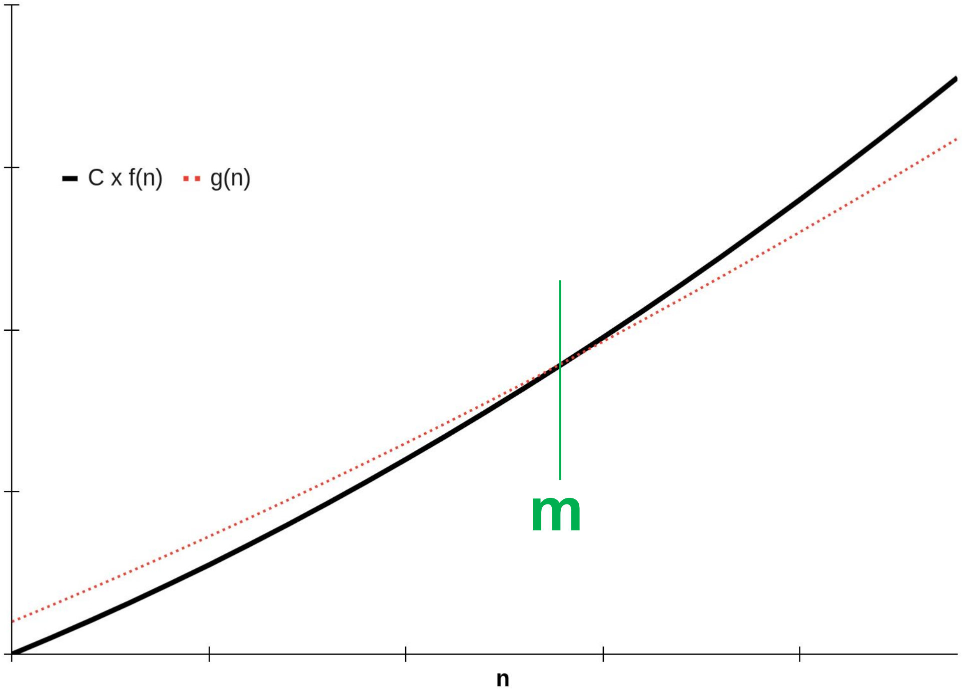
    e) $f(n)\ x\ g(n)\ x\ l(n) \Rightarrow \Theta(n^2) \text{ x } \Theta(n.lg(n)) \text{ x } \Theta(n.lg^2(n)) \Rightarrow \Theta(n^4.lg^3(n))$

    f) $\Theta(\Theta(\Theta(\Theta(f(n))))) \Rightarrow \Theta(n)$

✓

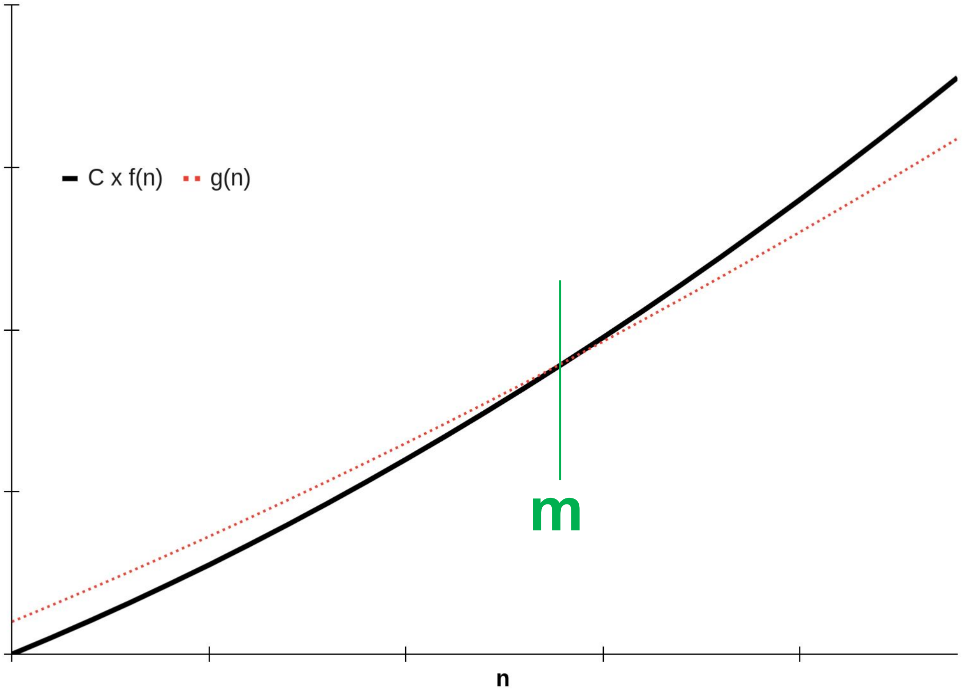
Definição da Notação O

- **g(n) é O(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para *n* ≥ *m*, temos que |g(n)| ≤ c x |f(n)|



Definição da Notação O

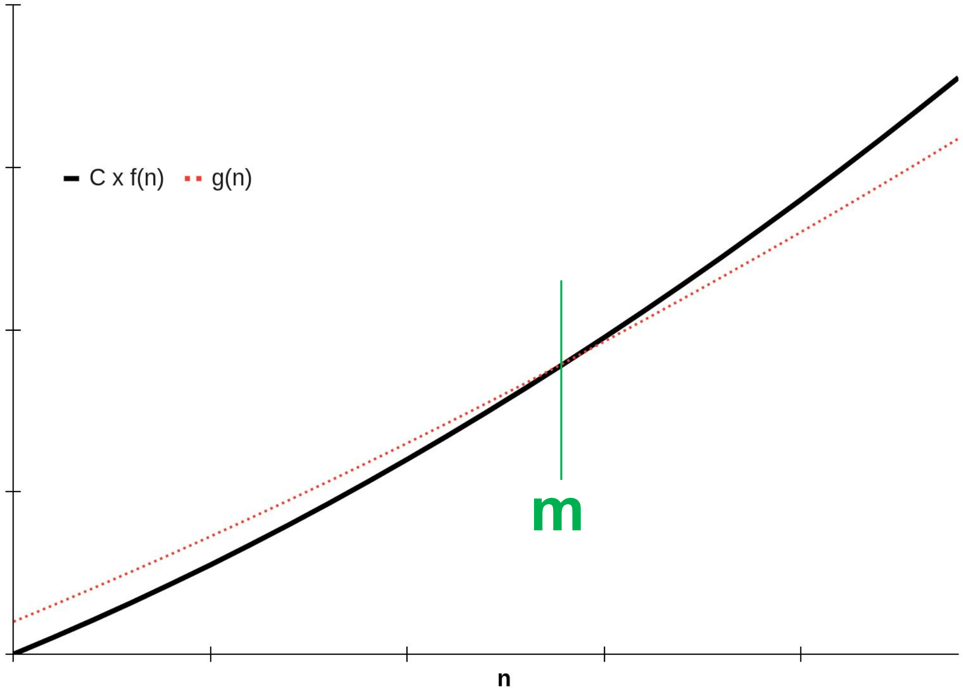
- **g(n) é O(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para $n \geq m$, temos que $|g(n)| \leq c \times |f(n)|$



f(n) é um limite assintótico superior para g(n)

