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	c_8	n-1
10	**fim para**		

Análise de Algoritmos

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8 (n-1)$

		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

$$T(n)=c1n + c2(n-1) + c4(n-1) + c5\sum_{j=2}^{n} t_j + c6\sum_{j=2}^{n}(t_j-1) + c7\sum_{j=2}^{n}(t_j-1) + c8\,(n-1)$$

- Melhor caso: vetor já ordenado (A[i] ≤ chave na linha 5 ➔ t_j=1 para j=2,3,...,n)
- $T(n)=c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) =$
  $(c_1+c_2+c_4+c_5+c_8)n - (c_2+c_4+c_5+c_8)$
- Tempo de execução, neste caso, pode ser expresso como *an + b* para constantes **a** e **b** que dependem dos custos de instrução c_i ➔ **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... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

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

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

- 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 ➔ **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

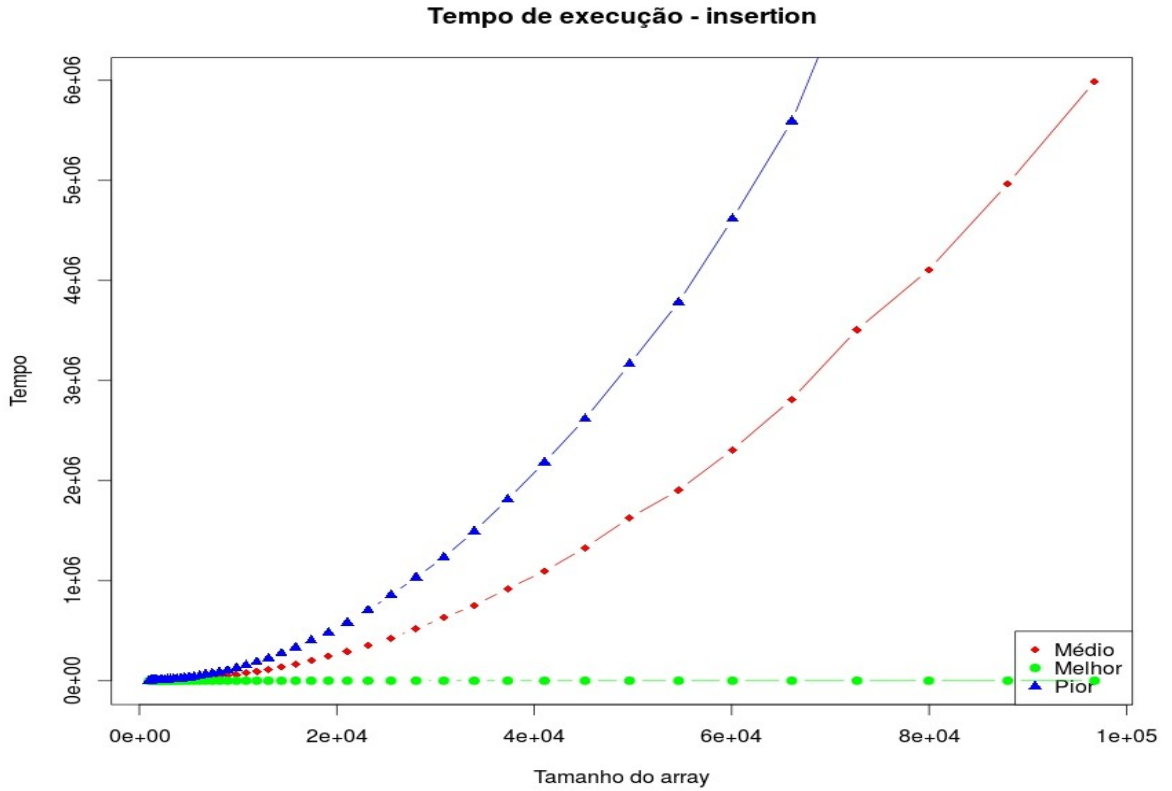
- Nas análises anteriores, foram feitas algumas simplificações em relação às constantes, chegando à função linear e à função quadrática
- **Taxa de crescimento** ou **ordem de crescimento**:
  - considera apenas o termo inicial de uma fórmula (exemplo: an^2), pois os termos de mais baixa ordem são relativamente 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*[2]) *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: assimptótico.*

