

Aula 02: Análise de algoritmos — introdução

David Déharbe

Programa de Pós-graduação em Sistemas e Computação

Universidade Federal do Rio Grande do Norte

Centro de Ciências Exatas e da Terra

Departamento de Informática e Matemática Aplicada



Introdução

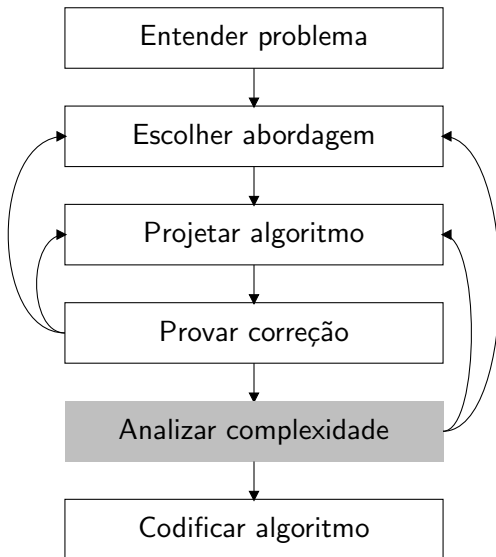
Considerações iniciais

Medição do tamanho da entrada

Medição do tempo de execução

Ordens de crescimento

Notação Θ



analysis

detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation: statistical analysis — an analysis of popular culture.

— Dictionary Apple 2.2.3

analysis

detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation: statistical analysis — an analysis of popular culture.

— Dictionary Apple 2.2.3

- ▶ correção
- ▶ simplicidade
- ▶ generalidade
- ▶ recursos necessários para ser aplicado
 - ▶ tempo de processador
 - ▶ quantidade de memória

analysis

detailed examination of the elements or structure of something, typically as a basis for discussion or interpretation: statistical analysis — an analysis of popular culture.

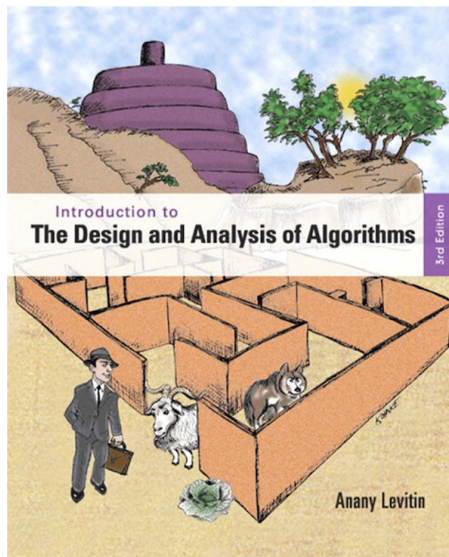
— Dictionary Apple 2.2.3

- ▶ correção
- ▶ simplicidade
- ▶ generalidade
- ▶ recursos necessários para ser aplicado ←
 - ▶ tempo de processador
 - ▶ quantidade de memória

Estrutura da apresentação

1. arcabouço de análise, noção de crescimento assintótico;
2. notações assintóticas; O , Ω , Θ ;
3. análise de algoritmos não recursivos;
4. análise de algoritmos recursivos.

Bibliografia usada



- ▶ eficácia temporal, complexidade temporal:
 - ▶ quão rápido o algoritmo se executa?
 - ▶ quantos ciclos de processadores são necessários para executar o algoritmo?
- ▶ eficácia espacial, complexidade espacial:
 - ▶ quanta memória o algoritmo requer para armazenar os dados que manipula?
 - ▶ quantas unidades de memória precisam ser alocadas?

- ▶ recursos computacionais escassos;
- ▶ computação móvel, computação ubíqua: computação = energia;
- ▶ geralmente a velocidade é um recurso mais crítico;
- ▶ tempo de execução tem sido o aspecto onde ganhos são maiores

- ▶ recursos computacionais escassos;
- ▶ computação móvel, computação ubíqua: computação = energia;
- ▶ geralmente a velocidade é um recurso mais crítico;
- ▶ tempo de execução tem sido o aspecto onde ganhos são maiores

Foco na complexidade temporal

Parâmetro da complexidade

Observação

Para quase todos os algoritmos, quanto maior for o tamanho da entrada, maior é o número de computações necessárias.

É natural querer definir a complexidade de um algoritmo em função do tamanho da entrada.