Definição da Notação O

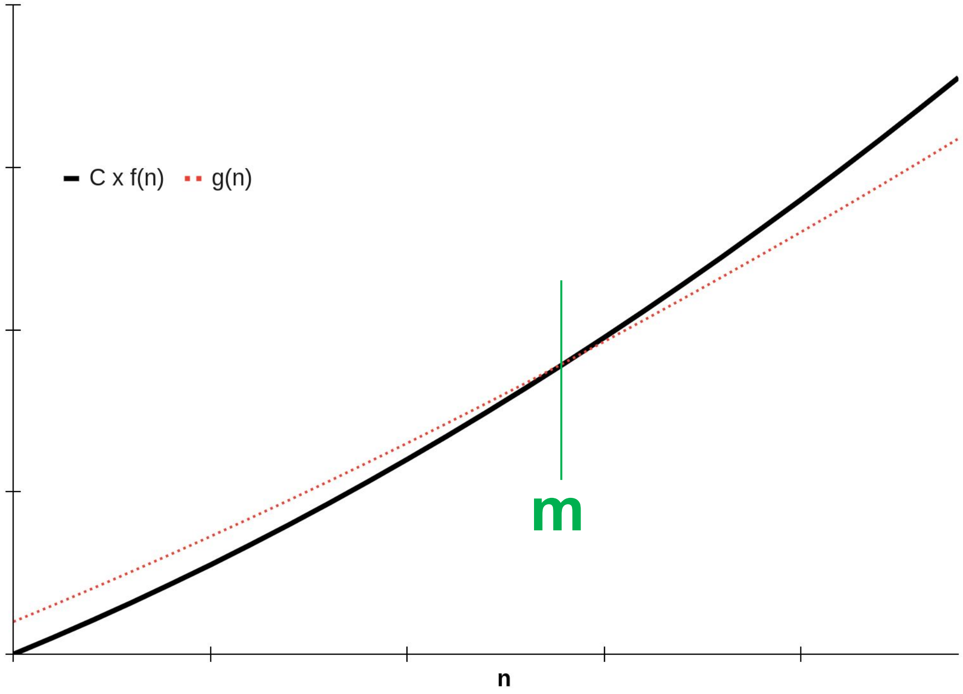
- **g(n) é O(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para $n \geq m$, temos que $|g(n)| \leq c \times |f(n)|$



f(n) domina assintoticamente g(n)

Definição da Notação O

- **g(n) é O(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para $n \geq m$, temos que $|g(n)| \leq c \times |f(n)|$



O comportamento assintótico das funções representa o limite quando n cresce

Exercício Resolvido (12)

- Dada a definição da notação O:

  a) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^2|$, provando que $3n^2 + 5n + 1$ é $O(n^2)$

  b) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^3|$, provando que $3n^2 + 5n + 1$ é $O(n^3)$

  c) Prove que $3n^2 + 5n + 1$ **não é** $O(n)$

Exercício Resolvido (12)

- Dada a definição da notação O:

a) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^2|$, provando que $3n^2 + 5n + 1$ é $O(n^2)$

b) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^3|$, provando que $3n^2 + 5n + 1$ é $O(n^3)$

c) Prove que $3n^2 + 5n + 1$ **não é** $O(n)$

Pause!

Exercício Resolvido (12)

- Dada a definição da notação O:

  a) Mostre os valores de c e m tal que, para $n \geq m$, $\boxed{|3n^2 + 5n + 1| \leq c \times |n^2|}$, provando que $3n^2 + 5n + 1$ é $O(n^2)$

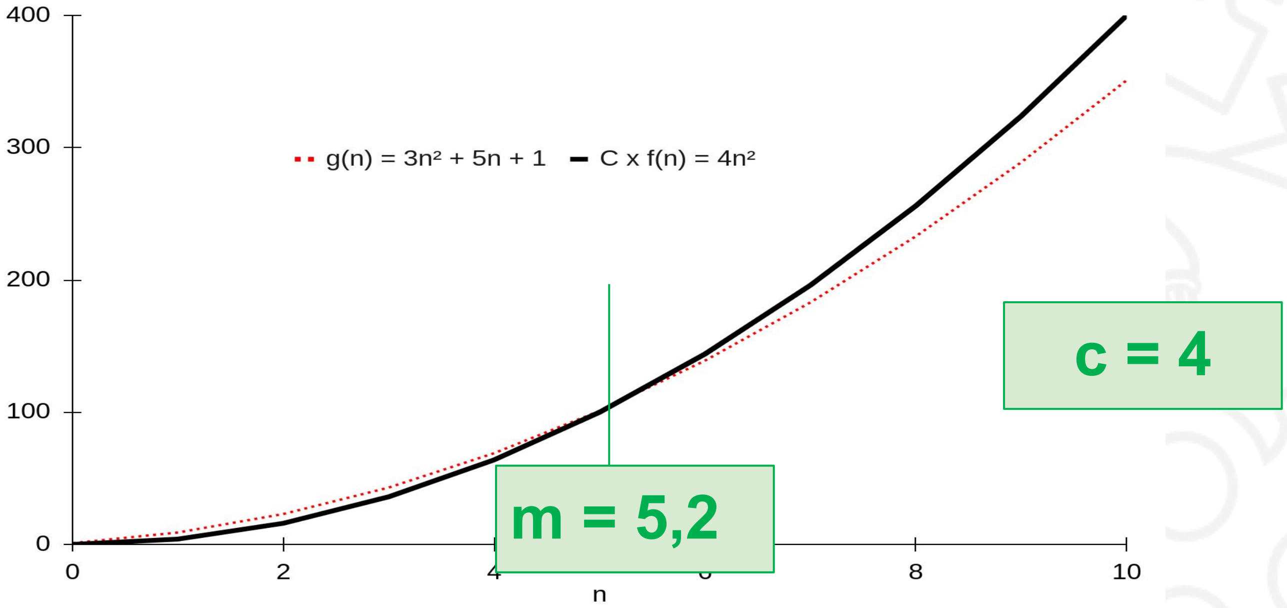
Para que tal inequação seja verdadeira, c tem que ser maior do que três (e.g., quatro)