  - **Assíntota**

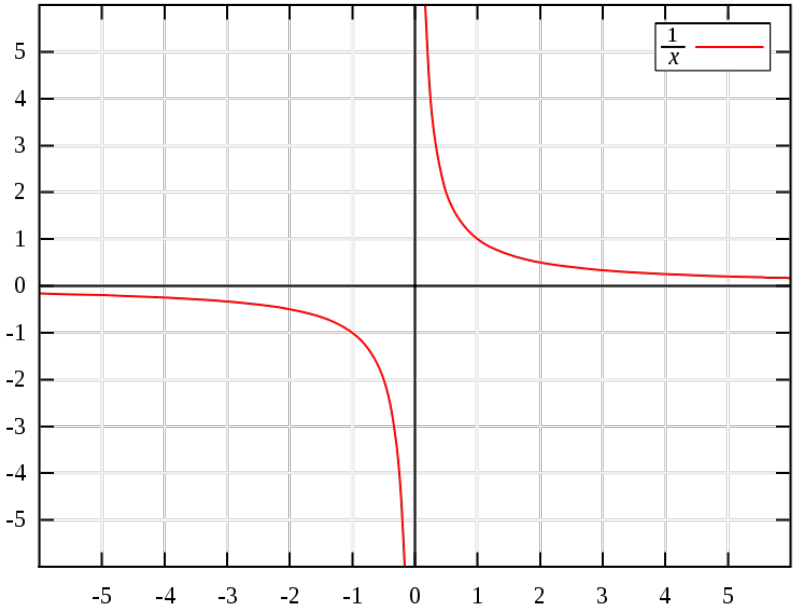
    sf (*gr asýmptotos*) *Geom* Linha reta que se aproxima indefinidamente de uma curva sem nunca poder tocá-la. *Var: assímptota.*

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

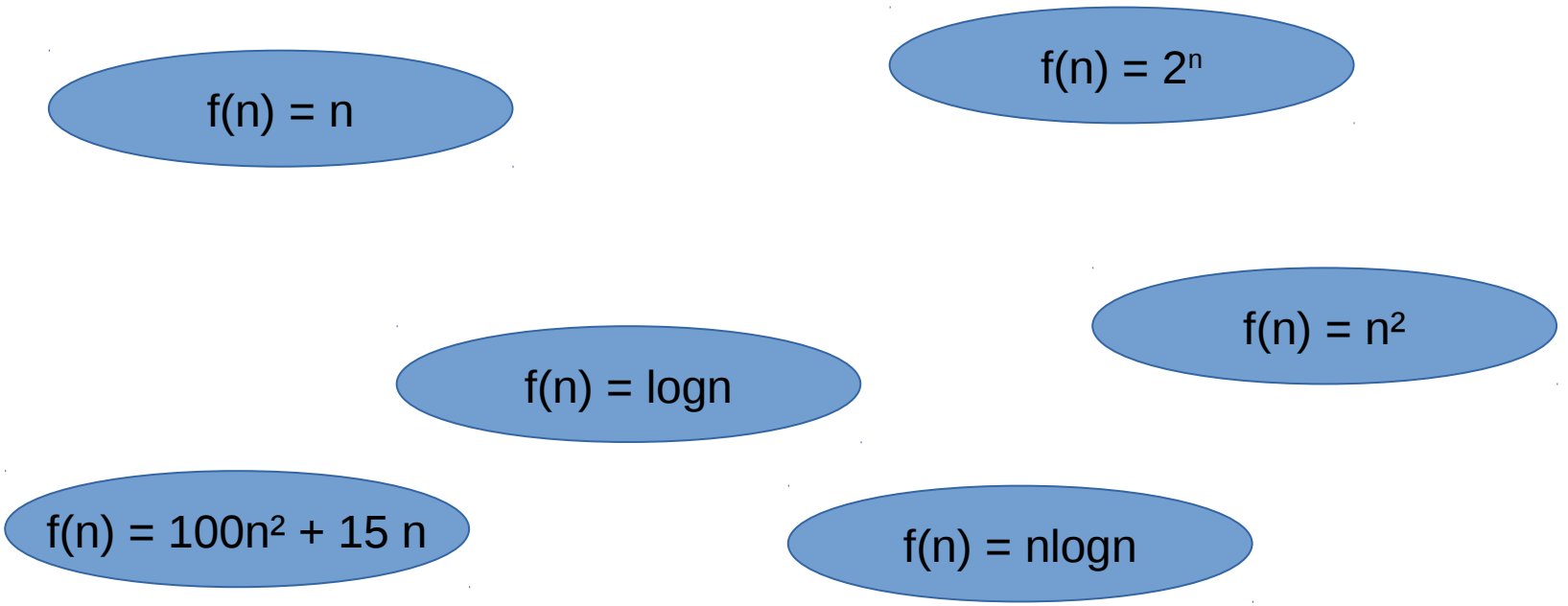
- Escolha do algoritmo não é um problema crítico quando ***n*** é pequeno.
  - O problema é quando ***n*** cresce.
- Por isso, é usual analisar o comportamento das funções de custo quando ***n*** é bastante grande:
  - analisa-se o comportamento assintótico das funções de custo;
  - representa o *limite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