Exemplos

- ▶ processamento de uma lista: o número de elementos na lista;
- ▶ processamento de uma matriz: o número de linhas e colunas da matriz;
- ▶ processamento em grafo: número de vértices, número de arestas;
- ▶ processamento de números: número de bits usados para representar os números ($\lfloor \log_2 n \rfloor + 1$).
- ▶ verificar se uma fórmula de lógica Booleana é válida: número de variáveis proposicionais.

Exercícios

1. calcular a soma de n números;
2. calcular $n!$;
3. encontrar o maior elemento em uma lista de n elementos;
4. algoritmo para multiplicar dois inteiros decimais de n dígitos cada;
5. crivo de Eratóstenes;
6. algoritmo de Euclides.

Unidade de medição do tempo de execução de um algoritmo

Abordagem ingênua

- ▶ implementar o algoritmo na sua linguagem de programação favorita;
- ▶ medir o tempo de execução da implementação para diferentes entradas é ingênua

Problemas:

- ▶ características do hardware tem influência;
- ▶ compilador tem influência;
- ▶ dificuldade de medição precisa.

Não é satisfatório



Unidade de medição do tempo de execução de um algoritmo

Precisamos de uma abordagem que não dependa de fator externos ao algoritmo.

- ▶ contar quantas vezes cada comando do algoritmo é executado.

Exemplo (Cormen et al.)



Exemplo

INSERTION-SORT(A)

1	for $j = 2$ to $\text{length}[A]$	$c_1 \cdot n$
2	do	
3	$\text{key} \leftarrow A[j]$	$c_2 \cdot (n - 1)$
4	$i \leftarrow j - 1$	$c_3 \cdot (n - 1)$
5	while $i > 0 \wedge A[i] > \text{key}$	$c_4 \cdot \sum_{j=2}^n t_j$
6	do	
7	$A[i + 1] \leftarrow A[i]$	$c_5 \cdot \sum_{j=2}^n (t_j - 1)$
8	$i \leftarrow i - 1$	$c_6 \cdot \sum_{j=2}^n (t_j - 1)$
9	$A[i + 1] \leftarrow \text{key}$	$c_7 \cdot (n - 1)$

- ▶ c_i : custo de executar cada linha
- ▶ t_j : número de vezes que o teste é efetuado para o elemento $A[j]$.
- ▶ custo total: $c_1 \cdot n + c_2 \cdot (n - 1) + c_3 \cdot (n - 1) + c_4 \cdot \sum_{j=2}^n t_j + c_5 \cdot \sum_{j=2}^n (t_j - 1) + c_6 \cdot \sum_{j=2}^n (t_j - 1) + c_7 \cdot (n - 1)$

Exemplo

$$\begin{aligned}T(n) &= c_1.n + c_2.(n-1) + c_3.(n-1) + c_4.\sum_{j=2}^n t_j + \\&\quad c_5.\sum_{j=2}^n (t_j - 1) + c_6.\sum_{j=2}^n (t_j - 1) + c_7.(n-1) \\&= (c_1 + c_2 + c_3 + c_7).n - (c_2 + c_3 + c_7) \\&\quad + c_4.\sum_{j=2}^n t_j + (c_5 + c_6).\sum_{j=2}^n (t_j - 1)\end{aligned}$$

Observamos que a complexidade do algoritmo depende de n , dos t_i e dos c_i .

Exemplo

$$\begin{aligned}T(n) &= c_1 \cdot n + c_2 \cdot (n - 1) + c_3 \cdot (n - 1) + c_4 \cdot \sum_{j=2}^n t_j + \\&\quad c_5 \cdot \sum_{j=2}^n (t_j - 1) + c_6 \cdot \sum_{j=2}^n (t_j - 1) + c_7 \cdot (n - 1) \\&= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\&\quad + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1)\end{aligned}$$

Observamos que a complexidade do algoritmo depende de n , dos t_i e dos c_i .

Exercício

Os c_i dependem da plataforma de execução, não de A .

1. Qual o menor valor possível para os t_i ? Corresponde a qual situação?
2. Qual o maior valor possível para os t_i ? Corresponde a qual situação?
3. Qual a complexidade do algoritmo nestas duas situações?