✓

Exercício Resolvido (12)

- Dada a definição da notação O:

  a) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^2|$, provando que $3n^2 + 5n + 1$ é $O(n^2)$

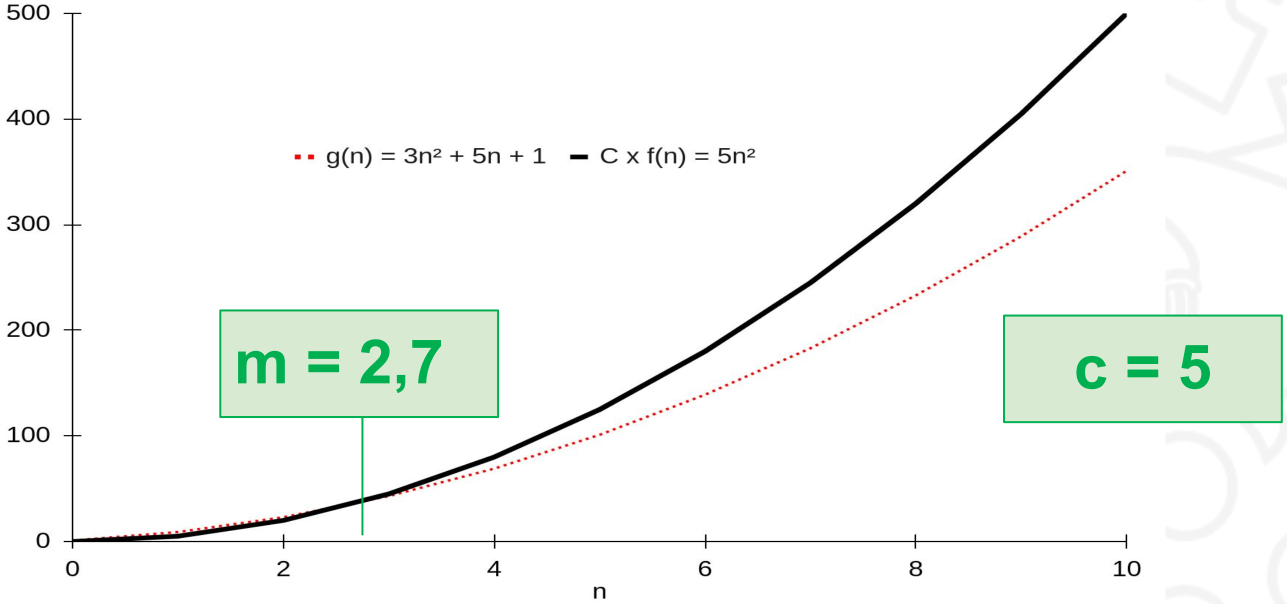


PUC Minas Virtual

Exercício Resolvido (12)

- Dada a definição da notação O:

a) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^2|$, provando que $3n^2 + 5n + 1$ é $O(n^2)$



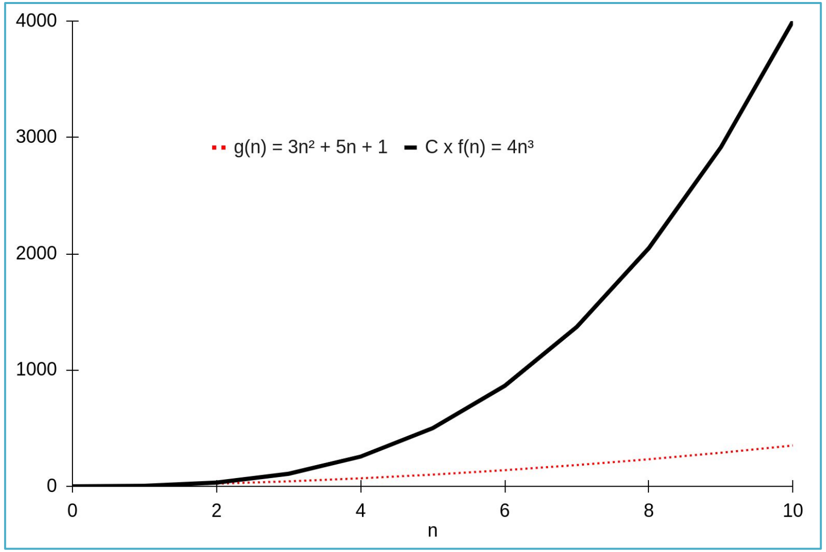
PUC Minas Virtual

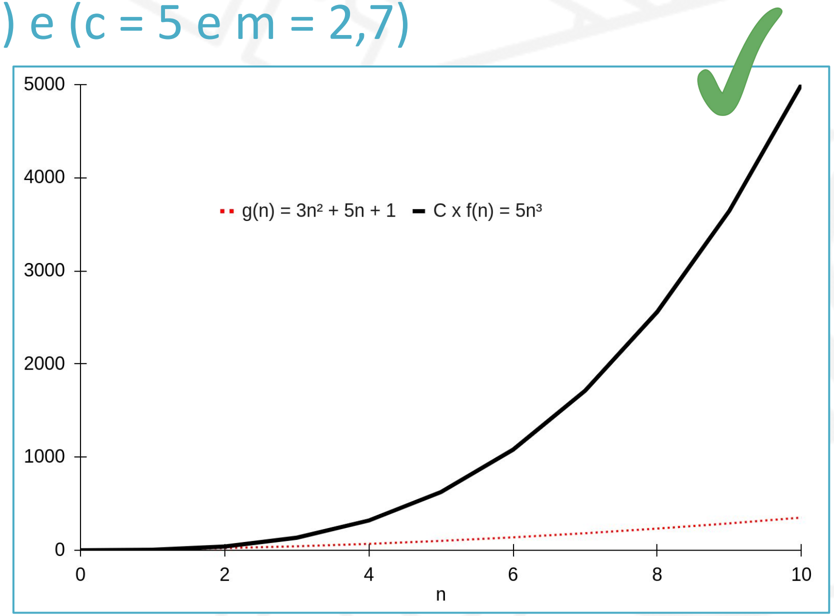
Exercício Resolvido (12)

- Dada a definição da notação O:

b) Mostre os valores de c e m tal que, para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n^3|$, provando que $3n^2 + 5n + 1$ é $O(n^3)$

RESPOSTA: Novamente, (c = 4 e m = 5,7) e (c = 5 e m = 2,7)





Exercício Resolvido (12)