Crescimento Assintótico de Funções

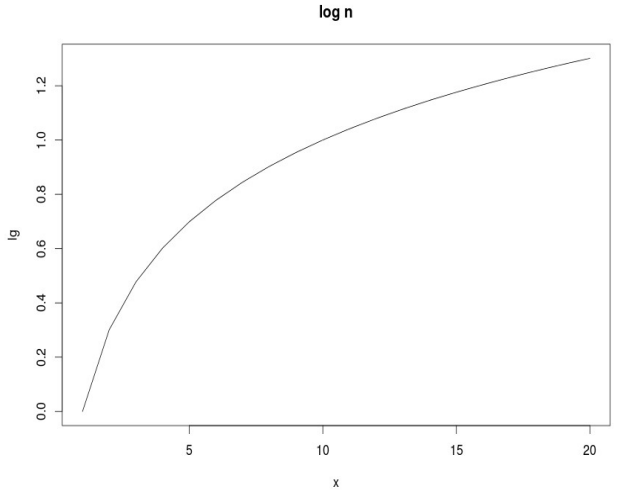
- Eficiência assintótica dos algoritmos:
  - estuda a maneira como o tempo de execução de um algoritmo aumenta com o tamanho da entrada no limite, à medida que o tamanho da entrada aumenta indefinidamente (sem limitação)
  - em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha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

Vamos ordenar

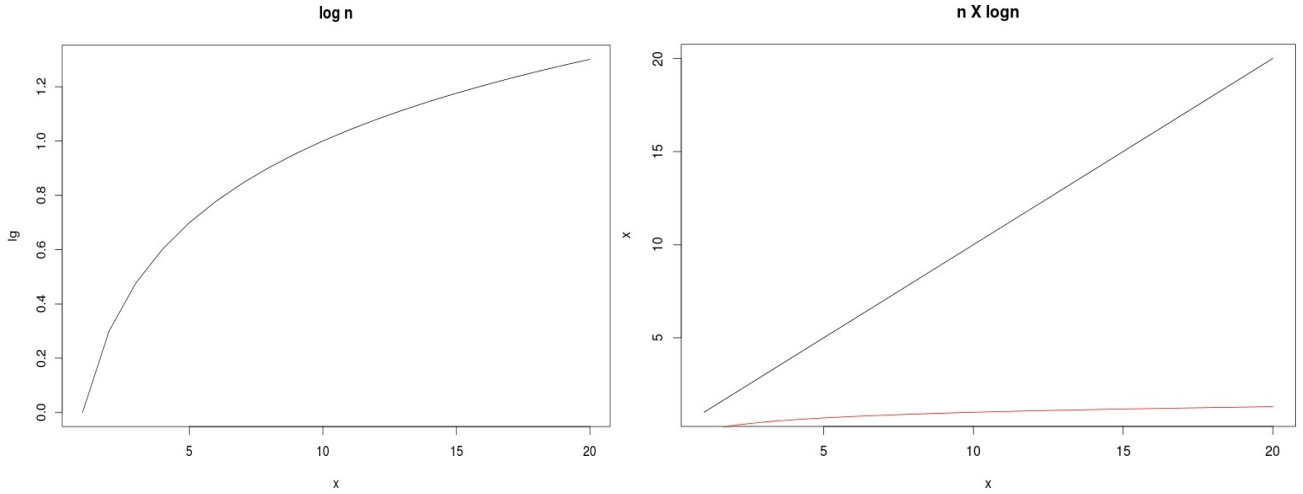
Quais funções “crescem mais”?



