

# Complexidade Assintótica

**Professores:**

**Norton T. Roman**

**Fátima L. S. Nunes**

# Vamos lembrar dos algoritmos de ordenação

- Quais vocês já conhecem?

# Vamos lembrar dos algoritmos de ordenação

- Quais vocês já conhecem?
  - Seleção (Selection Sort)
  - Inserção (Insertion Sort)
  - Bolha (Bubble Sort)

# Vamos lembrar dos algoritmos de ordenação

- Quais vocês já conhecem?
  - Seleção (Selection Sort)
  - Inserção (Insertion Sort)
  - Bolha (Bubble Sort)

Qual é o melhor ???

# Vamos lembrar dos algoritmos de ordenação

- Quais vocês já conhecem?
  - Seleção (Selection Sort)
  - Inserção (Insertion Sort)
  - Bolha (Bubble Sort)

Qual é o melhor ???

# Vamos lembrar dos algoritmos de ordenação

## Inserção

```
void insercaoDireta(int [] numeros)
{
    for (int ivet=1; ivet < numeros.length; ivet++)
    {
        int numAInserir = numeros[ivet];
        int isubv = ivet;

        while ((isubv > 0) &&
            (numeros [isubv -1] > numoAInserir))
        {
            numeros[isubv] = numeros[isubv - 1];
            isubv--;
        }
        numeros[isubv] = numAInserir;
    }
}
```

## Bolha

```
void bolha(int [] numeros)
{
    for (ivet = numeros.length - 1; ivet > 0; ivet--)
    {
        for (isubv = 0; isubv < ivet; isubv++)
        {
            if (numeros[isubv ] > numeros[isubv+1])
            {
                temp = numeros[isubv];
                numeros [isubv] = numeros [isubv+1];
                numeros [isubv+1] = temp;
            }
        }
    }
}
```

# Vamos lembrar dos algoritmos de ordenação

Melhor em quê?

- tempo
- memória
- dificuldade

# Recordando...

- **Algoritmo**

O que é?

# Recordando...

## ■ Algoritmo

Informalmente (Cormen *et al.*, 2002):

- Qualquer procedimento computacional bem definido que toma algum valor ou conjunto de valores como **entrada** e produz algum valor ou conjunto de valores com **saída**.
- Sequência de passos computacionais que transformam a entrada na saída.

# Análise de Algoritmos

- **O que é analisar um algoritmo?**

# Análise de Algoritmos

- **O que é analisar um algoritmo?**
  - Prever os recursos de que o algoritmo necessitará.
  - Quais recursos?

# Análise de Algoritmos

- **O que é analisar um algoritmo?**
  - Prever os recursos de que o algoritmo necessitará.
  - Quais recursos?
    - memória, largura de banda de comunicação, hardware
    - principal: **tempo de computação**
  - **Análise de algoritmos:**
    - permite escolher o algoritmo **mais eficiente** dentre um conjunto de candidatos para resolver um problema

# Análise de Algoritmos

- Em geral, tempo de duração de um algoritmo cresce com o tamanho da entrada
  - É usual descrever o **tempo de execução** de um programa como uma função do **tamanho de sua entrada**.

# Análise de Algoritmos

- Em geral, tempo de duração de um algoritmo cresce com o tamanho da entrada
  - Tamanho de entrada ( $n$ ):
    - depende do problema estudado
    - maioria dos problemas: número de itens de entrada
    - exemplo: ordenação (quantidade de elementos do arranjo)
  - Tempo de execução:
    - quantidade de operações primitivas ou etapas executadas para uma determinada entrada
    - vamos considerar que cada linha  $i$  leva um tempo constante  $c_i$

# Análise de Algoritmos

## ➤ Função de custo de um algoritmo

- representa o custo de tempo de cada instrução e o número de vezes que cada instrução é executada

Exemplo: *insertion-sort*(A) (entrada: array A que tem tamanho n)

	custo	vezes
1 para j = 2 até tamanho[A] faça	$C_1$	n
2 chave = A[j]	$C_2$	n-1
3 // ordenando elementos à esquerda	0	n-1
4 i = j - 1	$C_4$	n-1
5 enquanto i > 0 e A[i] > chave faça	$C_5$	$\sum_{j=2}^n t_j$
6 A[i+1] = A[i]	$C_6$	$\sum_{j=2}^n (t_j - 1)$
7 i = i - 1	$C_7$	$\sum_{j=2}^n (t_j - 1)$
8 fim enquanto		
9 A[i+1] = chave		
10 fim para	$C_8$	n-1

# Análise de Algoritmos

➤ Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

	custo	vezes
1 para j = 2 até tamanho[A] faça	$c_1$	n
2 chave = A[j]	$c_2$	n-1
3 // ordenando elementos à esquerda	0	n-1
4 i = j - 1	$c_4$	n-1
5 enquanto i > 0 e A[i] > chave faça	$c_5$	$\sum_{j=2}^n t_j$
6 A[i+1] = A[i]	$c_6$	$\sum_{j=2}^n (t_j - 1)$
7 i = i - 1	$c_7$	$\sum_{j=2}^n (t_j - 1)$
8 fim enquanto		
9 A[i+1] = chave	$c_8$	n-1
10 fim para		

# Análise de Algoritmos

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)$$

- Melhor caso:** vetor já ordenado ( $A[i] \leq \text{chave}$  na linha 5  $\rightarrow t_j = 1$  para  $j = 2, 3, \dots, n$ )

- $T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) =$   
 $(c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$

- Tempo de execução, neste caso, pode ser expresso como  $an + b$  para constantes **a** e **b** que dependem dos custos de instrução  $c_i$   $\rightarrow$  **função linear** de  $n$

# Análise de Algoritmos

Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_{j-1}) + c_7 \sum_{j=2}^n (t_{j-1}) + c_8(n-1)$$

**Pior caso:** vetor em ordem inversa (deve comparar cada elemento  $A[j]$  com cada elemento do subarranjo ordenado  $A[j \dots j-1]$   $\rightarrow t_j = j$  para  $j=2,3,\dots,n$ )

$$\sum_{j=2}^n (j) = \frac{n(n-1)}{2} + 1$$

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \frac{n(n-1)}{2} + 1 + c_6 \frac{n(n-1)}{2} + c_7 \frac{n(n-1)}{2} + c_8(n-1) =$$

$$\left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - (c_2 + c_4 + c_5 + c_8)$$

Tempo de execução, neste caso, pode ser expresso como  $an^2 + bn + c$  para constantes  $a$ ,  $b$  e  $c$  que dependem dos custos de instrução  $c_i$   $\rightarrow$  **função quadrática** de  $n$

# Análise de Algoritmos

## ➤ Em geral:

- tempo de execução de um algoritmo é fixo para uma determinada entrada
- analisamos apenas o **pior caso** dos algoritmos:
  - é um limite superior sobre o tempo de execução de qualquer entrada;
  - pior caso ocorre com muita frequência para alguns algoritmos. Exemplo: registro inexistente em um banco de dados;
  - muitas vezes, o **caso médio** é quase tão ruim quanto o pior caso

# Análise de Algoritmos

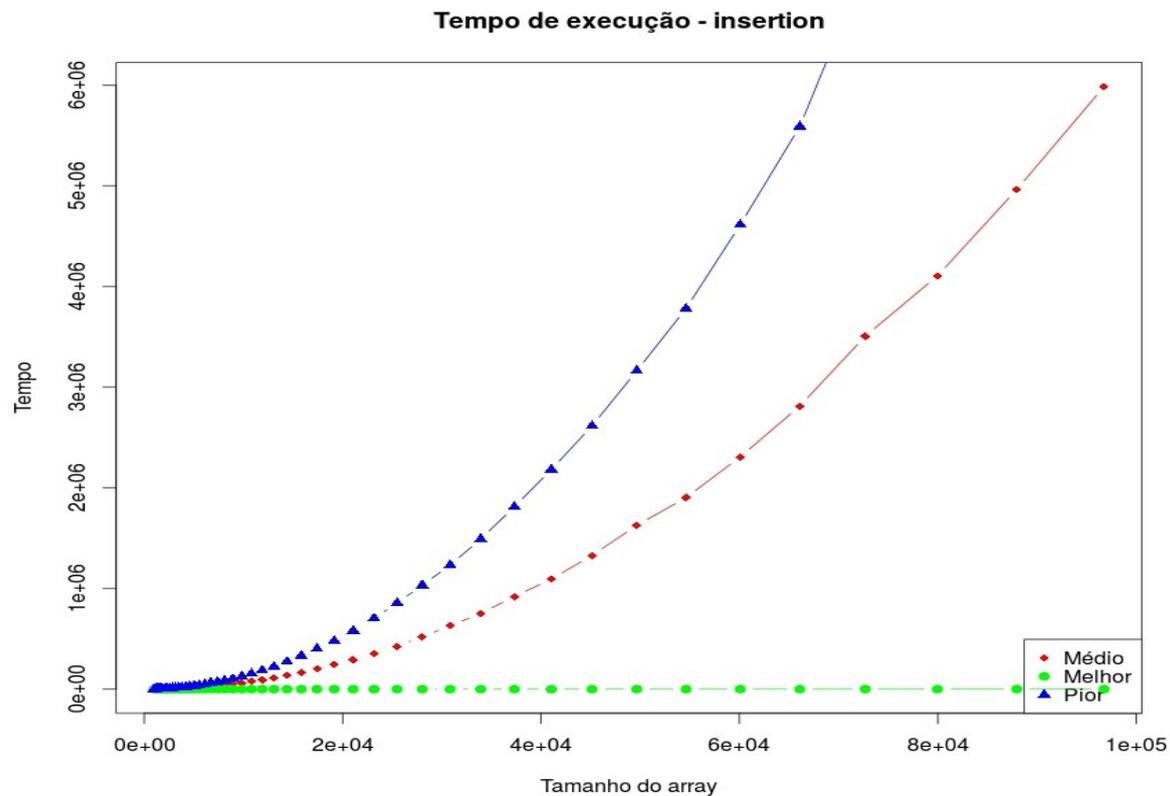
- Nas análises anteriores, foram feitas algumas simplificações em relação às constantes, chegando à função linear e à função quadrática
- **Taxa de crescimento** ou **ordem de crescimento**:
  - considera apenas o termo inicial de uma fórmula (exemplo:  $an^2$ ), pois os termos de mais baixa ordem são relativamente insignificantes para grandes valores de  $n$ ;
  - ignora o coeficiente constante do termo inicial também por ser menos significativo para grandes entradas;
  - Portanto, dizemos que: a ordenação por inserção, por exemplo, tem um tempo de execução do pior caso igual a  $\Theta(n^2)$  (*lê-se “theta de n ao quadrado”*);
  - Em geral, consideramos um algoritmo mais eficiente que outro se o tempo de execução do seu pior caso apresenta uma ordem de crescimento mais baixa.

# Exercício

- Criar o gráfico do insertion sort, medindo o tempo médio, melhor e pior caso.

# Exercício

- Criar o gráfico do insertion sort, medindo o tempo médio, melhor e pior caso.



# Complexidade? Assintótica?

- **Complexidade**

(cs) *sf* (*complexo+dade*) Qualidade do que é complexo.