- Dada a definição da notação O:

  c) Prove que $3n^2 + 5n + 1$ **não é** $O(n)$ ✓

RESPOSTA: Não existe par (c, m) tal que para $n \geq m$, $|3n^2 + 5n + 1| \leq c \times |n|$ seja verdadeira. Aumentando o valor de c, apenas retardamos o momento em que a curva quadrática supera a linear

Exercício Resolvido (12)

- Dada a definição da notação O:

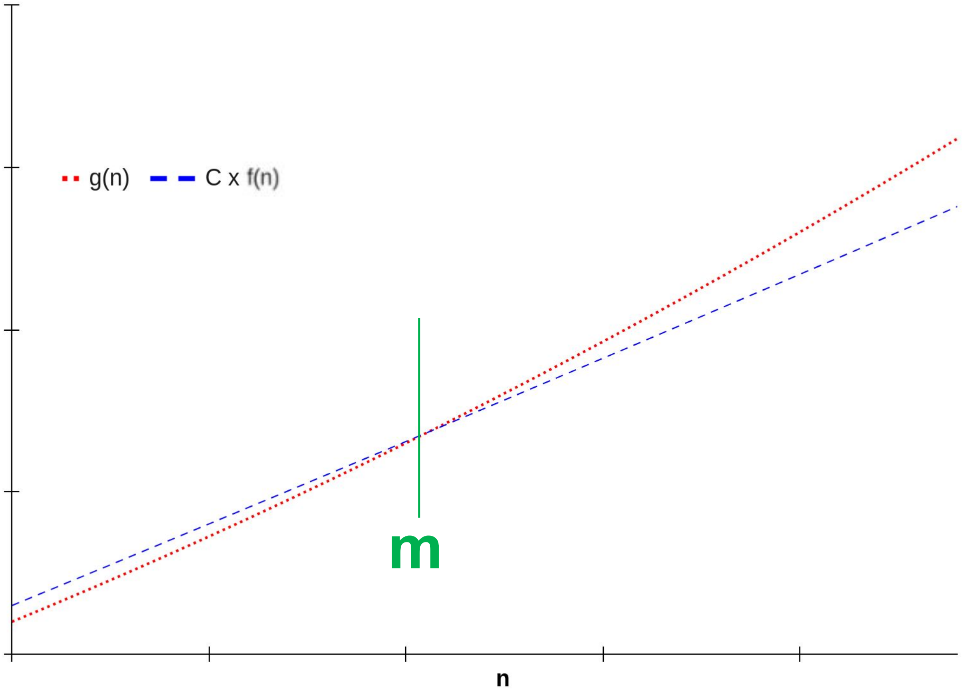
c) Prove que $3n^2 + 5n + 1$ **não é** $O(n)$

Fazendo C = 100		
n	$g(n) = 3n^2 + 5n + 1$	$C \times f(n) = 100 \times n$
0	1	**0**
5	101	**500**
10	351	**1000**
15	751	**1500**
20	1301	**2000**
25	2001	**2500**
30	2851	**3000**
35	**3851**	3500
40	**5001**	4000
45	**6301**	4500
50	**7751**	5000

Fazendo C = 1000		
n	$g(n) = 3n^2 + 5n + 1$	$C \times f(n) = 1000 \times n$
0	1	**0**
50	7751	**50000**
100	30501	**100000**
150	68251	**150000**
200	121001	**200000**
250	188751	**250000**
300	271501	**300000**
350	**369251**	350000
400	**482001**	400000
450	**609751**	450000
500	**752501**	500000

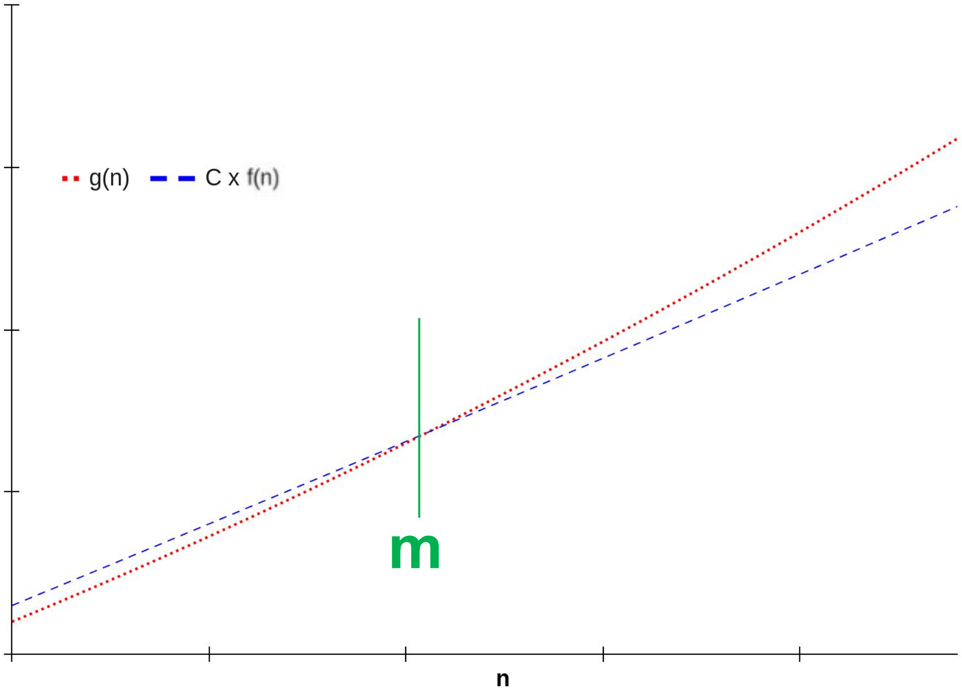
Definição da Notação Ω

- **g(n) é Ω(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para $n \geq m$, temos que $|g(n)| \geq c \times |f(n)|$