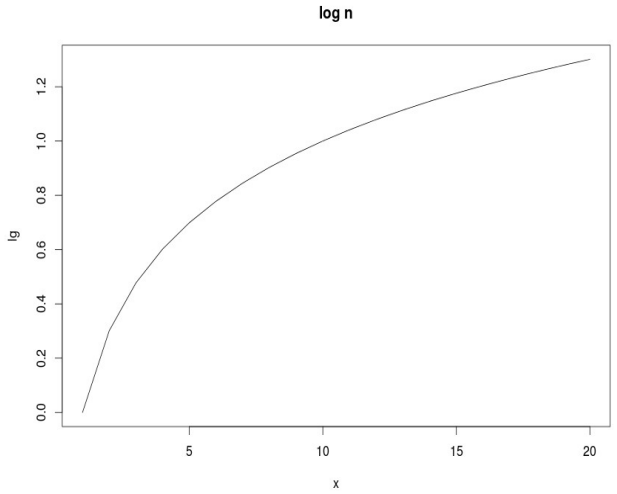
Vamos ordenar

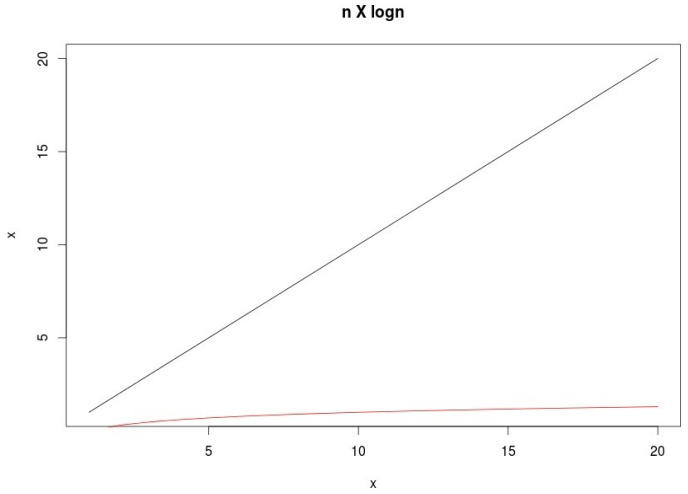


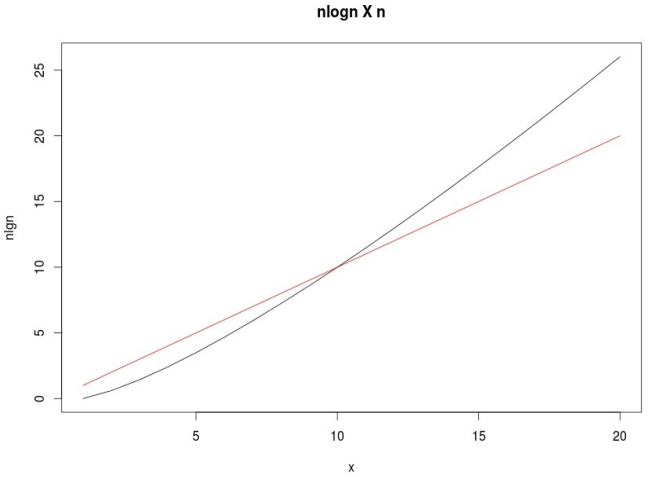
Vamos ordenar



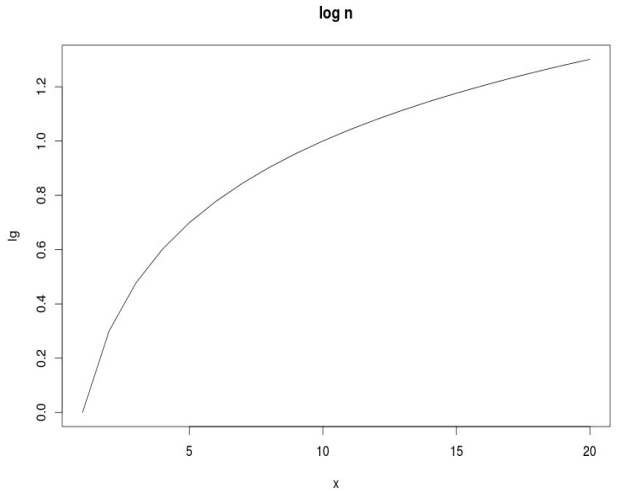
Vamos ordenar