- **Complexo**

(cs) *adj* (*lat complexu*) **1** Que abrange ou encerra muitos elementos ou partes. **2** Que pode ser considerado sob vários pontos de vista. **3** Complicado.

# Complexidade? Assintótica?

## ▪ Assintótico

*adj (assíntota+ico<sup>2</sup>) Geom* **1** Pertencente ou relativo à assíntota. **2** Qualificativo do espaço compreendido entre uma curva e a sua assíntota. **3** Diz-se da direção paralela de uma assíntota. *Var: assíptótico.*

## ▪ Assíntota

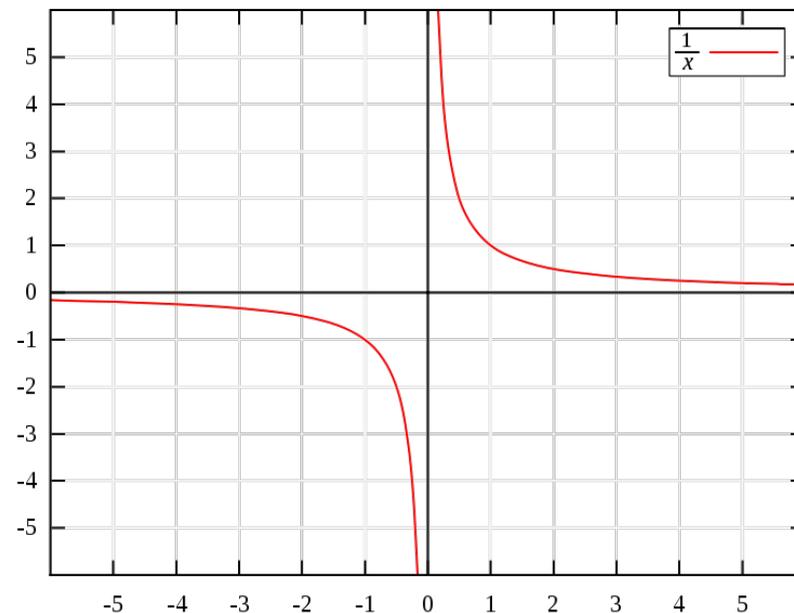
*sf (gr asýmptotos) Geom* Linha reta que se aproxima indefinidamente de uma curva sem nunca poder tocá-la. *Var: assíptota.*

# Complexidade? Assintótica?

## ■ Assíntota

*sf (gr asýmptotos) Geom* Linha reta que se aproxima indefinidamente de uma curva sem nunca poder tocá-la. *Var: assímptota.*

**A função  $f(x)=1/x$  tem como assíntotas os eixos coordenados.**



(Fonte: <http://pt.wikipedia.org/wiki/Assímptota>)

# Complexidade assintótica

- Em ciência da computação e matemática aplicada, particularmente a análise de algoritmos, análise real, e engenharia, **análise assintótica** é um método de descrever o comportamento de limites. Exemplos incluem o desempenho de algoritmos quando aplicados a um volume muito grande de dados de entrada, ou o comportamento de sistemas físicos quando eles são muito grandes.

# Crescimento Assintótico de Funções

- Escolha do algoritmo não é um problema crítico quando  $n$  é pequeno.
  - O problema é quando  $n$  cresce.
- Por isso, é usual analisar o comportamento das funções de custo quando  $n$  é bastante grande:
  - analisa-se o comportamento assintótico das funções de custo;
  - representa o *limite do comportamento da função de custo quando  $n$  cresce*.

# Crescimento Assintótico de Funções

- Eficiência assintótica dos algoritmos:
  - estuda a maneira como o **tempo de execução** de um algoritmo **aumenta** com o tamanho da entrada no limite, **à medida que o tamanho da entrada aumenta indefinidamente** (sem limitação)
  - em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para toda as entradas, exceto as pequenas.

# Vamos ordenar

Quais funções “crescem mais”?

$$f(n) = n$$

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

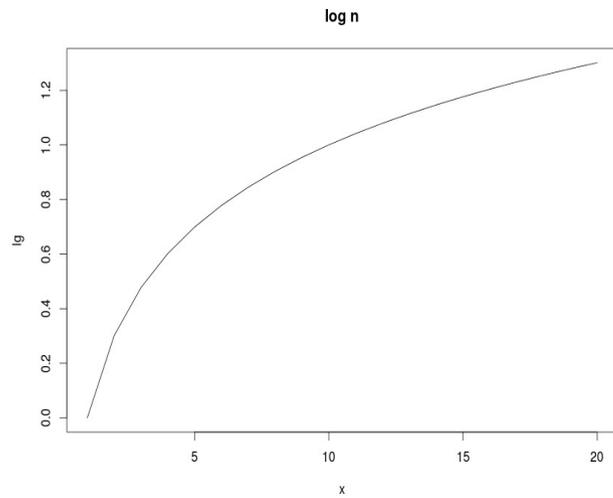
$$f(n) = \log n$$

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

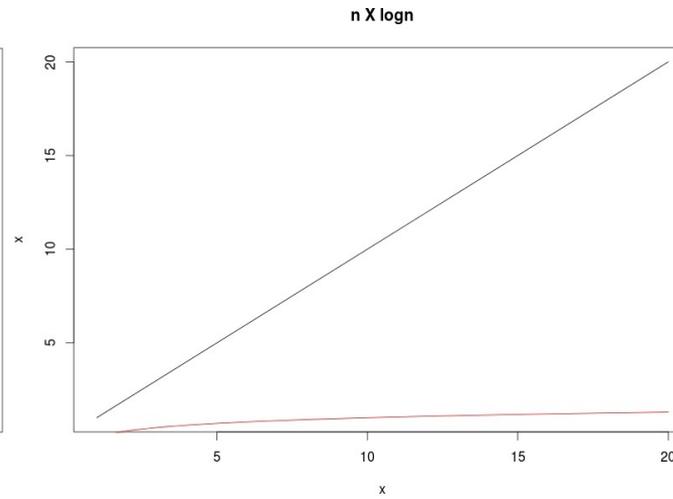
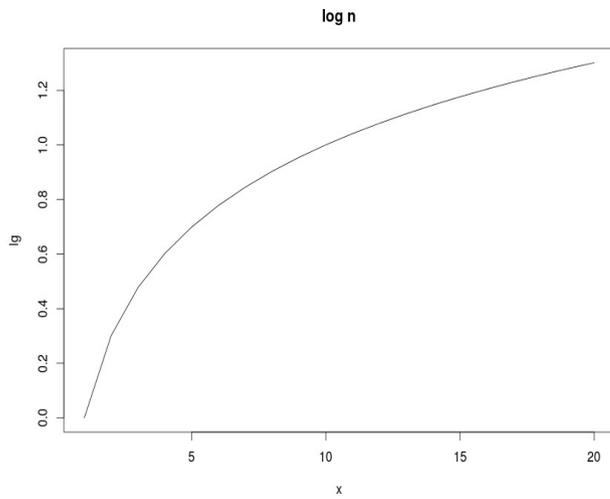
$$f(n) = 100n^2 + 15n$$

$$f(n) = n \log n$$

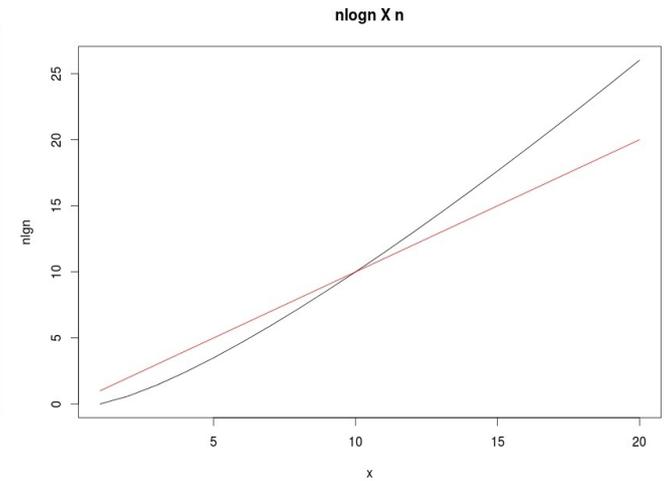
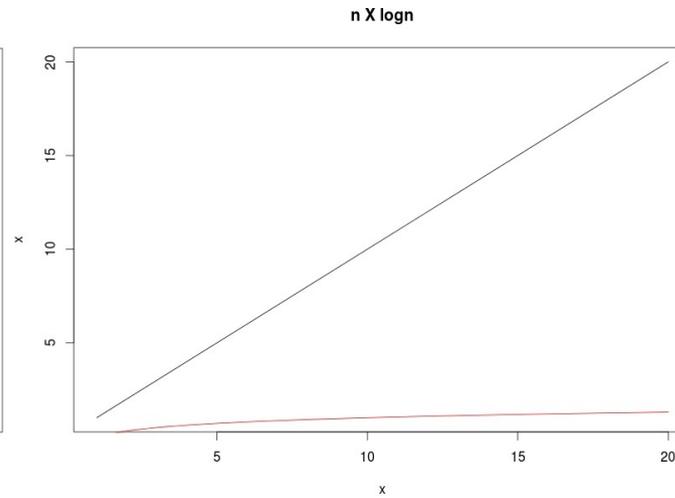
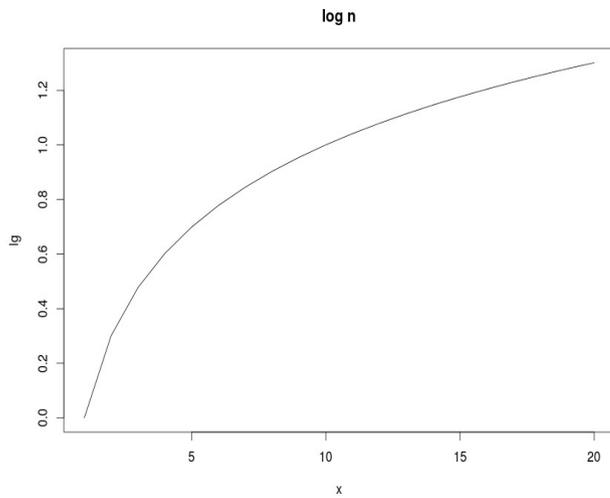
# Vamos ordenar