Exemplo: melhor caso

INSERTION-SORT(A)

1	for $j = 2$ to $\text{length}[A]$	$c_1 \cdot n$
2	do	
3	$\text{key} \leftarrow A[j]$	$c_2 \cdot (n - 1)$
4	$i \leftarrow j - 1$	$c_3 \cdot (n - 1)$
5	while $i > 0 \wedge A[i] > \text{key}$	$c_4 \cdot \sum_{j=2}^n t_j$
6	do	
7	$A[i + 1] \leftarrow A[i]$	$c_5 \cdot \sum_{j=2}^n (t_j - 1)$
8	$i \leftarrow i - 1$	$c_6 \cdot \sum_{j=2}^n (t_j - 1)$
9	$A[i + 1] \leftarrow \text{key}$	$c_7 \cdot (n - 1)$

- ▶ O teste do laço é avaliado uma vez apenas.
- ▶ $t_j = 1$ para $j = 2, \dots, n$

Exemplo: melhor caso

$$\sum_{j=2}^n 1 = n - 1 \text{ e } \sum_{j=2}^n 1 - 1 = 0.$$

$$\begin{aligned} T(n) = & (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ & + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \end{aligned}$$

Exemplo: melhor caso

$$\sum_{j=2}^n 1 = n - 1 \text{ e } \sum_{j=2}^n 1 - 1 = 0.$$

$$\begin{aligned} T(n) &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \\ &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot (n - 1) \end{aligned}$$

Exemplo: melhor caso

$$\sum_{j=2}^n 1 = n - 1 \text{ e } \sum_{j=2}^n 1 - 1 = 0.$$

$$\begin{aligned} T(n) &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \\ &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot (n - 1) \end{aligned}$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7) \cdot n - (c_2 + c_3 + c_4 + c_7)$$

Exemplo: pior caso

INSERTION-SORT(A)

1	for $j = 2$ to $\text{length}[A]$	$c_1 \cdot n$
2	do	
3	$\text{key} \leftarrow A[j]$	$c_2 \cdot (n - 1)$
4	$i \leftarrow j - 1$	$c_3 \cdot (n - 1)$
5	while $i > 0 \wedge A[i] > \text{key}$	$c_4 \cdot \sum_{j=2}^n t_j$
6	do	
7	$A[i + 1] \leftarrow A[i]$	$c_5 \cdot \sum_{j=2}^n (t_j - 1)$
8	$i \leftarrow i - 1$	$c_6 \cdot \sum_{j=2}^n (t_j - 1)$
9	$A[i + 1] \leftarrow \text{key}$	$c_7 \cdot (n - 1)$

- ▶ O teste do laço é avaliado j vezes.
- ▶ $t_j = j$ para $j = 2, \dots, n$

Exemplo: pior caso

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

$$\begin{aligned} T(n) = & (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ & + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \end{aligned}$$

Exemplo: pior caso

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

$$\begin{aligned} T(n) &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \\ &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad c_4 \cdot \left(\frac{n(n+1)}{2} - 1 \right) + (c_5 + c_6) \cdot \frac{n(n-1)}{2} \end{aligned}$$

Exemplo: pior caso

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

$$\begin{aligned} T(n) &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot \sum_{j=2}^n t_j + (c_5 + c_6) \cdot \sum_{j=2}^n (t_j - 1) \\ &= (c_1 + c_2 + c_3 + c_7) \cdot n - (c_2 + c_3 + c_7) \\ &\quad + c_4 \cdot \left(\frac{n(n+1)}{2} - 1 \right) + (c_5 + c_6) \cdot \frac{n(n-1)}{2} \end{aligned}$$

$$\begin{aligned} T(n) &= \frac{c_4 + c_5 + c_6}{2} \cdot n^2 + \\ &\quad \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7 \right) \cdot n \\ &\quad - \left(c_2 + c_3 + c_4 + \frac{c_5}{2} + \frac{c_6}{2} + c_7 \right) \end{aligned}$$

O conceito de operação básica

- ▶ Esta forma de análise é demasiadamente detalhista;
- ▶ Estes detalhes são inúteis, pois os c_i são fatores externos ao algoritmo.
- ▶ Ao invés disto, deve-se identificar a operação predominante na execução do algoritmo: a *operação básica*.
- ▶ A operação básica é aquela operação mais executada pelo algoritmo.
- ▶ Basta contar quantas vezes o algoritmo executa a operação básica.

Ponto chave

O arcabouço clássico de análise de complexidade de algoritmos é contar quantas vezes a operação básica é executada, em função do tamanho da entrada n .



Identificação da operação básica

Dica

Em algoritmos iterativos, é a operação que fica no laço mais aninhado.

Identificação da operação básica

Dica

Em algoritmos iterativos, é a operação que fica no laço mais aninhado.