Definição da Notação Ω

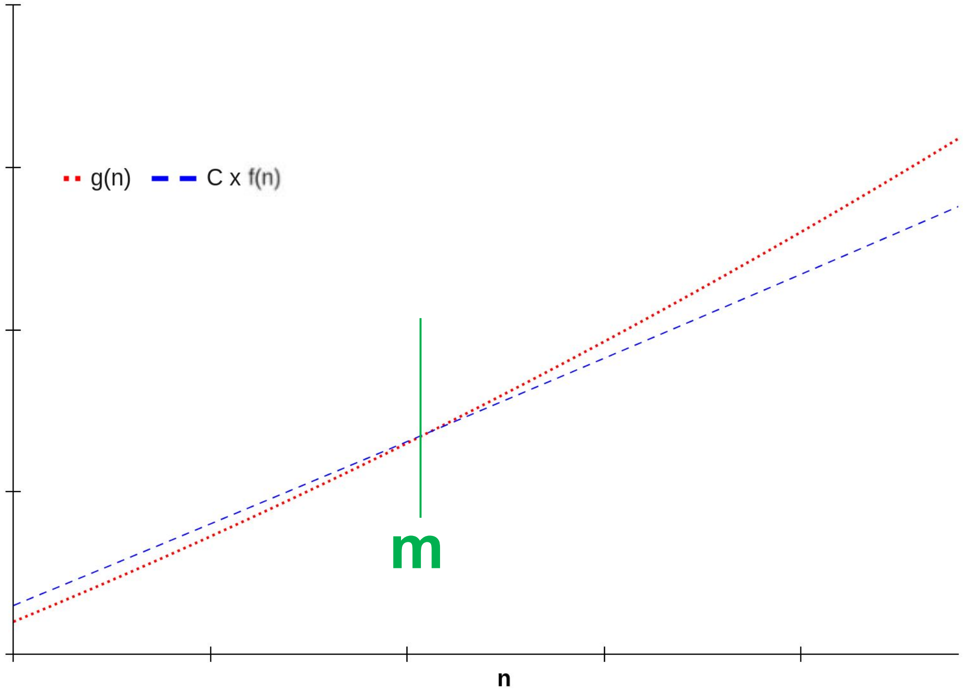
- **g(n) é Ω(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para n ≥ m, temos que |g(n)| ≥ c x |f(n)|



f(n) é um limite assintótico inferior para g(n)

Definição da Notação Ω

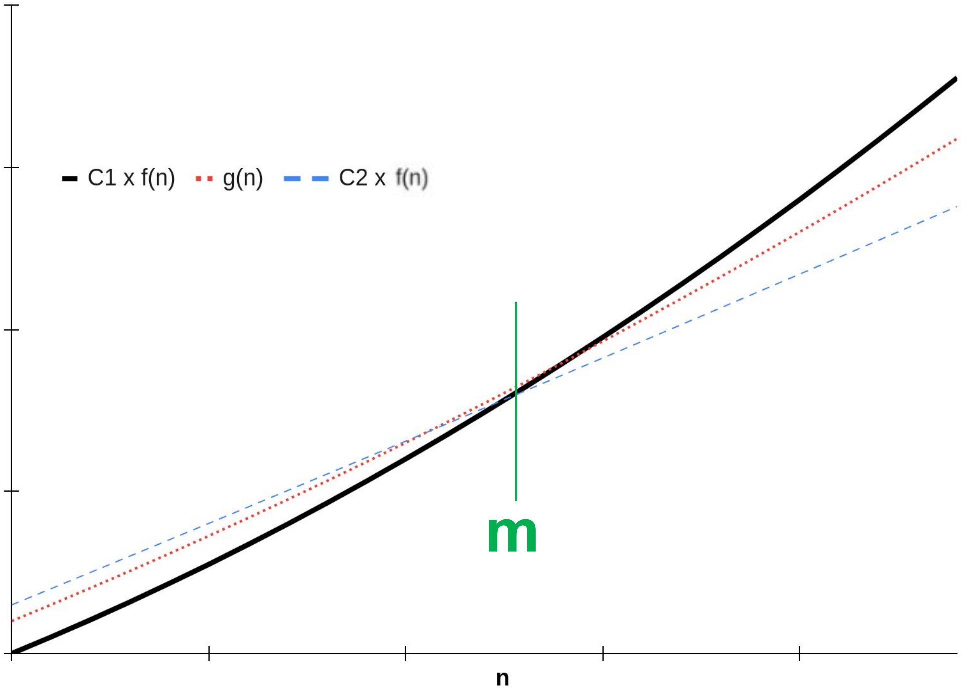
- **g(n) é Ω(f(n))**, se existirem as constantes positivas *c* e *m* tais que, para n ≥ m, temos que |g(n)| ≥ c x |f(n)|



Note que g(n) é Ω(f(n)) sss f(n) é O(g(n))

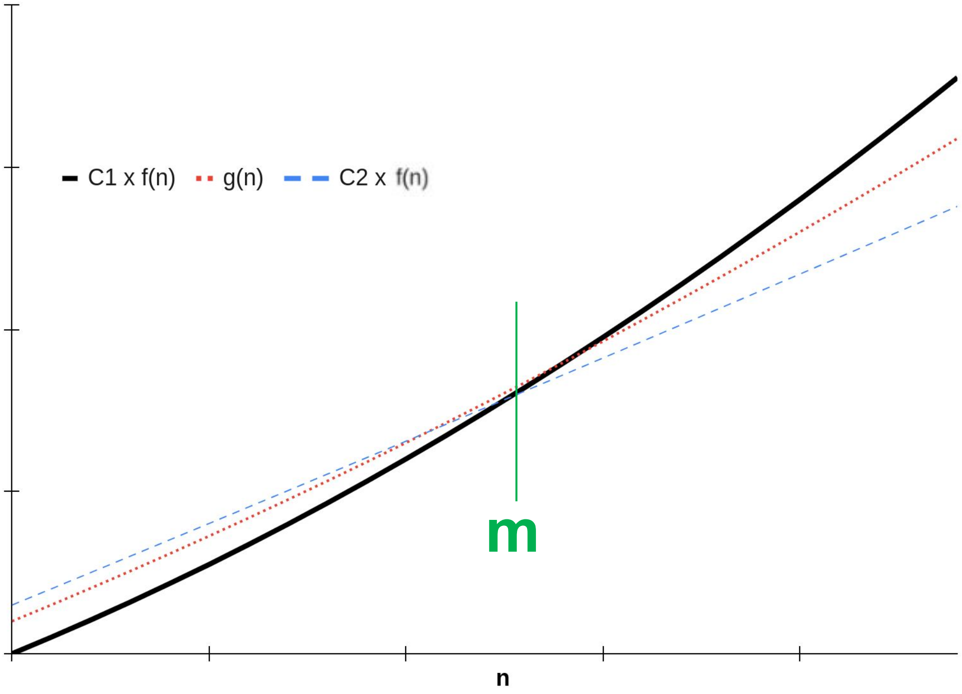
Definição da Notação Θ

- **g(n) é Θ(f(n))**, se existirem constantes positivas c_1, c_2 e m tais que, para $n \geq m$, temos que $c_1 \times |f(n)| \leq |g(n)| \leq c_2 \times |f(n)|$