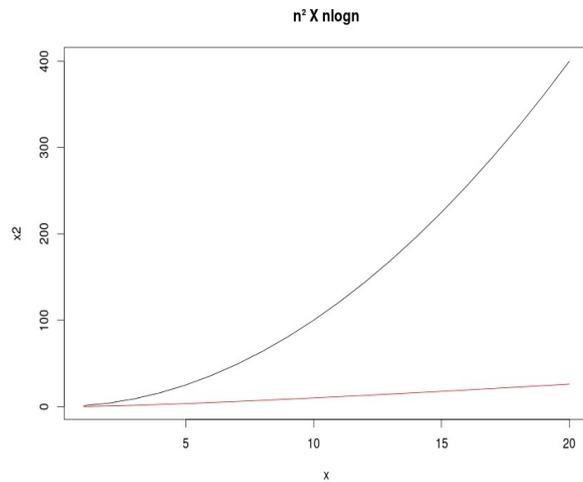
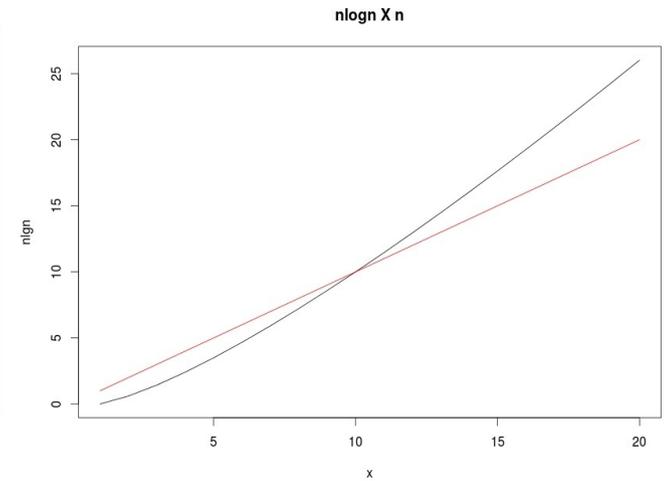
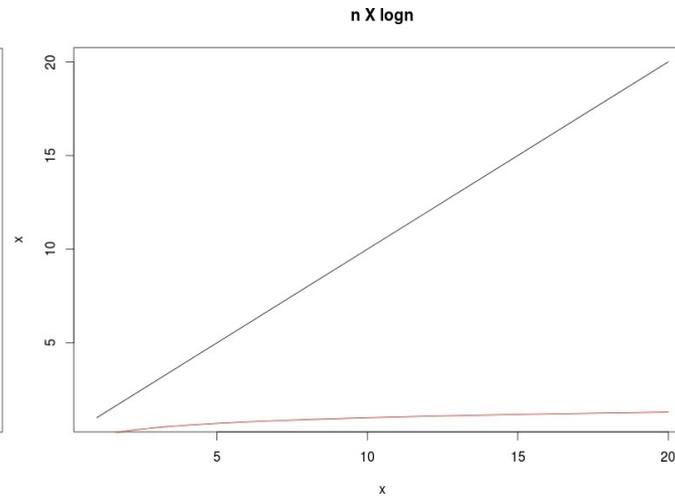
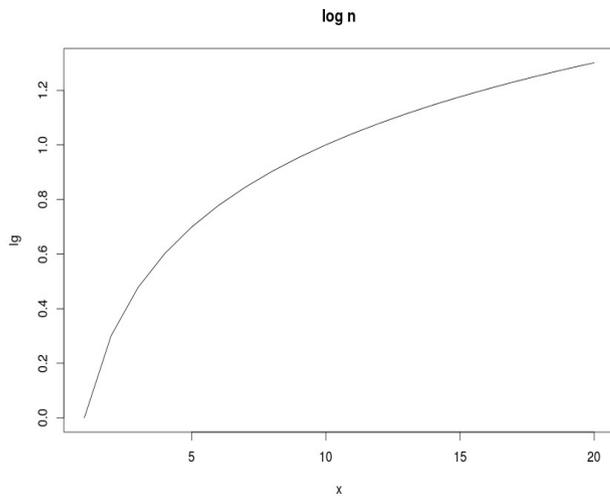
# Vamos ordenar



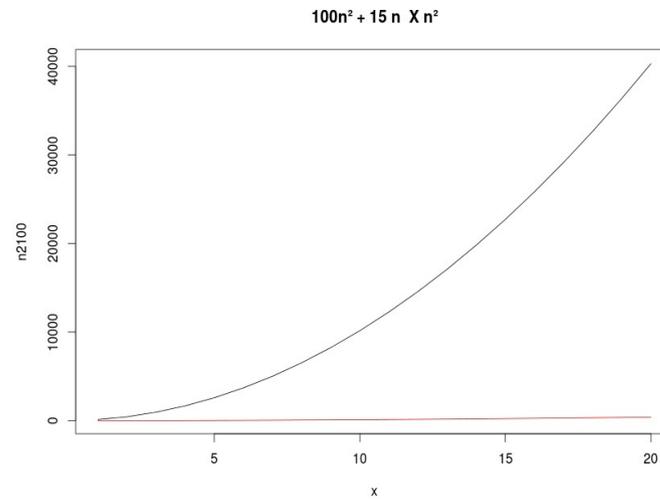
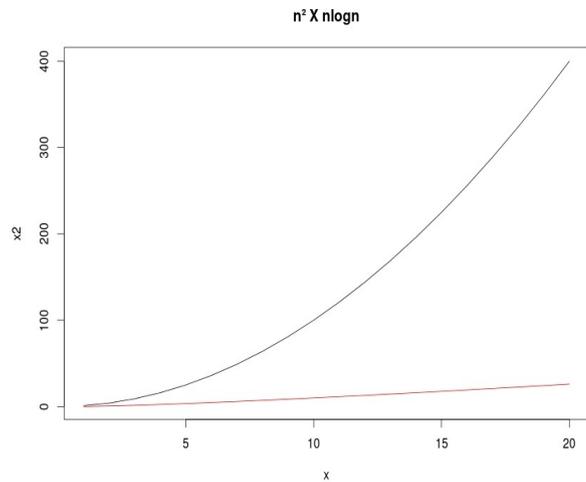
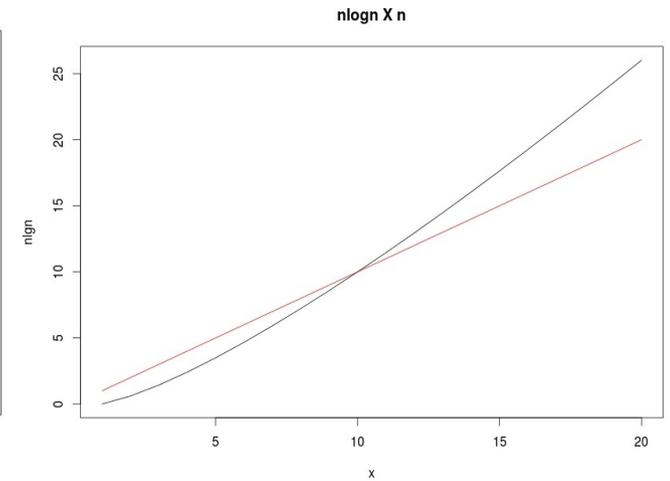
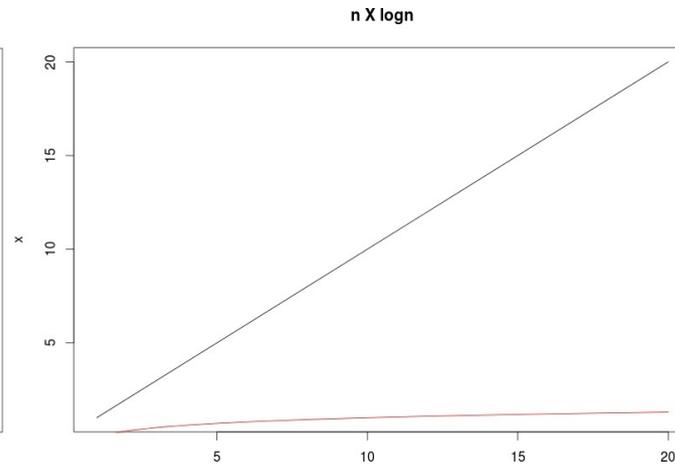
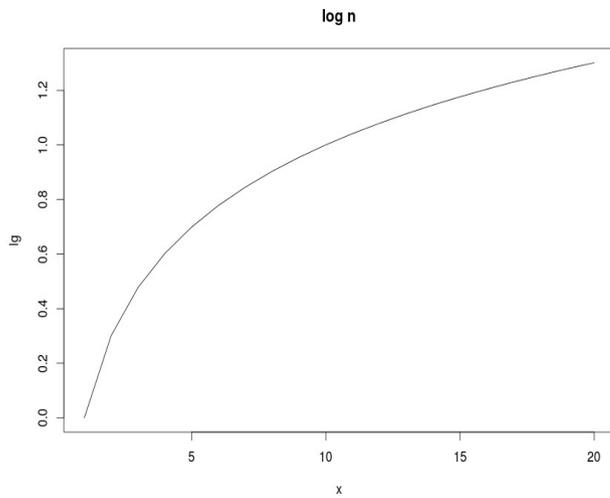
# Vamos ordenar