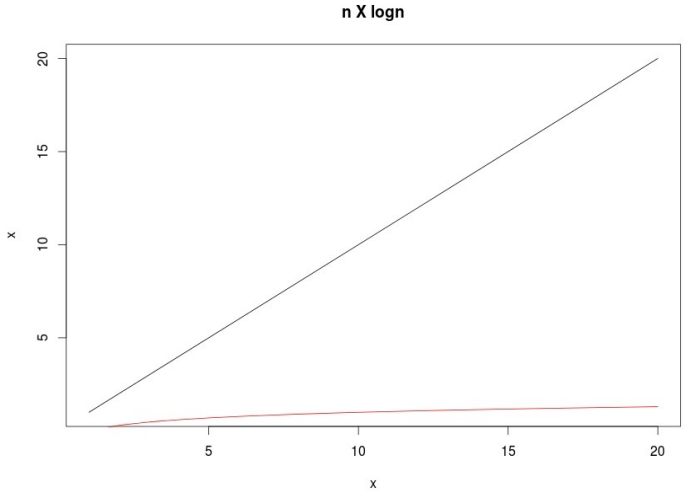


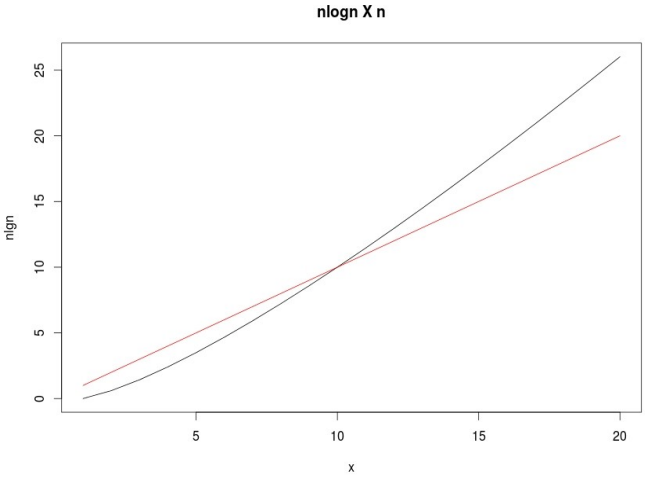


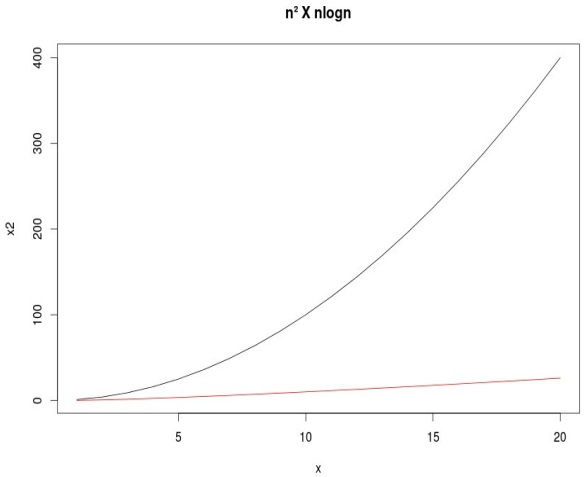


Vamos ordenar

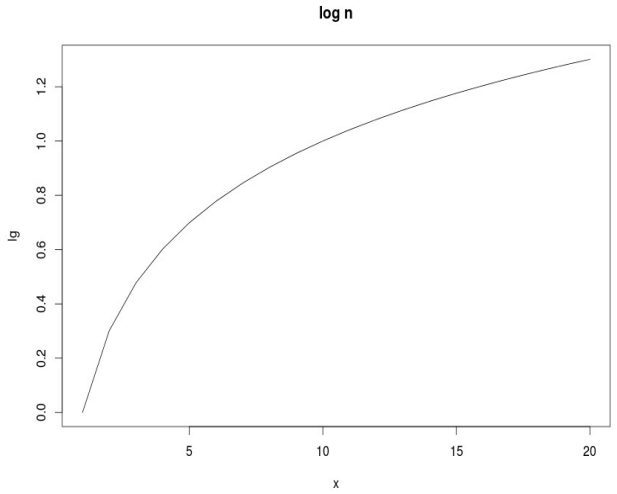


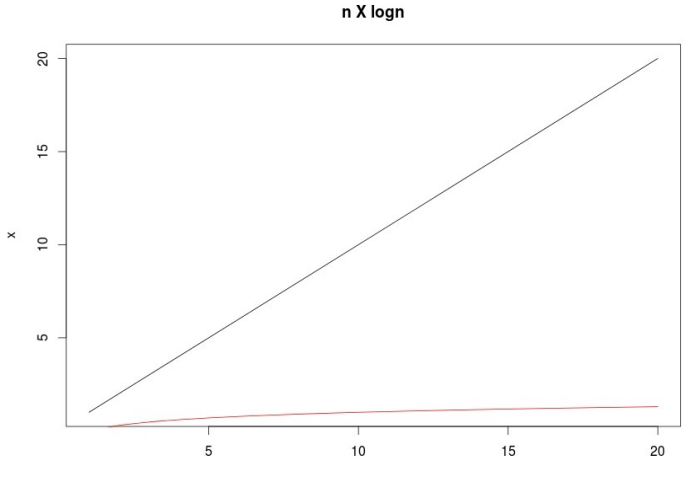