INSERTION-SORT(A)

1	for $j = 2$ to $\text{length}[A]$	$c_1 \cdot n$
2	do	
3	$\text{key} \leftarrow A[j]$	$c_2 \cdot (n - 1)$
4	$i \leftarrow j - 1$	$c_3 \cdot (n - 1)$
5	while $i > 0 \wedge A[i] > \text{key}$	$c_4 \cdot \sum_{j=2}^n t_j$
6	do	
7	$A[i + 1] \leftarrow A[i]$	$c_5 \cdot \sum_{j=2}^n (t_j - 1)$
8	$i \leftarrow i - 1$	$c_6 \cdot \sum_{j=2}^n (t_j - 1)$
9	$A[i + 1] \leftarrow \text{key}$	$c_7 \cdot (n - 1)$

Em outras palavras: quantas comparações são necessárias para ordenar n elementos com o algoritmo de ordenação por inserção?

Exercício

Qual a operação básica do algoritmo de busca linear?

LINEAR-SEARCH(A, v)

```
1   $j \leftarrow 1$ 
2  while  $A[j] \neq v$  and  $j \leq \text{length}(A)$ 
3      do  $j \leftarrow j + 1$ 
4  if  $j \leq \text{length}(A)$ 
5      then return  $j$ 
6  else return NIL
```

Exercício

1. Considere o algoritmo de soma de duas matrizes $N \times M$. Qual a operação básica? Quantas vezes é executada?
2. Considere o algoritmo de multiplicação de uma matriz $N \times M$ por uma matriz $M \times P$. Qual a operação básica? Quantas vezes é executada?

Considerações

- ▶ Considere um algoritmo qualquer.
- ▶ Seja c o custo da execução da operação básica.
- ▶ Seja $C(n)$ o número de vezes que esta operação é executada pelo algoritmo.
- ▶ O tempo de execução do algoritmo $T(n)$, para uma entrada de tamanho n é tal que $T(n) \approx c \times C(n)$.
- ▶ Atenção $T(n)$ é *aproximado*:
 - ▶ c é aproximado,
 - ▶ as operações básicas não são contabilizadas.
 - ▶ é uma estimativa razoável menos no caso de n ser muito pequeno ou muito grande.

$$T(n) \approx c \times C(n).$$

- ▶ Quantas vezes mais rápido será executado o algoritmo em um computador 10 vezes mais rápido?
- ▶ Assumindo $C(n) = \frac{1}{2} \cdot n \cdot (n - 1)$: quantas vezes mais tempo levará a execução do algoritmo se multiplicarmos o tamanho da entrada por dois?

$$T(n) \approx c \times C(n).$$

- ▶ Quantas vezes mais rápido será executado o algoritmo em um computador 10 vezes mais rápido?

10

- ▶ Assumindo $C(n) = \frac{1}{2} \cdot n \cdot (n - 1)$: quantas vezes mais tempo levará a execução do algoritmo se multiplicarmos o tamanho da entrada por dois?

$$T(n) \approx c \times C(n).$$

- ▶ Quantas vezes mais rápido será executado o algoritmo em um computador 10 vezes mais rápido?

10

- ▶ Assumindo $C(n) = \frac{1}{2} \cdot n \cdot (n - 1)$: quantas vezes mais tempo levará a execução do algoritmo se multiplicarmos o tamanho da entrada por dois?

$$\begin{aligned} T(2n)/T(n) &= \frac{c \times C(2n)}{c \times C(n)} = \frac{C(2n)}{C(n)} \\ &= \frac{2n \cdot (2n - 1)}{n \cdot (n - 1)} = \frac{4n^2 - 2n}{n^2 - n} \\ &\approx 4 \text{ se } n \text{ for grande} \end{aligned}$$

Observações

Para responder a pergunta:

- ▶ Assumindo $C(n) = \frac{1}{2}.n.(n - 1)$: quantas vezes mais tempo levará a execução do algoritmo se multiplicarmos o tamanho da entrada por dois?

Observações

Para responder a pergunta:

- ▶ Assumindo $C(n) = \frac{1}{2}.n.(n - 1)$: quantas vezes mais tempo levará a execução do algoritmo se multiplicarmos o tamanho da entrada por dois?

Observe que:

1. Não precisamos saber o valor de c para responder: ele foi simplificado.
2. O fator multiplicativo $\frac{1}{2}$ também foi simplificado.
3. Em $C(n)$ apenas o monômio de maior coeficiente foi determinante para calcular o resultado (assumindo n é grande o suficiente).

Ponto chave

A análise de algoritmos desconsidera os fatores multiplicativos, e concentra-se no *crescimento assintótico* considerando entradas de grande tamanho.