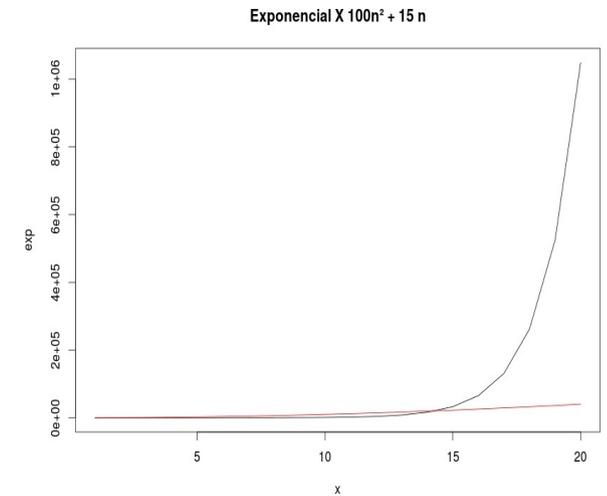
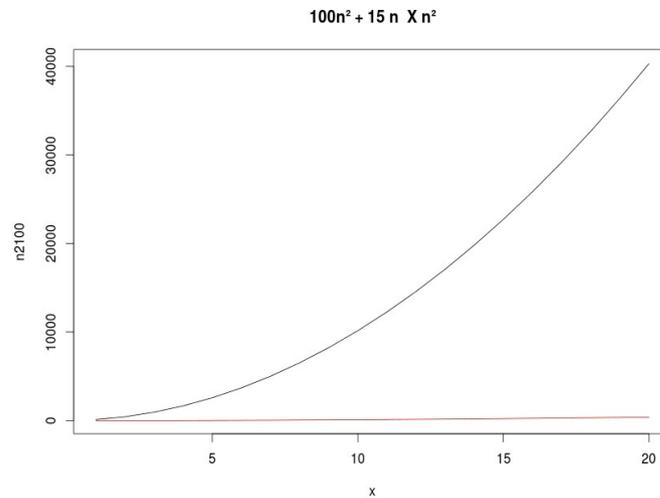
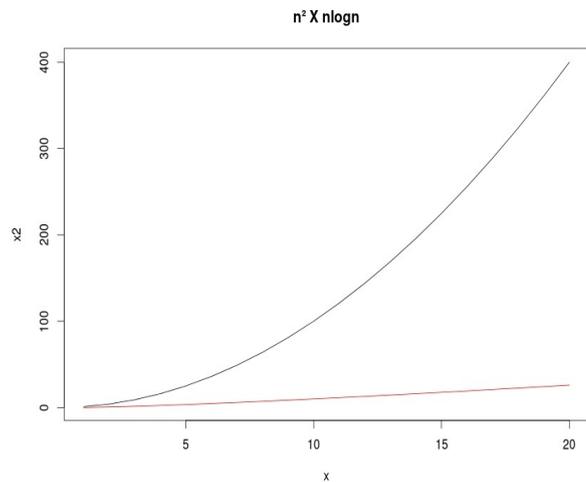
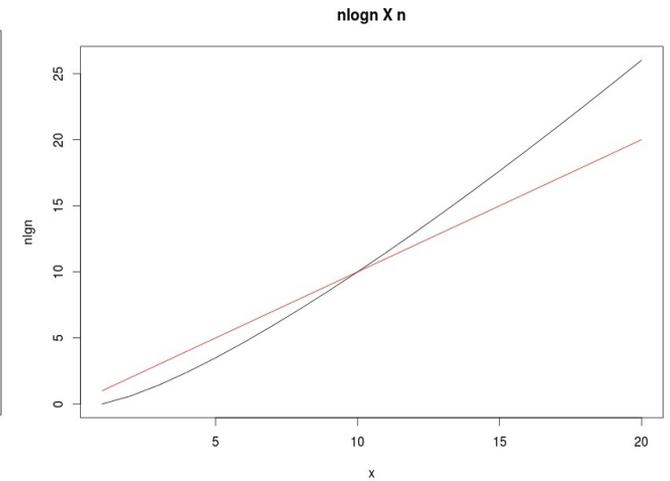
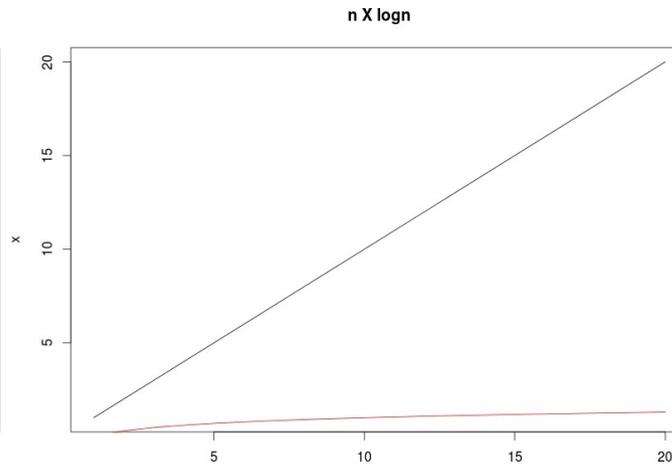
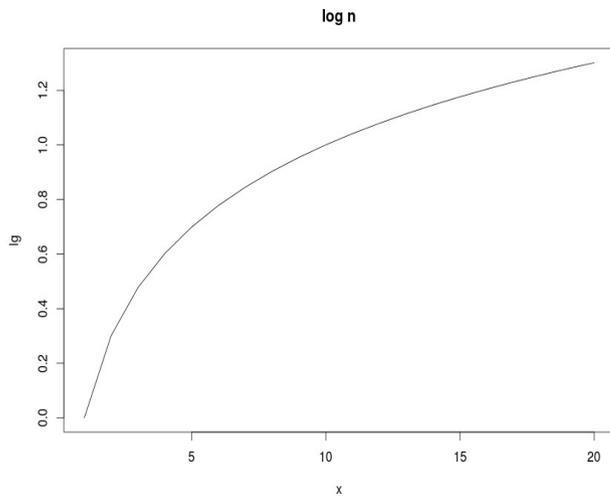
# Vamos ordenar



# Vamos ordenar



# Vamos ordenar



# Comportamento Assintótico

- Vamos comparar funções assintoticamente, ou seja, para valores grandes, desprezando constantes multiplicativas e termos de menor ordem.

	$n = 100$	$n = 1000$	$n = 10^4$	$n = 10^6$	$n = 10^9$
$\log n$	2	3	4	6	9
$n$	100	1000	$10^4$	$10^6$	$10^9$
$n \log n$	200	3000	$4 \cdot 10^4$	$6 \cdot 10^6$	$9 \cdot 10^9$
$n^2$	$10^4$	$10^6$	$10^8$	$10^{12}$	$10^{18}$
$100n^2 + 15n$	$1,0015 \cdot 10^6$	$1,00015 \cdot 10^8$	$\approx 10^{10}$	$\approx 10^{14}$	$\approx 10^{20}$
$2^n$	$\approx 1,26 \cdot 10^{30}$	$\approx 1,07 \cdot 10^{301}$	?	?	?

# Comportamento Assintótico

- Supondo uma máquina que execute 1 milhão ( $10^6$ ) de operações por segundo

n

Função de custo	10	20	30	40	50	60
$n$	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s	0,00006s
$n^2$	0,0001s	0,0004s	0,0009s	0,0016s	0,0025s	0,0036s
$n^3$	0,001s	0,008s	0,027s	0,064s	0,125s	0,216s
$n^5$	0,1s	3,2s	24,3s	1,7min	5,2min	12,96min
$2^n$	0,001s	1,04s	17,9min	12,7dias	35,7 anos	366 séc.
$3^n$	0,059s	58min	6,5anos	3855séc.	$10^8$ séc.	$10^{13}$ séc.

# Comportamento Assintótico

- Influência do aumento de velocidade dos computadores no tamanho do problema, considerando a complexidade assintótica
  - Exemplo: um aumento de 1000 vezes na velocidade do computador resolve, considerando o mesmo tempo, um problema dez vezes maior de complexidade  $\Theta(n^3)$  e um problema 1000 vezes maior se a complexidade for  $\Theta(n)$ .

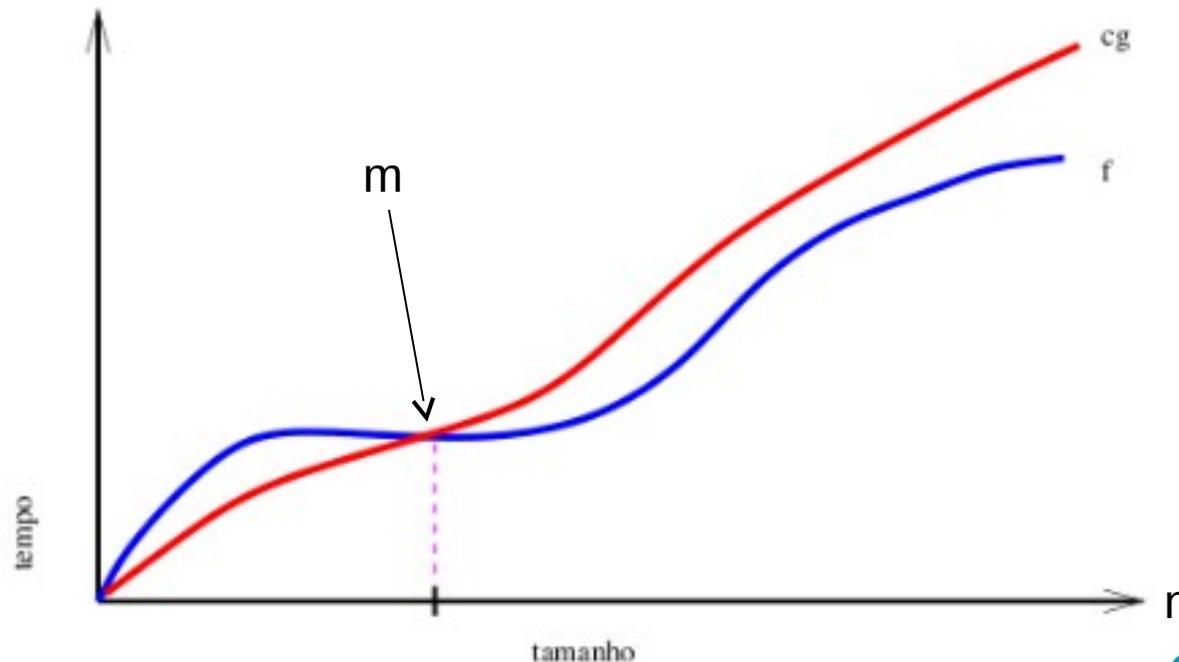
Função de custo	Computador Atual (C)	Computador 100C	Computador 1000C
$n$	$x$	$100x$	$1000x$
$n^2$	$x$	$10x$	$31.6x$
$n^3$	$x$	$4,6x$	$10x$
$2^n$	$x$	$x + 6,6$	$x + 10$

# Comportamento Assintótico - Resumindo...

- Se  $f(n)$  é a função de complexidade de um algoritmo A
  - O comportamento assintótico de  $f(n)$  representa o limite do comportamento do custo (complexidade) de A quando  $n$  cresce.
- A análise de um algoritmo (função de complexidade)
  - Geralmente considera apenas algumas operações elementares ou mesmo uma operação elementar (e.g., o número de comparações).
- A complexidade assintótica relata crescimento assintótico das operações elementares.

# Relacionamento Assintótico

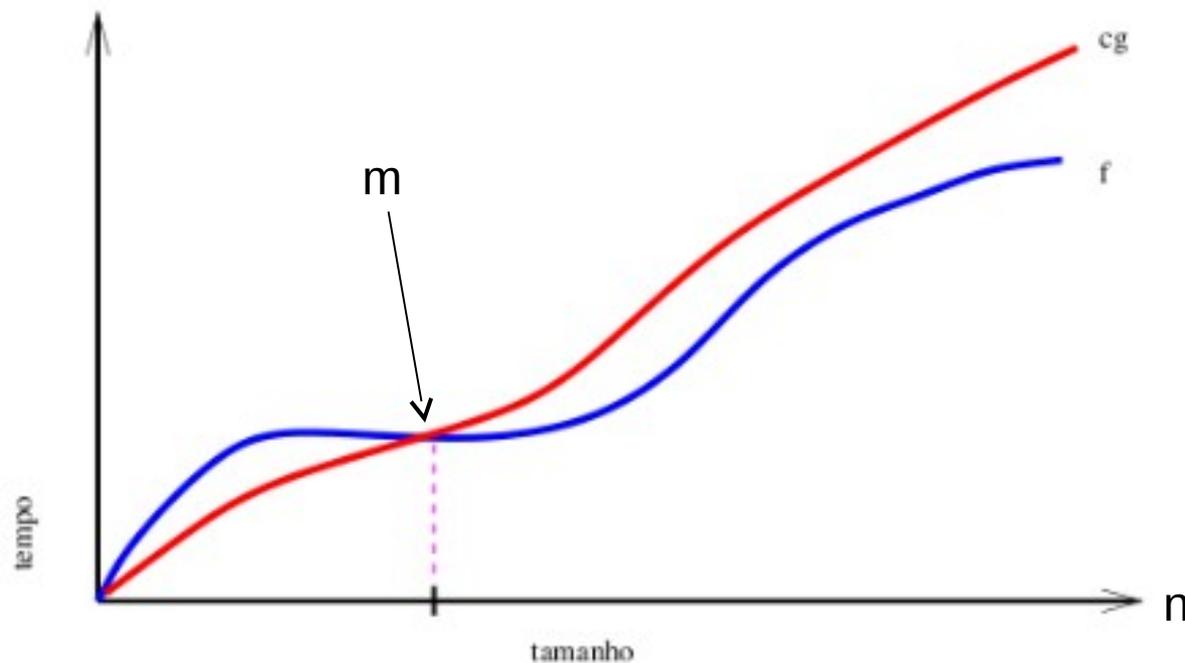
- **Definição:**
  - Uma função  $g(n)$  domina assintoticamente outra função  $f(n)$  se existem duas constantes positivas  $c$  e  $m$  tais que, para  $n \geq m$ , tem-se  $|f(n)| \leq c \cdot |g(n)|$ .