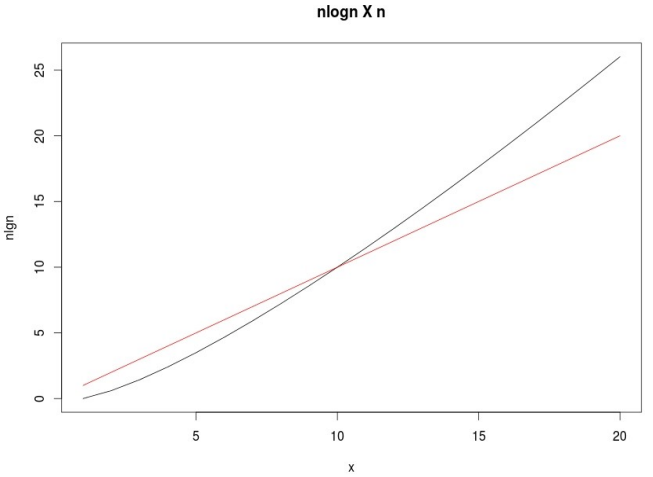


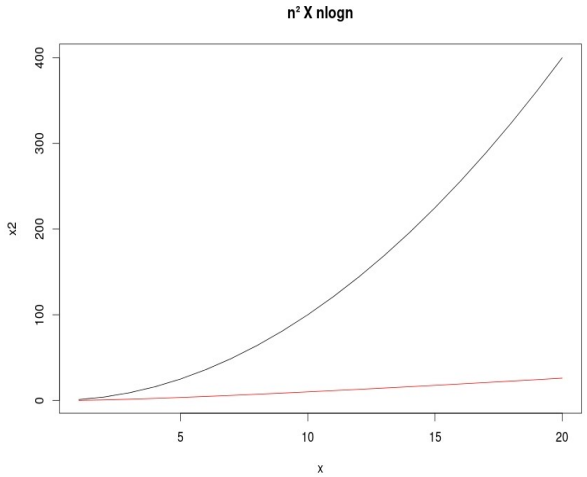


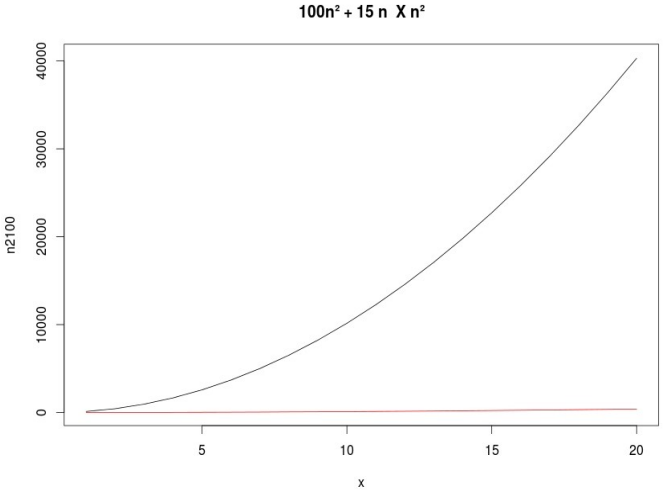
Vamos ordenar



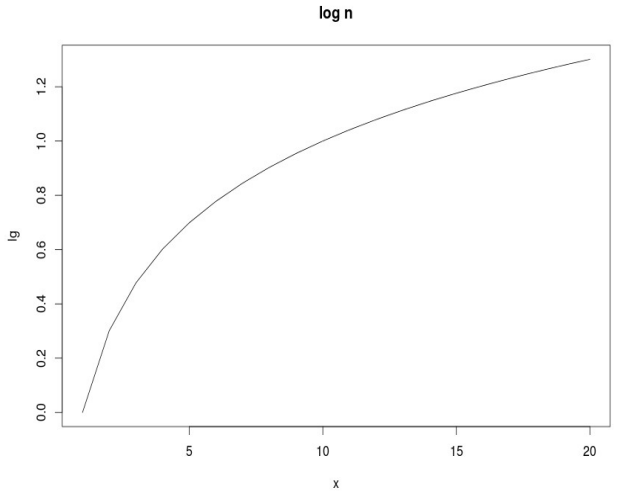