Definição da Notação Θ

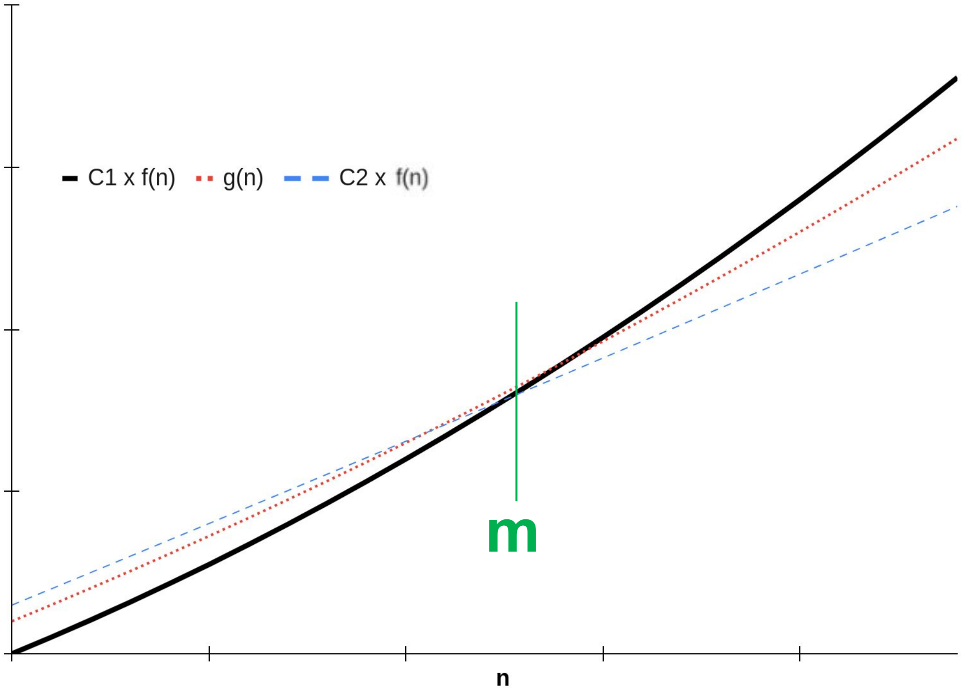
- **g(n) é Θ(f(n))**, se existirem constantes positivas c_1, c_2 e m tais que, para $n \geq m$, temos que $c_1 \times |f(n)| \leq |g(n)| \leq c_2 \times |f(n)|$



f(n) é um limite assintótico justo para g(n)

Definição da Notação Θ

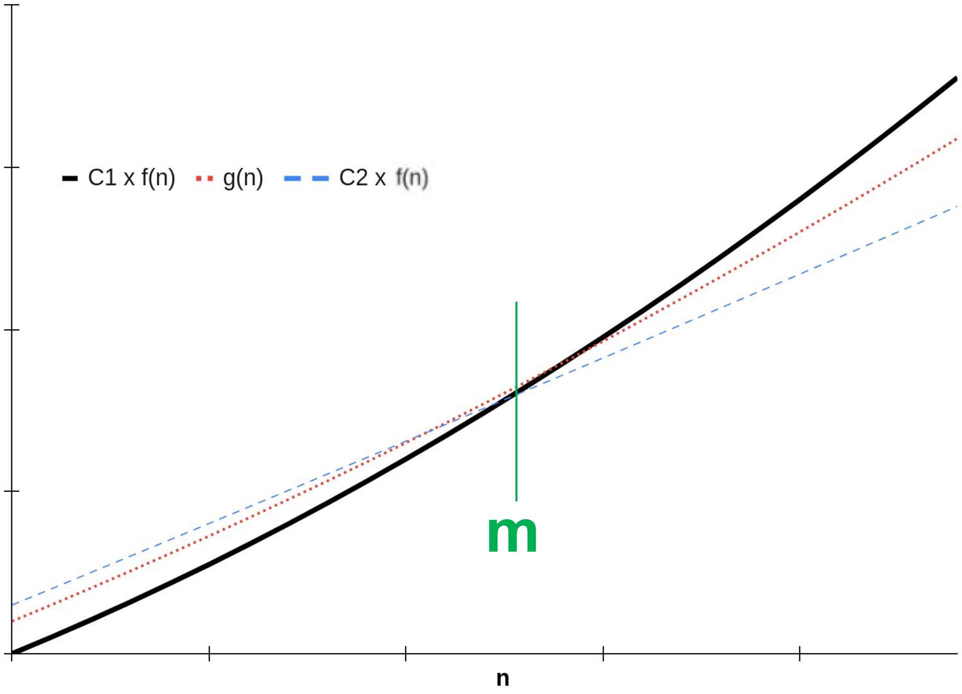
- **g(n) é Θ(f(n))**, se existirem constantes positivas c_1, c_2 e m tais que, para $n \geq m$, temos que $c_1 \times |f(n)| \leq |g(n)| \leq c_2 \times |f(n)|$



Se g(n) é O(f(n)) e Ω(f(n)), então, g(n) é Θ(f(n))

Definição da Notação Θ

- **g(n) é Θ(f(n))**, se existirem constantes positivas c_1, c_2 e m tais que, para $n \geq m$, temos que $c_1 \times |f(n)| \leq |g(n)| \leq c_2 \times |f(n)|$



g(n) é Θ(f(n))
sss
g(n) é O(f(n)) and Ω(f(n))

Classes de Algoritmos

- Constante: $\Theta(1)$
- Logarítmico: $\Theta(\lg n)$
- Linear: $\Theta(n)$
- Linear-logarítmico: $\Theta(n \times \lg n)$
- Quadrático: $\Theta(n^2)$
- Cúbico: $\Theta(n^3)$
- Exponencial: $\Theta(c^n)$
- Fatorial: $\Theta(n!)$

Algoritmos Polinomiais

- Um algoritmo é polinomial se é $\Theta(n^p)$ para algum inteiro p
- Problemas com algoritmos polinomiais são considerados tratáveis
- Problemas para os quais não há algoritmos polinomiais são considerados intratáveis
- Classes de problemas e o problema $P \stackrel{?}{=} NP$

Exercício Resolvido (13)

- Apresente a função e a ordem de complexidade para o **número de comparações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
    int max, min;
    if (array[0] > array[1]){
        max = array[0];
        min = array[1];
    } else {
        max = array[1];
        min = array[0];
    }
    for (int i = 2; i < n; i++){
        if (array[i] > max){
            max = array[i];
        } else if (array[i] < min){
            min = array[i];
}   }   }
```

Exercício Resolvido (13)

- Apresente a função e a ordem de complexidade para o **número de comparações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
    int max, min;
    if (array[0] > array[1]){
        max = array[0];
        min = array[1];
    } else {
        max = array[1];
        min = array[0];
    }
    for (int i = 2; i < n; i++){
        if (array[i] > max){
            max = array[i];
        } else if (array[i] < min){
            min = array[i];
}   }   }
```

Pause!