# Relacionamento Assintótico

## Definição:

- Uma função  $g(n)$  domina assintoticamente outra função  $f(n)$  se existem duas constantes positivas  $c$  e  $m$  tais que, para  $n \geq m$ , tem-se  $|f(n)| \leq c \cdot |g(n)|$ .

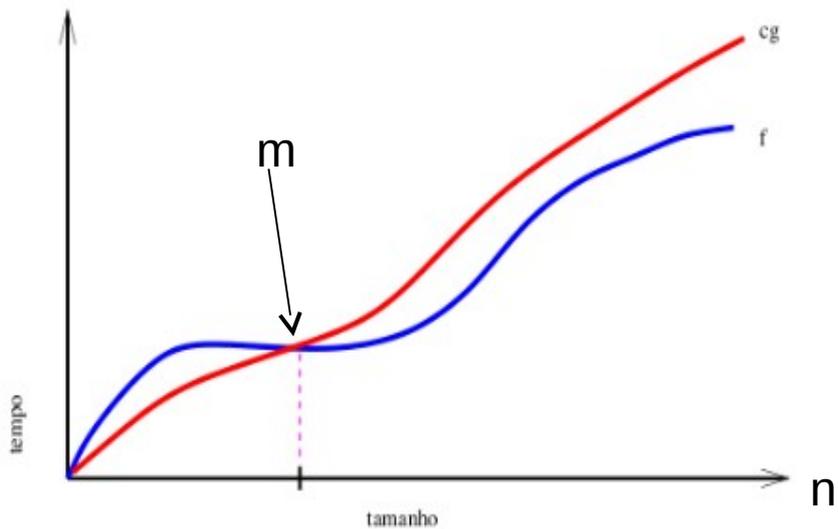


- Exemplo:
- $g(n) = n$  e  $f(n) = n^2$
- Alguém domina alguém?
  - ???

# Relacionamento Assintótico

## Definição:

- Uma função  $g(n)$  domina assintoticamente outra função  $f(n)$  se existem duas constantes positivas  $c$  e  $m$  tais que, para  $n \geq m$ , tem-se  $|f(n)| \leq c \cdot |g(n)|$ .



## Exemplo:

- $g(n) = n$  e  $f(n) = n^2$

- Alguém domina alguém?

- $|n| \leq |n^2|$  para todo  $n \in \mathbb{N}$

- Para  $c = 1$  e  $m = 0 \Rightarrow |g(n)| \leq |f(n)|$

- Portanto,  $f(n)$  domina assintoticamente  $g(n)$ .

# Relacionamento Assintótico

- $g(n) = n$  e  $f(n) = -n^2$
- Alguém domina alguém?
  - ???

# Relacionamento Assintótico

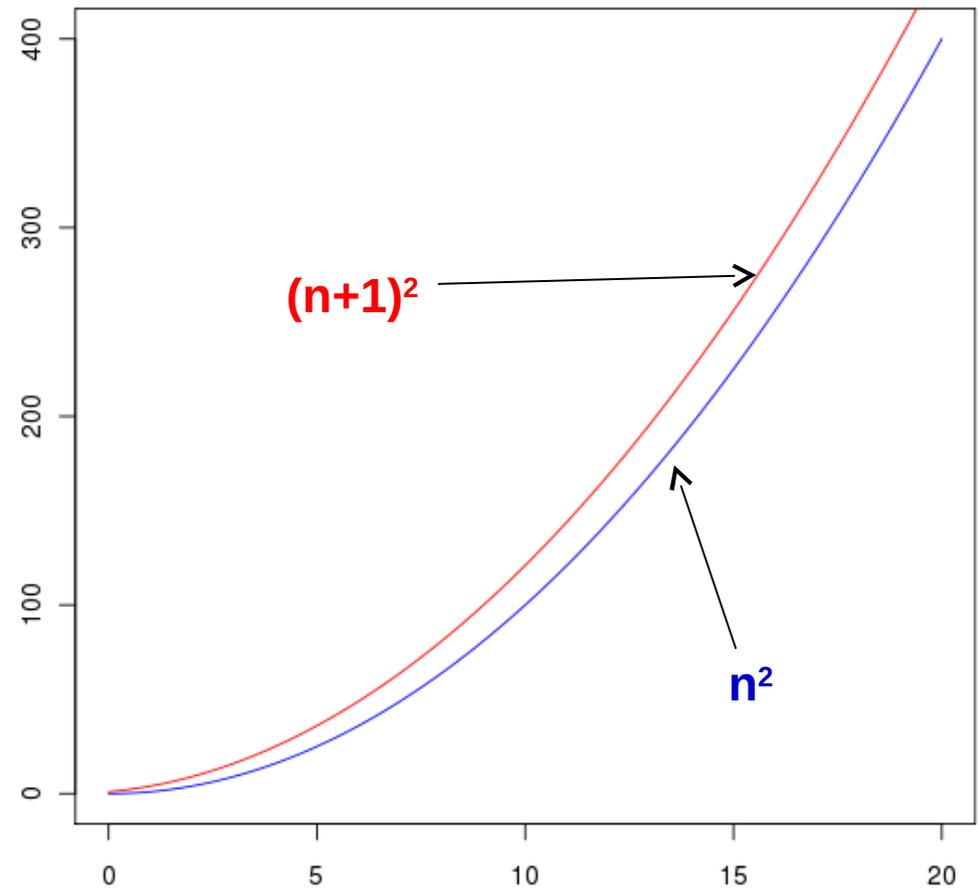
- $g(n) = n$  e  $f(n) = -n^2$
- Alguém domina alguém?
  - $|n| \leq |-n^2|$  para todo  $n \in \mathbb{N}$ .
    - Por ser módulo, o sinal não importa
  - Para  $c = 1$  e  $m = 0 \Rightarrow |g(n)| \leq |f(n)|$ .
- Portanto,  $f(n)$  domina assintoticamente  $g(n)$ .

# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - ???

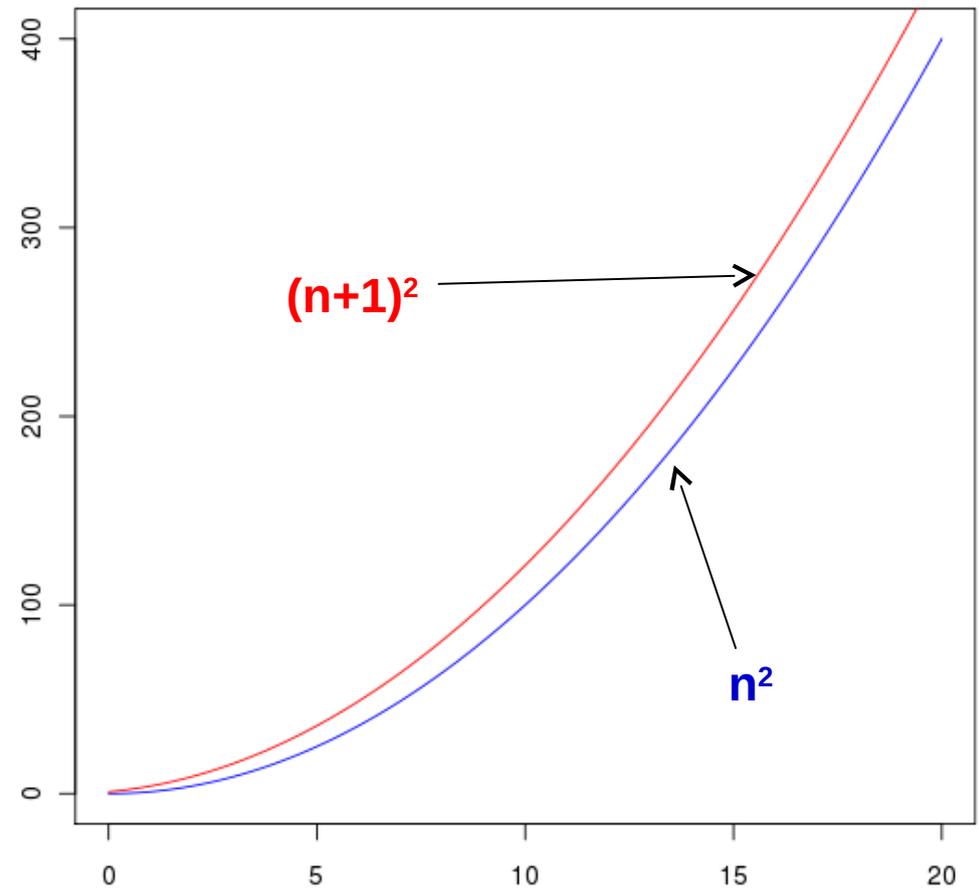
# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - Vamos colocar em um gráfico



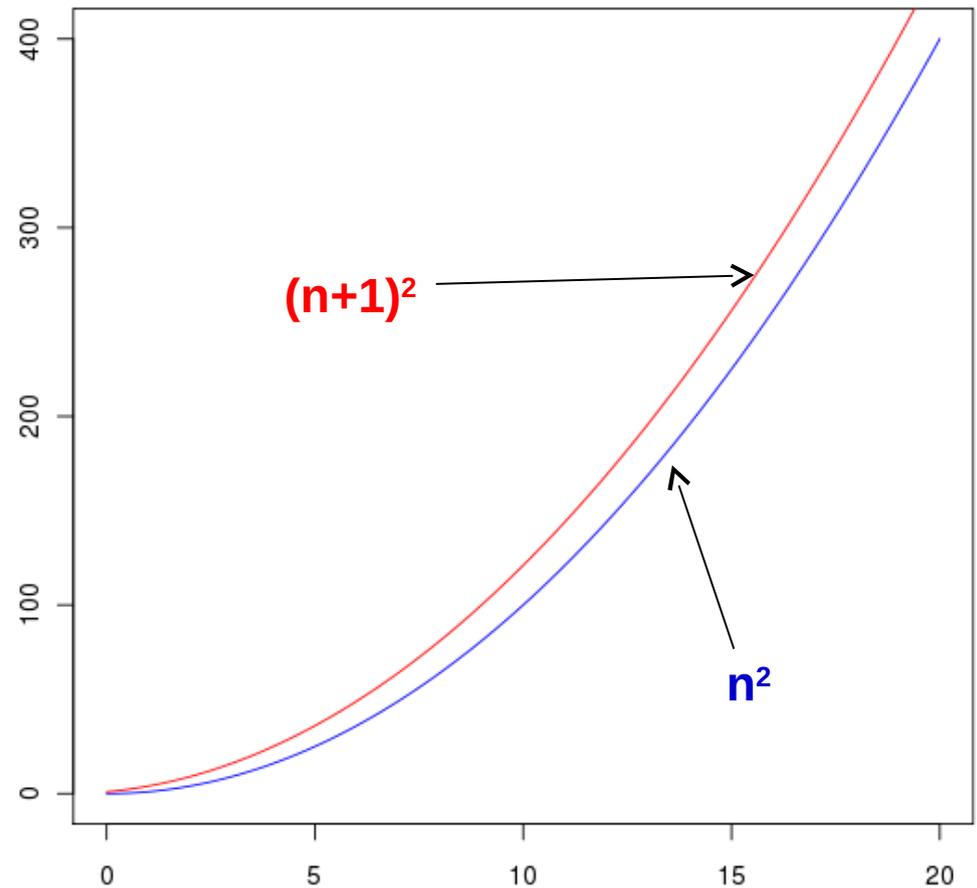
# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - Vamos colocar em um gráfico
  - $|n^2| \leq |(n+1)^2|$ , para  $n \geq 0$
  - $g(n)$  domina  $f(n)$