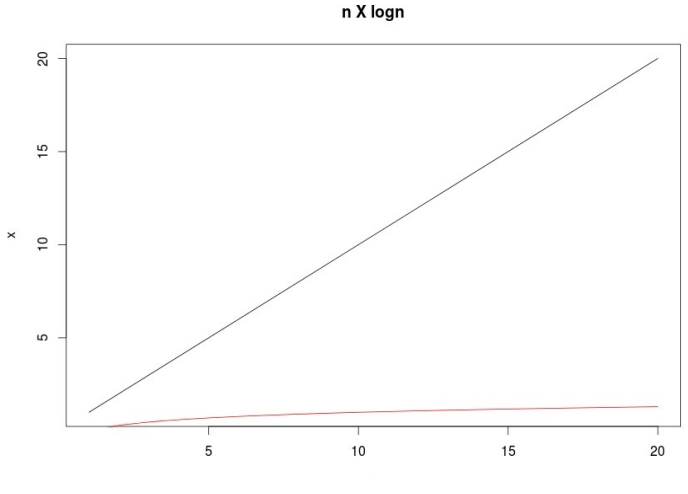


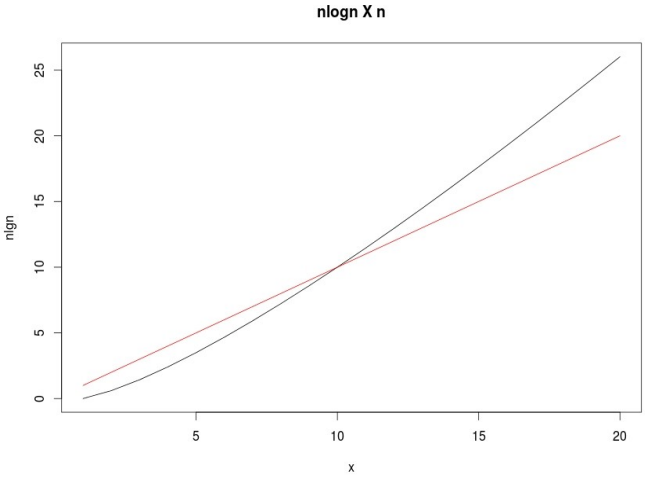


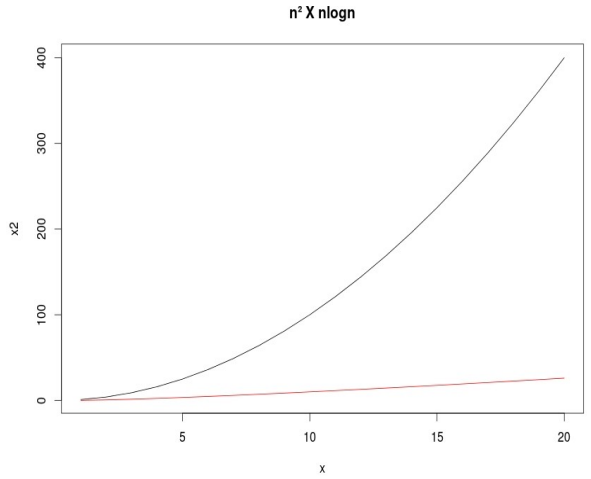


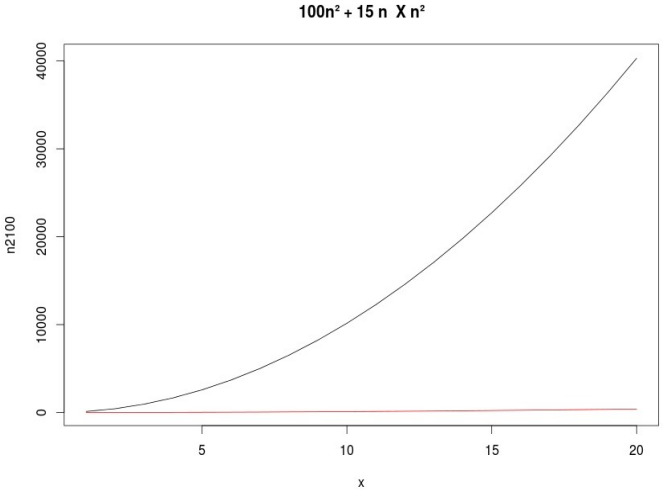
Vamos ordenar

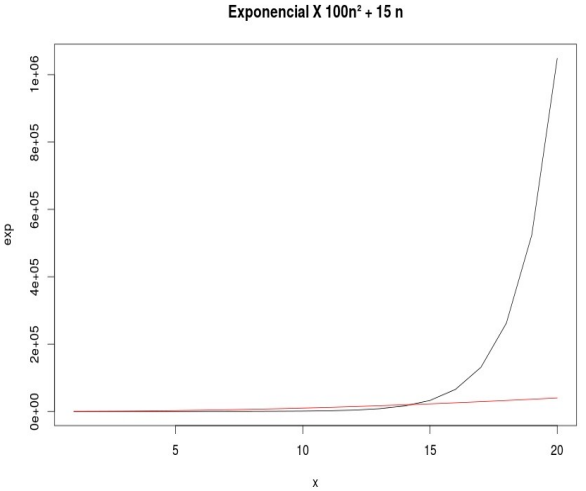












Comportamento Assintótico

- 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.	10^8séc.	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.

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