Crescimento asintótico

- ▶ O custo do algoritmo para entradas pequenas é geralmente irrelevante.
- ▶ A diferença entre algoritmos se faz com entradas de grande tamanho.
- ▶ Para entradas de grande tamanho, o crescimento asintótico do custo de computação é o aspecto mais importante.

Valores (aproximados) de funções significativas

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3,3	10	$3,3 \times 10^1$	10^2	10^3	10^3	$3,6 \times 10^6$
10^2	6,6	10^2	$6,6 \times 10^2$	10^4	10^6	$1,3 \times 10^{30}$	$9,3 \times 10^{157}$
10^3	10	10^3	$1,0 \times 10^4$	10^6	10^9		
10^4	13	10^4	$1,3 \times 10^5$	10^6	10^9		
10^5	17	10^5	$1,7 \times 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2,0 \times 10^7$	10^{12}	10^{18}		

Valores (aproximados) de funções significativas

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3,3	10	$3,3 \times 10^1$	10^2	10^3	10^3	$3,6 \times 10^6$
10^2	6,6	10^2	$6,6 \times 10^2$	10^4	10^6	$1,3 \times 10^{30}$	$9,3 \times 10^{157}$
10^3	10	10^3	$1,0 \times 10^4$	10^6	10^9		
10^4	13	10^4	$1,3 \times 10^5$	10^8	10^{12}		
10^5	17	10^5	$1,7 \times 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2,0 \times 10^7$	10^{12}	10^{18}		

A função logaritmo é a que cresce mais devagar.

- ▶ Algoritmos de complexidade logarítmica tem custo imperceptível.
- ▶ A base do logaritmo é irrelevante: $\log_a n = \log_b n \times \log_a b$. As funções só são diferentes por uma constante multiplicativa.



Valores (aproximados) de funções significativas

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3,3	10	$3,3 \times 10^1$	10^2	10^3	10^3	$3,6 \times 10^6$
10^2	6,6	10^2	$6,6 \times 10^2$	10^4	10^6	$1,3 \times 10^{30}$	$9,3 \times 10^{157}$
10^3	10	10^3	$1,0 \times 10^4$	10^6	10^9		
10^4	13	10^4	$1,3 \times 10^5$	10^8	10^{12}		
10^5	17	10^5	$1,7 \times 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2,0 \times 10^7$	10^{12}	10^{18}		

As funções 2^n e $n!$ tem crescimento muito rápido.

- ▶ Para qualquer entrada não pequena, o valor é astronômico;
- ▶ O tempo de execução para entradas grandes excede a idade estimada do universo ∞ .
- ▶ Não são práticos para entradas que não sejam pequenas.

Como reagem essas funções quando n é duplicado? quadruplicado?

- ▶ $\log_2 n$
- ▶ n
- ▶ $n \log_2 n$
- ▶ n^2
- ▶ n^3
- ▶ 2^n
- ▶ $n!$

Exercício

Para cada par de funções, indique se a primeira tem maior crescimento assintótico que a segunda, menor crescimento assintótico, ou se os crescimentos assintóticos são iguais:

- ▶ $n.(n + 1)$ e $2000.n^2$;
- ▶ $\log_2 n$ e $\ln n$;
- ▶ 2^{n-1} e 2^n ;
- ▶ $100.n^2$ e $0,01.n^3$;
- ▶ $\log_2^2 n$ e $\log_2 n^2$;
- ▶ $(n - 1)!$ e $n!$.

Exemplo: crescimento assintótico

$$\begin{aligned} T(n) = & \frac{c_4+c_5+c_6}{2}.n^2 + \\ & (c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7).n \\ & - (c_2 + c_3 + c_4 + \frac{c_5}{2} + \frac{c_6}{2} + c_7) \end{aligned}$$

$$T(n) = a.n^2 + b.n + c$$

onde a, b, c são constantes que dependem da plataforma de execução.

Quando n é grande, o fator predominante é $a.n^2$.

Diz se que

- ▶ $T(n) \in \Theta(n^2)$.
- ▶ O crescimento assintótico do tempo de execução é n^2 .
- ▶ O algoritmo é *quadrático*.

- ▶ tamanho da(s) entrada(s)
- ▶ operação básica
- ▶ crescimento assintótico
- ▶ funções tipicamente encontradas

logarítmica $\log n$,

linear n ,

quase-linear $n \log n$,

quadrática n^2 ,

polinomial n^k , onde $k > 1$

exponencial k^n , onde $k > 1$

fatorial $n!$