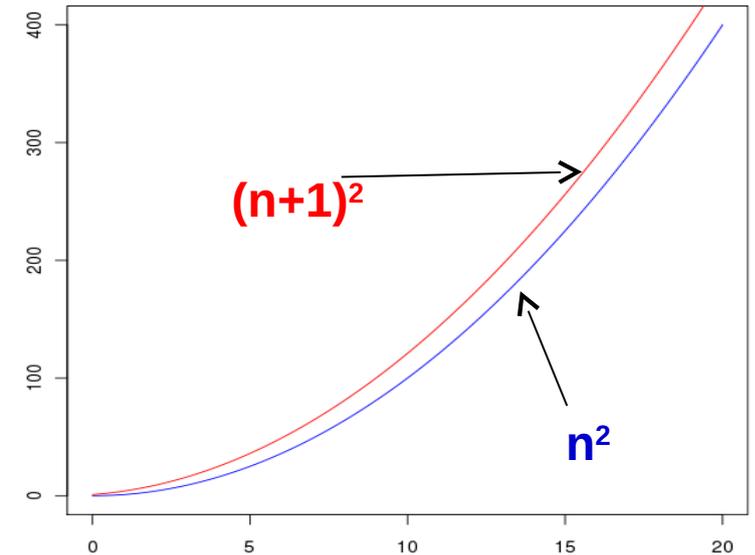
# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - Será somente isso?
  - Não há como  $f(n)$  dominar  $g(n)$ ?
    - ???



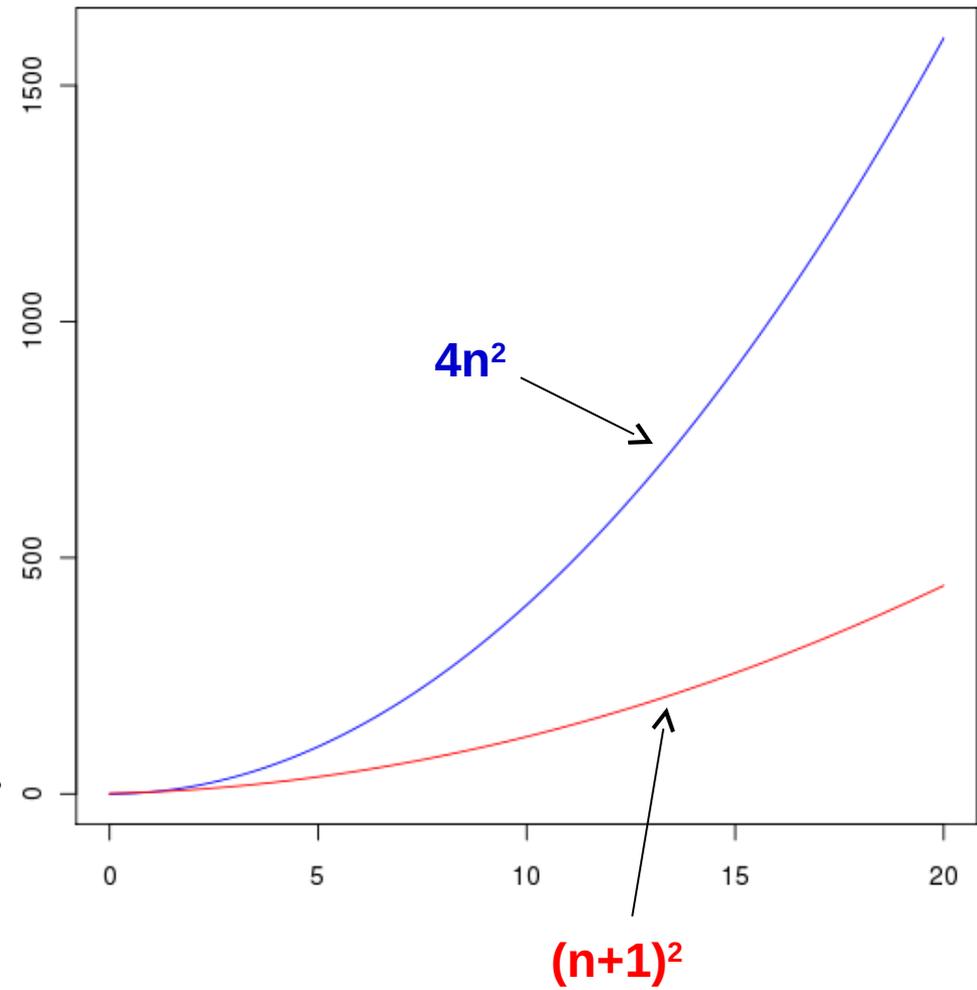
# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - Não há como  $f(n)$  dominar  $g(n)$ ?
    - Lembre que a definição envolve também uma constante.
    - Suponha que queremos  $g(n) \leq cf(n)$
    - Então  $|(n+1)^2| \leq |cn^2|$
    - Mas, para isso, basta que  $|(n+1)^2| \leq |(\sqrt{c} n)^2|$ ,
      - ou  $|n+1| \leq |\sqrt{c} n|$
    - Se  $\sqrt{c} = 2$ , ou seja,  $c=4$ , isso é verdade



# Relacionamento Assintótico

- $g(n) = (n+1)^2$  e  $f(n) = n^2$
- Alguém domina alguém?
  - $|(n+1)^2| \leq |4n^2|$ , para  $n \geq 1$
  - $f(n)$  domina  $g(n)$ , para  $n \geq 1$
- Nesse caso, dizemos que  $f(n)$  e  $g(n)$  dominam assintoticamente uma a outra.



# Notação O

- Knuth(1971) \* criou a notação **O** (lê-se "O grande") para expressar que  **$g(n)$**  domina assintoticamente  **$f(n)$** 
  - Escreve-se  **$f(n) = O(g(n))$**  e lê-se: " **$f(n)$**  é da ordem no máximo  **$g(n)$** ".
- Para que serve isto para o Bacharel em Sistemas de Informação?

---

\*Knuth, D.E. (1971) "Mathematical Analysis of Algorithms". *Proceedings IFIP Congress 71, vol. 1, North Holland, Amsterdam, Holanda, 135-143.*

# Notação O

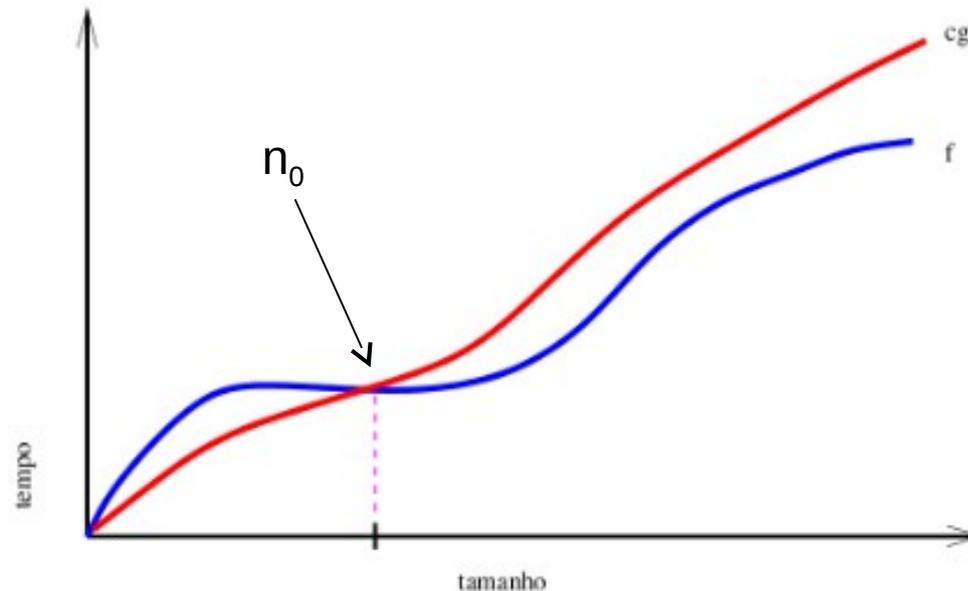
- Knuth(1971) \* criou a notação **O** (lê-se "O grande") para expressar que  **$g(n)$**  domina assintoticamente  **$f(n)$** 
  - Escreve-se  **$f(n) = O(g(n))$**  e lê-se: " **$f(n)$**  é da ordem no máximo  **$g(n)$** ".
- Para que serve isto ?
  - Muitas vezes calcular a função de complexidade  **$g(n)$**  de um algoritmo A é complicado.
  - É mais fácil determinar que  **$f(n)$**  é  **$O(g(n))$** , isto é, que assintoticamente  **$f(n)$**  cresce no máximo como  **$g(n)$** .

---

\*Knuth, D.E. (1971) "Mathematical Analysis of Algorithms". *Proceedings IFIP Congress 71, vol. 1, North Holland, Amsterdam, Holanda, 135-143.*

# Notação O

- **Definição:**
  - $O(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}$
- Informalmente, dizemos que, se  $f(n) \in O(g(n))$ , então  $f(n)$  cresce no máximo tão rapidamente quanto  $g(n)$ .



# Notação O

- Definição:

- $O(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}$

- $\frac{3}{2}n^2 - 2n \in O(n^2) ?$

- ???

# Notação O

- **Definição:**

- $O(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq f(n) \leq cg(n), \text{ para todo } n \geq n_0\}$

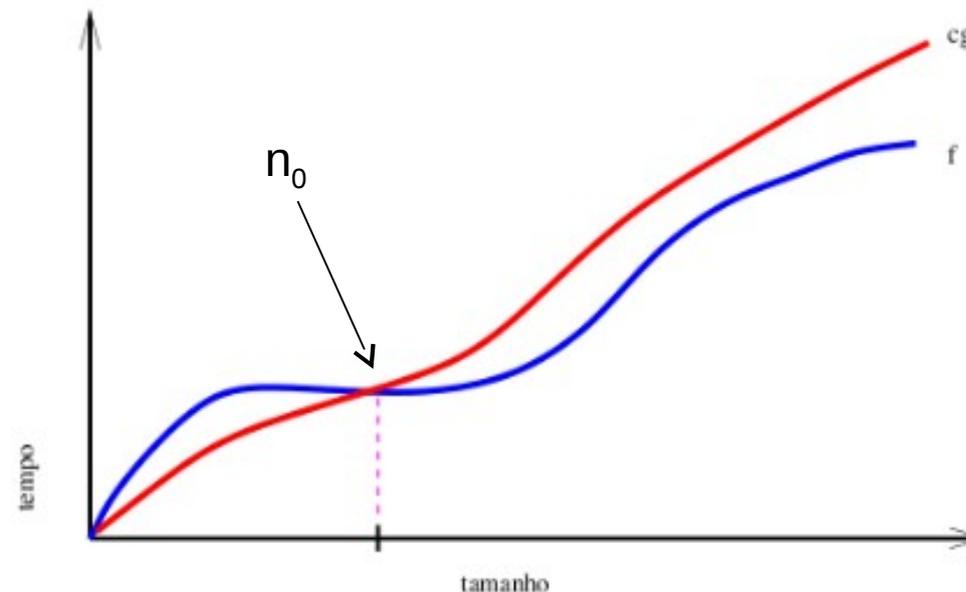
- $\frac{3}{2}n^2 - 2n \in O(n^2)$  ?