Exercício Resolvido (13)

- Apresente a função e a ordem de complexidade para o **número de comparações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
        int max, min;
        if (array[0] > array[1]){
                max = array[0];
                min = array[1];
        } else {
                max = array[1];
                min = array[0];
        }
        for (int i = 2; i < n; i++){
                if (array[i] > max){
                        max = array[i];
                } else if (array[i] < min){
                        min = array[i];
}       }       }
```

Função de Complexidade para Comparações	
Pior	$f(n) = 1 + 2(n - 2)$
Melhor	$f(n) = 1 + (n - 2)$

Ordem de Complexidade para Comparações	
Pior	$O(n)$, $\Omega(n)$ e $\Theta(n)$
Melhor	

Exercício Resolvido (14)

- Apresente a função e a ordem de complexidade para o **número de movimentações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
    int max, min;
    if (array[0] > array[1]){
        max = array[0];
        min = array[1];
    } else {
        max = array[1];
        min = array[0];
    }
    for (int i = 2; i < n; i++){
        if (array[i] > max){
            max = array[i];
        } else if (array[i] < min){
            min = array[i];
}   }   }
```

Exercício Resolvido (14)

- Apresente a função e a ordem de complexidade para o **número de movimentações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
        int max, min;
        if (array[0] > array[1]){
                max = array[0];
                min = array[1];
        } else {
                max = array[1];
                min = array[0];
        }
        for (int i = 2; i < n; i++){
                if (array[i] > max){
                        max = array[i];
                } else if (array[i] < min){
                        min = array[i];
}       }       }
```

Pause!

Exercício Resolvido (14)

- Apresente a função e a ordem de complexidade para o **número de movimentações de registros** no **pior** e **melhor** caso

```
void imprimirMaxMin(int [] array, int n){
          int max, min;
          if (array[0] > array[1]){
                    max = array[0];
                    min = array[1];
          } else {
                    max = array[1];
                    min = array[0];
          }
          for (int i = 2; i < n; i++){
                    if (array[i] > max){
                              max = array[i];
                    } else if (array[i] < min){
                              min = array[i];
}         }         }
```

Função de Complexidade para Movimentações	
Pior	$f(n) = 2 + (n - 2)$
Melhor	$f(n) = 2 + \cancel{(n - 2) \times 0}$

Ordem de Complexidade para Movimentações	
Pior	$O(n)$, $\Omega(n)$ e $\Theta(n)$
Melhor	$O(1)$, $\Omega(1)$ e $\Theta(1)$

Exercício Resolvido (15)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
i = 0;
while (i < n) {
        i++;        a--;
}
if (b > c) {
        i--;
} else {
        i--;
        a--;
}
```

Exercício Resolvido (15)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
i = 0;
while (i < n) {
        i++;        a--;
}
if (b > c) {
        i--;
} else {
        i--;
        a--;
}
```

Pause!

Exercício Resolvido (15)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
i = 0;
while (i < n) {
        i++;        a--;
}
if (b > c) {
        i--;
} else {
        i--;
        a--;
}
```

Função de Complexidade	
Pior	$f(n) = n + 2$
Melhor	$f(n) = n + 1$

Ordem de Complexidade	
Pior	$O(n)$, $\Omega(n)$ e $\Theta(n)$
Melhor	

Exercício Resolvido (16)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a--;
        b--;
    }
    c--;
}
```

Exercício Resolvido (16)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a--;
        b--;
    }
    c--;
}
```

Pause!

Exercício Resolvido (16)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
        a--;
        b--;
    }
    c--;
}
```

Função de Complexidade	
Todos os casos	$f(n) = (2n + 1)n$

Ordem de Complexidade	
Todos os casos	$O(n^2)$, $\Omega(n^2)$ e $\Theta(n^2)$

✓

Exercício Resolvido (17)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 1; j <= n; j *= 2) {
        b--;
    }
}
```

Exercício Resolvido (17)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 1; j <= n; j *= 2) {
        b--;
    }
}
```

Pause!

Exercício Resolvido (17)

- Apresente a função e a ordem de complexidade para o **número de subtrações** para o **pior** e **melhor** caso