- Fazendo  $c = 3/2$ , teremos  $\frac{3}{2}n^2 - 2n \leq \frac{3}{2}n^2$  , para  $n_0 \geq 2$

- Outras constantes podem existir, mas o que importa é que existe alguma escolha para as constantes

# Notação O

- Usamos a notação  $O$  para dar um limite superior sobre uma função, dentro de um fator constante.



# Notação $O$

- Com a notação  $O$  podemos descrever frequentemente o tempo de execução de um algoritmo apenas inspecionando a estrutura global do algoritmo.
  - **Exemplo:**
    - estrutura de laço duplamente aninhado no algoritmo *insertion-sort* (visto anteriormente) produz um limite superior  $O(n^2)$  no pior caso:
      - custo do laço interno é limitado na parte superior por  $O(1)$  (constante)
      - índices  $i$  e  $j$  são no máximo  $n$
      - laço interno é executado no máximo uma vez para cada um dos  $n^2$  pares de valores correspondentes a  $i$  e  $j$

# Notação $O$ - o pior caso

- Como a notação  $O$  dá um limite superior, quando empregado ao pior caso...

# Notação $O$ - o pior caso

- Como a notação  $O$  dá um limite superior, quando empregado ao pior caso...
  - indica que esse limite vale para **qualquer** instância daquele algoritmo.

# Notação $O$ - o pior caso

- Como a notação  $O$  dá um limite superior, quando empregado ao pior caso...
  - indica que esse limite vale para **qualquer** instância daquele algoritmo.
- Assim, o limite  $O(n^2)$  do pior caso do insertion sort também se aplica a qualquer entrada
- Veremos que o mesmo não é verdadeiro para a notação  $\Theta$

# Operações com a notação $O$

$$f(n) = O(f(n))$$

$$c \times f(n) = O(f(n)), c \text{ é uma constante}$$

$$O(f(n)) + O(f(n)) = O(f(n))$$

$$O(O(f(n))) = O(f(n))$$


$$O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$$

$$O(f(n))O(g(n)) = O(f(n)g(n))$$

$$f(n)O(g(n)) = O(f(n)g(n))$$

# Operações com a notação $O$

- A regra  $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$  pode ser usada para calcular o tempo de execução de uma sequência de trechos de um programa
  - Suponha 3 trechos:  $O(n)$ ,  $O(n^2)$  e  $O(n \log n)$
  - Qual o tempo de execução do algoritmo como um todo?
    - ???

# Operações com a notação O

- A regra  $O(f(n)) + O(g(n)) = O(\max(f(n), g(n)))$  pode ser usada para calcular o tempo de execução de uma sequência de trechos de um programa
  - Suponha 3 trechos:  $O(n)$ ,  $O(n^2)$  e  $O(n \log n)$
  - Qual o tempo de execução do algoritmo como um todo?
    - Lembre-se que o tempo de execução é a soma dos tempos de cada trecho
    - $O(n) + O(n^2) + O(n \log n) = \max(O(n), O(n^2), O(n \log n)) = O(n^2)$

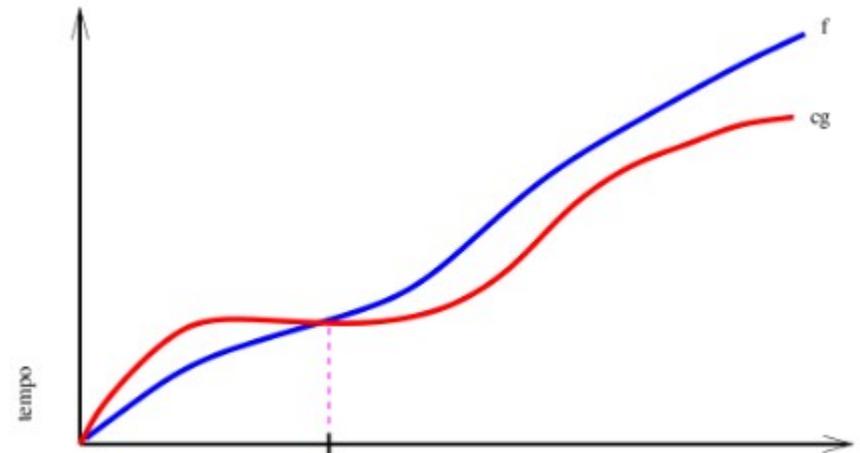
# Notação $\Omega$

- **Definição:**

- $\Omega(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), \text{ para todo } n \geq n_0\}$

- Informalmente, dizemos que, se  $f(n) \in \Omega(g(n))$ , então  $f(n)$  cresce no mínimo tão lentamente quanto  $g(n)$ .

- Note que se  $f(n) \in O(g(n))$  define um limite superior para  $f(n)$ ,  $\Omega(g(n))$  define um limite inferior



# Notação $\Omega$

- Definição:
  - $\Omega(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), \text{ para todo } n \geq n_0\}$
  - $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$  ?
    - ???

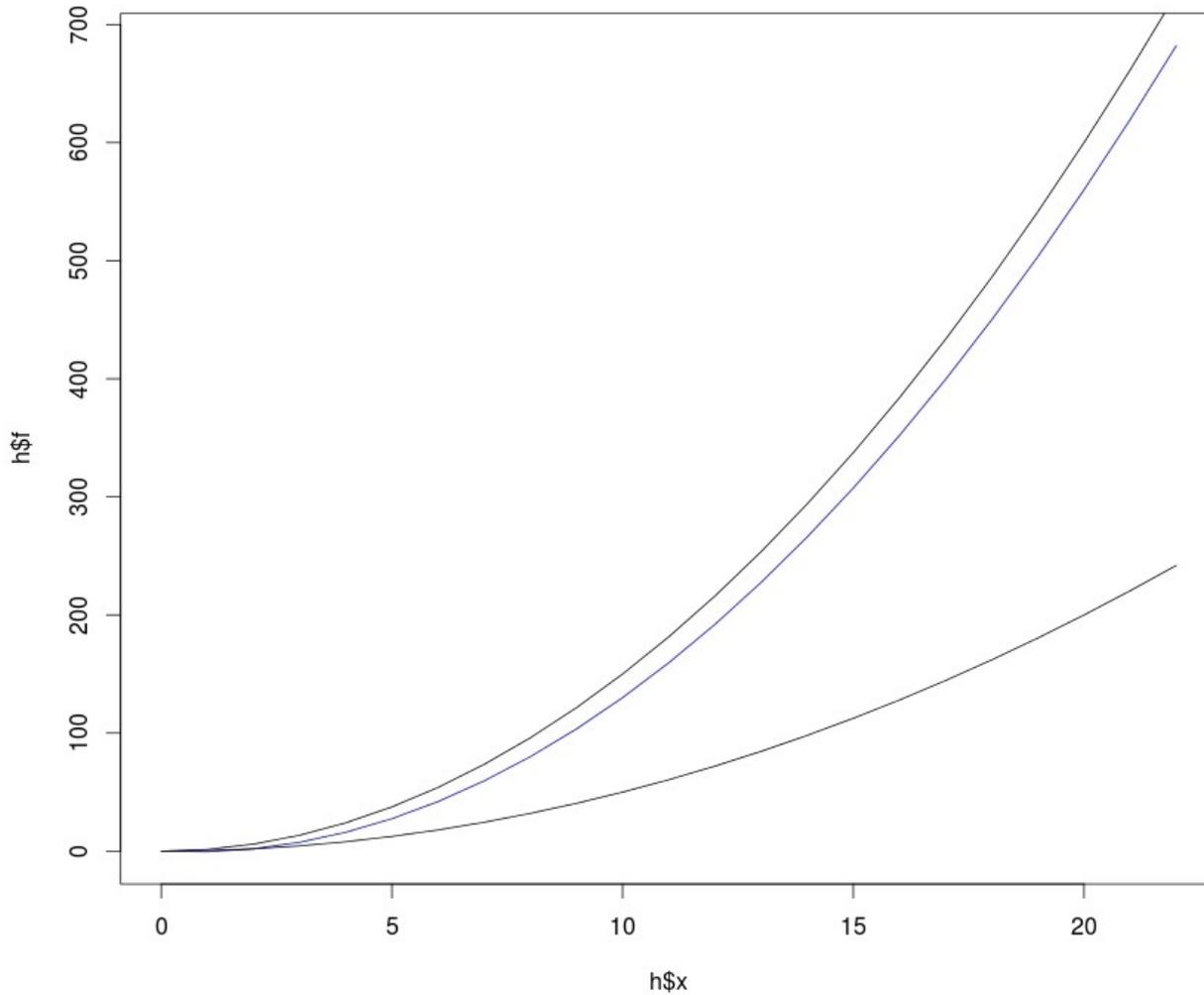
# Notação $\Omega$

- Definição:

- $\Omega(g(n)) = \{f(n): \text{existem constantes positivas } c \text{ e } n_0 \text{ tais que } 0 \leq cg(n) \leq f(n), \text{ para todo } n \geq n_0\}$
- $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$  ?

- Fazendo  $c = 1/2$ , teremos  $\frac{3}{2}n^2 - 2n \geq \frac{1}{2}n^2$  , para  $n_0 \geq 2$

# Notação $O$ e $\Omega$



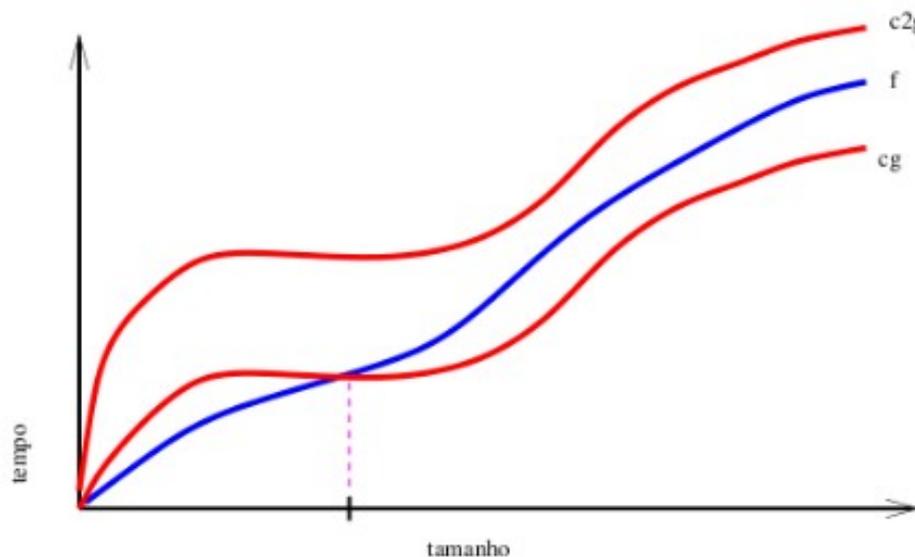
$\frac{3}{2}n^2 - 2n$   
 $\frac{3}{2}n^2$   
 $\frac{1}{2}n^2$

# Notação $\Theta$

## Definição:

- $\Theta(g(n)) = \{f(n): \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \text{ para todo } n \geq n_0\}$

Informalmente, dizemos que, se  $f(n) \in \Theta(g(n))$ , então  $f(n)$  cresce tão rapidamente quanto  $g(n)$ .



# Notação $\Theta$

- Definição:

- $\Theta(g(n)) = \{f(n): \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \text{ para todo } n \geq n_0\}$

- $\frac{3}{2}n^2 - 2n \in \Theta(n^2)?$

- ???

# Notação $\Theta$

- Definição:

- $\Theta(g(n)) = \{f(n): \text{existem constantes positivas } c_1, c_2 \text{ e } n_0 \text{ tais que } 0 \leq c_1g(n) \leq f(n) \leq c_2g(n), \text{ para todo } n \geq n_0\}$

- $\frac{3}{2}n^2 - 2n \in \Theta(n^2)?$

- Fazendo  $c_1 = 1/2$  e  $c_2 = 3/2$  teremos  $|\frac{1}{2}n^2| \leq |\frac{3}{2}n^2 - 2n| \leq |\frac{3}{2}n^2|$

para  $n_0 \geq 2$

# Notação $\Theta$

- Mas, já vimos que:

- $\frac{3}{2}n^2 - 2n \in O(n^2) \rightarrow \left| \frac{3}{2}n^2 - 2n \right| \leq \left| \frac{3}{2}n^2 \right|$

- $\frac{3}{2}n^2 - 2n \in \Omega(n^2) \rightarrow \left| \frac{1}{2}n^2 \right| \leq \left| \frac{3}{2}n^2 - 2n \right| \quad \text{e ...}$

- $\frac{3}{2}n^2 - 2n \in \Theta(n^2) \rightarrow \left| \frac{1}{2}n^2 \right| \leq \left| \frac{3}{2}n^2 - 2n \right| \leq \left| \frac{3}{2}n^2 \right|$

- Será coincidência?

- ???

# Notação $\Theta$

- Mas, já vimos que:

- $\frac{3}{2}n^2 - 2n \in O(n^2) \rightarrow \left| \frac{3}{2}n^2 - 2n \right| \leq \left| \frac{3}{2}n^2 \right|$

- $\frac{3}{2}n^2 - 2n \in \Omega(n^2) \rightarrow \left| \frac{1}{2}n^2 \right| \leq \left| \frac{3}{2}n^2 - 2n \right| \quad \text{e ...}$

- $\frac{3}{2}n^2 - 2n \in \Theta(n^2) \rightarrow \left| \frac{1}{2}n^2 \right| \leq \left| \frac{3}{2}n^2 - 2n \right| \leq \left| \frac{3}{2}n^2 \right|$

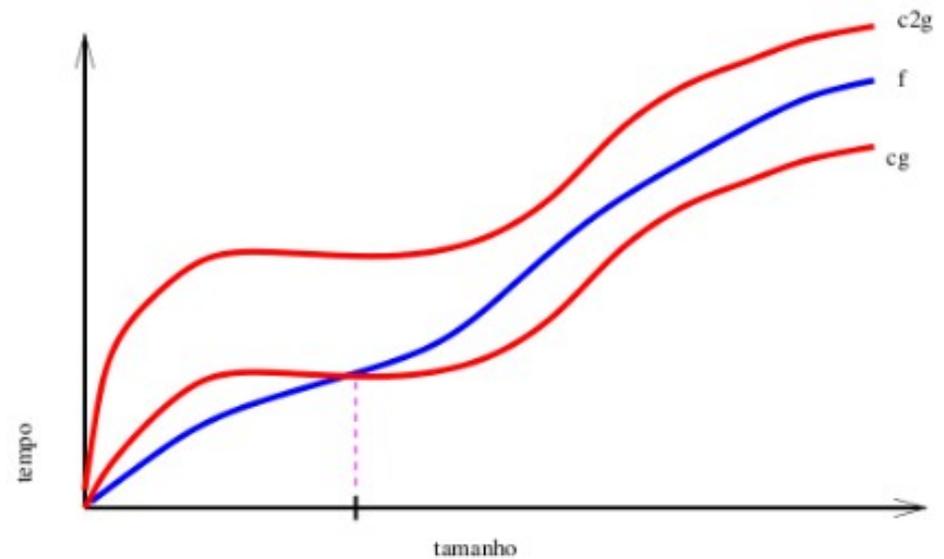
- Será coincidência?

- Não!

- Se  $f(n) \in O(g(n))$  e  $f(n) \in \Omega(g(n))$ , então  $f(n) \in \Theta(g(n))$

# Notação $\Theta$

- Mas:
  - Será coincidência?
    - Não!
    - Se  $f(n) \in O(g(n))$  e  $f(n) \in \Omega(g(n))$ , então  $f(n) \in \Theta(g(n))$



# Notação $\Theta$ – pior caso

- O tempo limite de  $\Theta(n^2)$  para o pior caso *do insertion sort*
- Não implica um tempo  $\Theta(n^2)$  para qualquer entrada
- Por exemplo, se pegarmos o melhor caso, vemos que ele tem  $\Theta(n)$

# Notação $o$

- **Definição:**
  - $o(g(n)) = \{f(n): \text{para toda constante positiva } c, \text{ existe uma constante } n_0 > 0 \text{ tal que } 0 \leq f(n) < cg(n), \text{ para todo } n \geq n_0\}$
- Informalmente, dizemos que, se  $f(n) \in o(g(n))$ , então  $f(n)$  cresce mais lentamente que  $g(n)$ .
  - Intuitivamente, na notação  $o$ , a função  $f(n)$  tem crescimento muito menor que  $g(n)$  quando  $n$  tende para o infinito

# Notação $o$

- $1000n^2 \in o(n^3)$  ?
  - ???

# Notação o

- $1000n^2 \in o(n^3)$  ?
  - Para todo valor de  $c$ , um  $n_0$  que satisfaz a definição é:

$$n_0 = \left\lceil \frac{1000}{c} \right\rceil + 1$$

# Notação $o$

- Qual a diferença entre  $O$  e  $o$ ?
  - $O$ : existem constantes positivas  $c$  e  $n_0$  tais que  $0 \leq f(n) \leq cg(n)$ , para todo  $n \geq n_0$ 
    - A expressão  $0 \leq f(n) \leq cg(n)$  é válida para alguma constante  $c > 0$
  - $o$ : para toda constante positiva  $c$ , existe uma constante  $n_0 > 0$  tal que  $0 \leq f(n) < cg(n)$ , para todo  $n \geq n_0$ 
    - A expressão  $0 \leq f(n) < cg(n)$  é válida para toda constante  $c > 0$

# Notação $\omega$

- **Definição:**
  - $\omega(g(n)) = \{f(n): \text{para toda constante positiva } c, \text{ existe uma constante } n_0 > 0 \text{ tal que } 0 \leq cg(n) < f(n), \text{ para todo } n \geq n_0 \}$
- Informalmente, dizemos que, se  $f(n) \in \omega(g(n))$ , então  $f(n)$  cresce mais rapidamente que  $g(n)$ .
  - Intuitivamente, na notação  $\omega$ , a função  $f(n)$  tem crescimento muito maior que  $g(n)$  quando  $n$  tende para o infinito

# Notação $\omega$

- $\omega$  está para  $\Omega$ , da mesma forma que  $o$  está para  $O$ 
  - $O$  e  $\Omega$  são chamados de assintoticamente firmes
- $|\frac{1}{1000}n^2| \in \omega(n)$ ?
  - ???

# Notação $\omega$

- $\omega$  está para  $\Omega$ , da mesma forma que  $o$  está para  $O$ 
  - $O$  e  $\Omega$  são chamados de assintoticamente firmes
- $|\frac{1}{1000}n^2| \in \omega(n)$ ?
- Para todo valor de  $c$ , um  $n_0$  que satisfaz a definição é:

$$n_0 = |1000c| + 1$$

# Propriedades das Classes

## Reflexividade:

$$f(n) \in O(f(n)).$$

$$f(n) \in \Omega(f(n)).$$

$$f(n) \in \Theta(f(n)).$$

## Simetria:

$$f(n) \in \Theta(g(n)) \text{ se, e somente se, } g(n) \in \Theta(f(n)).$$

## Simetria Transposta:

$$f(n) \in O(g(n)) \text{ se, e somente se, } g(n) \in \Omega(f(n)).$$

$$f(n) \in o(g(n)) \text{ se, e somente se, } g(n) \in \omega(f(n)).$$

# Propriedades das Classes

## Transitividade:

Se  $f(n) \in O(g(n))$  e  $g(n) \in O(h(n))$ , então  $f(n) \in O(h(n))$ .

Se  $f(n) \in \Omega(g(n))$  e  $g(n) \in \Omega(h(n))$ , então  $f(n) \in \Omega(h(n))$ .

Se  $f(n) \in \Theta(g(n))$  e  $g(n) \in \Theta(h(n))$ , então  $f(n) \in \Theta(h(n))$ .

Se  $f(n) \in o(g(n))$  e  $g(n) \in o(h(n))$ , então  $f(n) \in o(h(n))$ .

Se  $f(n) \in \omega(g(n))$  e  $g(n) \in \omega(h(n))$ , então  $f(n) \in \omega(h(n))$ .

# Exercício

Quais as relações de comparação assintótica ( $O$ ,  $\Omega$ ,  $\Theta$ ) das funções:

$$f_1(n) = 2^\pi$$

$$f_2(n) = 2^n$$

$$f_3(n) = n \log n$$

$$f_4(n) = \log n$$

$$f_5(n) = 100n^2 + 150000n$$

$$f_6(n) = n + \log n$$

$$f_7(n) = n^2$$

$$f_8(n) = n$$

	$f_1$	$f_2$	$f_3$	$f_4$	$f_5$	$f_6$	$f_7$	$f_8$
$f_1$	$\Theta$							
$f_2$		$\Theta$						
$f_3$			$\Theta$					
$f_4$				$\Theta$				
$f_5$					$\Theta$			
$f_6$						$\Theta$		
$f_7$							$\Theta$	
$f_8$								$\Theta$

# Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002 (Capítulo 3).
- Michael T. Goodrich & Roberto Tamassia. Estruturas de Dados e Algoritmos em Java. Editora Bookman, 4a. Ed. 2007 (Capítulo 4).
- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004 (Seção 1.3).
- Notas de aula dos professores Marcos Chaim, Cid de Souza, Cândida da Silva e Delano M. Beder.

# Complexidade Assintótica

**Professores:**

**Norton T. Roman**

**Fátima L. S. Nunes**