```
for (i = 0; i < n; i++) {
    for (j = 1; j <= n; j *= 2) {
        b--;
    }
}
```

Função de Complexidade	
Todos os casos	$f(n) = (\lfloor lg(n) \rfloor + 1) \times n = n \times \lfloor lg(n) \rfloor + n$

Ordem de Complexidade	
Todos os casos	$O(n \times lg(n))$, $\Omega(n \times lg(n))$ e $\Theta(n \times lg(n))$

Exercício Resolvido (18)

- Apresente o tipo de crescimento que melhor caracteriza as funções abaixo (Khan Academy, adaptado)

	Constante	Linear	Polinomial	Exponencial
$3n$				
1				
$(3/2)n$				
$2n^3$				
2^n				
$3n^2$				
1000				
$(3/2)^n$				

Exercício Resolvido (18)

- Apresente o tipo de crescimento que melhor caracteriza as funções abaixo (Khan Academy, adaptado)

	Constante	Linear	Polinomial	Exponencial
$3n$				
1				
$(3/2)n$				
$2n^3$				
2^n				
$3n^2$				
1000				
$(3/2)^n$				

Pause!

Exercício Resolvido (18)

- Apresente o tipo de crescimento que melhor caracteriza as funções abaixo (Khan Academy, adaptado) ✓

	Constante	Linear	Polinomial	Exponencial
$3n$		✓		
1	✓			
$(3/2)n$		✓		
$2n^3$			✓	
2^n				✓
$3n^2$			✓	
1000	✓			
$(3/2)^n$				✓

Exercício Resolvido (19)

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

Exercício Resolvido (19)

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

Pause!

Exercício Resolvido (19)

- Classifique as funções $f_1(n) = n^2$, $f_2(n) = n$, $f_3(n) = 2^n$, $f_4(n) = (3/2)^n$, $f_5(n) = n^3$ e $f_6(n) = 1$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

RESPOSTA:

$f_6(n) = 1 \;<\; f_2(n) = n \;<\; f_1(n) = n^2 \;<\; f_5(n) = n^3 \;<\; f_4(n) = (3/2)^n \;<\; f_3(n) = 2^n$

Exercício Resolvido (20)

- Classifique as funções $f_1(n) = n.\log_6(n)$, $f_2(n) = \lg(n)$, $f_3(n) = \log_8(n)$, $f_4(n) = 8n^2$, $f_5(n) = n.\lg(n)$, $f_6(n) = 64$, $f_7(n) = 6n^3$, $f_8(n) = 8^{2n}$ e $f_9(n) = 4n$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

Exercício Resolvido (20)

- Classifique as funções $f_1(n) = n.\log_6(n)$, $f_2(n) = \lg(n)$, $f_3(n) = \log_8(n)$, $f_4(n) = 8n^2$, $f_5(n) = n.\lg(n)$, $f_6(n) = 64$, $f_7(n) = 6n^3$, $f_8(n) = 8^{2n}$ e $f_9(n) = 4n$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado)

Pause!

Exercício Resolvido (20)

- Classifique as funções $f_1(n) = n.\log_6(n)$, $f_2(n) = \lg(n)$, $f_3(n) = \log_8(n)$, $f_4(n) = 8n^2$, $f_5(n) = n.\lg(n)$, $f_6(n) = 64$, $f_7(n) = 6n^3$, $f_8(n) = 8^{2n}$ e $f_9(n) = 4n$ de acordo com o crescimento, do mais lento para o mais rápido (Khan Academy, adaptado) ✓

RESPOSTA:

$$f_6(n) = 64 \;<\; f_3(n) = \log_8(n) \;<\; f_2(n) = \lg(n) \;<\; f_9(n) = 4n \;<\; f_1(n) = n.\log_6(n) \;<$$
$$f_5(n) = n.\lg(n) \;<\; f_4(n) = 8n^2 \;<\; f_7(n) = 6n^3 \;<\; f_8(n) = 8^{2n}$$

Exercício Resolvido (21)

- Faça a correspondência entre cada função f(n) com sua g(n) equivalente, em termos de O. Essa correspondência acontece quando f(n) = O(g(n)) (Khan Academy, adaptado)

f(n)	**g(n)**
$n + 30$	n^4
$n^2 + 2n - 10$	$3n - 1$
$n^3 \times 3n$	$lg(2n)$
$lg(n)$	$n^2 + 3n$

Exercício Resolvido (21)

- Faça a correspondência entre cada função f(n) com sua g(n) equivalente, em termos de O. Essa correspondência acontece quando f(n) = O(g(n)) (Khan Academy, adaptado)

Pause!

f(n)	g(n)
$n + 30$	n^4
$n^2 + 2n - 10$	$3n - 1$
$n^3 \times 3n$	$lg(2n)$
$lg(n)$	$n^2 + 3n$

Exercício Resolvido (21)

- Faça a correspondência entre cada função f(n) com sua g(n) equivalente, em termos de O. Essa correspondência acontece quando f(n) = O(g(n)) (Khan Academy, adaptado)

f(n)	**g(n)**
$n + 30$	n^4
$n^2 + 2n - 10$	$3n - 1$
$n^3 \ x \ 3n$	$lg(2n)$
$lg(n)$	$n^2 + 3n$

Exercícios

Exercício (1)

- Encontre o maior e menor valores em um *array* de inteiros e, em seguida, encontre a função de complexidade de tempo para sua solução

Exercício (2)

- Considerando o problema de encontrar o maior e menor valores em um *array*, veja os quatro códigos propostos e analisados no livro do Ziviani

Exercício (3)

- Preencha verdadeiro ou falso na tabela abaixo:

	$\Theta(1)$	$\Theta(\lg n)$	$\Theta(n)$	$\Theta(n.\lg(n))$	$\Theta(n^2)$	$\Theta(n^3)$	$\Theta(n^5)$	$\Theta(n^{20})$
$f(n) = \lg(n)$								
$f(n) = n \,.\, \lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

Exercício (4)

- Preencha verdadeiro ou falso na tabela abaixo:

	O(1)	O(lg n)	O(n)	O(n.lg(n))	$O(n^2)$	$O(n^3)$	$O(n^5)$	$O(n^{20})$
$f(n) = lg(n)$								
$f(n) = n . lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

Exercício (5)

- Preencha verdadeiro ou falso na tabela abaixo:

	$\Omega(1)$	$\Omega(\lg n)$	$\Omega(n)$	$\Omega(n.\lg(n))$	$\Omega(n^2)$	$\Omega(n^3)$	$\Omega(n^5)$	$\Omega(n^{20})$
$f(n) = \lg(n)$								
$f(n) = n \,.\, \lg(n)$								
$f(n) = 5n + 1$								
$f(n) = 7n^5 - 3n^2$								
$f(n) = 99n^3 - 1000n^2$								
$f(n) = n^5 - 99999n^4$								

Exercício (6)

- Dada a definição da notação Ω:

a) Mostre os valores de c e m tal que, para $n \geq m$, $|g(n)| \geq c \times |f(n)|$, provando que $3n^2 + 5n + 1$ é $\Omega(n^2)$

b) Mostre os valores de c e m tal que, para $n \geq m$, $|g(n)| \geq c \times |f(n)|$, provando que $3n^2 + 5n + 1$ é $\Omega(n)$

c) Prove que $3n^2 + 5n + 1$ **não é** $\Omega(n^3)$

Exercício (7)

- Dada a definição da notação Θ:

a) Mostre um valor para c_1, c_2 e m tal que, para $n \geq m$, $c_1 \times |f(n)| \leq |g(n)| \leq c_2 \times |f(n)|$, provando que $3n^2 + 5n + 1$ é $\Theta(n^2)$

b) Prove que $3n^2 + 5n + 1$ **não** é $\Theta(n)$

c) Prove que $3n^2 + 5n + 1$ **não** é $\Theta(n^3)$

Exercício (8)

- Suponha um sistema de monitoramento contendo os métodos telefone, luz, alarme, sensor e câmera, apresente a função e ordem de complexidade para o **pior** e **melhor** caso: (a) **método alarme**; (b) **outros métodos**.

```
void sistemaMonitoramento() {
        alarme(((telefone() == true && luz() == true)) ? 0 : 1);
        for (int i = 2; i < n; i++){
                if (sensor(i- 2) == true){
                        alarme (i - 2);
                } else if (camera(i- 2) == true){
                        alarme (i - 2 + n);
}  }  }
```

Exercício (9)

- Apresente um código, defina duas operações relevantes e apresente a função e a ordem de complexidade para as operações escolhidas no pior e melhor caso

Exercício (10)

- Anteriormente, verificamos que quando desejamos pesquisar a existência de ***um*** elemento em um *array* de números reais é adequado executar uma pesquisa sequencial cujo custo é $\Theta(n)$. Nesse caso, o custo de ordenar o *array* e, em seguida, aplicar uma pesquisa binária é mais elevado, $\Theta(n \times \lg(n)) + \Theta(\lg(n)) = \Theta(n \times \lg(n))$. Agora, supondo que desejamos efetuar ***n*** pesquisas, responda qual das duas soluções é mais eficiente