ANÁLISE DE ALGORITMOS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil

Agenda

Análise de algoritmos

Principais perspectivas (não-subjetivas):

- Eficiência temporal (complexidade temporal)
- Eficiência espacial (complexidade espacial)

Eficiência: uma função a partir do tamanho da entrada

- Entradas maiores $\Rightarrow$ maior tempo de execução

Exemplos:

- Ordenar uma lista: tamanho da lista
- Multiplicação de matrizes: tamanho das matrizes | qtd. elementos
- Corretor ortográfico: qtd. caracteres | qtd. palavras

Métrica de medição

Medição baseada em tempo

- Infraestrutura computacional
- Qualidade do compilador

$C(n)$ = qtd. de execução da operação básica

- Ordenação: comparação valores
- Multiplicação de matrizes: multiplicação

Seja c_{op} o tempo de execução da operação básica

- $T(n) \approx c_{op} C(n)$

Métrica de medição

Exemplo:

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2$$

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} = \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4$$

Ordem de crescimento

Ordem de crescimento

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

[1]

Considerando: 10^{12} (1 trilhão) operações por segundo

- 2^{100} operações $\approx 4 \cdot 10^{10}$ anos

[1] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Casos: pior, melhor, médio

Algoritmo: SequentialSearch(A[0..n-1], K)

```
1  i ← 0;
2  while i < n ∧ A[i] ≠ K do
3  |    i ← i + 1;
4  if i < n then return i;
5  else return −1;
```

Operação básica: quantidade de verificações $A[i] \neq K$:

- $C_{worst}(n) = n$
- $C_{best}(n) = 1$

Procurar outro algoritmo se $C_{best}(n)$ não for aceitável

Casos: pior, melhor, médio

Assumindo:

- $p(0 \leq p \leq 1)$ – probabilidade de encontrar valor
- probabilidade igual para todo i

$$
\begin{aligned}
C_{avg}(n) &= (1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdot + n \cdot \frac{p}{n}) + n \cdot (1 - p) \\
&= \frac{p}{n}(1 + 2 + \cdots + i + \cdots + n) + n(1 - p) \\
&= \frac{p}{n}\frac{n(n+1)}{2} + n(1 - p) \\
&= \frac{p(n+1)}{2} + n(1 - p)
\end{aligned}
$$

Casos: pior, melhor, médio

Assumindo:

- $p (0 \leq p \leq 1)$ – probabilidade de encontrar valor
- probabilidade igual para todo i

$$
\begin{aligned}
C_{avg}(n) &= (1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdot + n \cdot \frac{p}{n}) + n \cdot (1 - p) \\
&= \frac{p}{n}(1 + 2 + \cdots + i + \cdots + n) + n(1 - p) \\
&= \frac{p}{n}\frac{n(n+1)}{2} + n(1 - p) \\
&= \frac{p(n+1)}{2} + n(1 - p)
\end{aligned}
$$

Calcular $C_{avg}(n)$ é normalmente mais difícil

- Em alguns algoritmos, $C_{avg} \ll C_{worst}$

Casos: pior, melhor, médio

Assumindo:

- $p (0 \leq p \leq 1)$ – probabilidade de encontrar valor
- probabilidade igual para todo i

$$
\begin{aligned}
C_{avg}(n) &= (1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdot + n \cdot \frac{p}{n}) + n \cdot (1 - p) \\
&= \frac{p}{n}(1 + 2 + \cdots + i + \cdots + n) + n(1 - p) \\
&= \frac{p}{n}\frac{n(n+1)}{2} + n(1 - p) \\
&= \frac{p(n+1)}{2} + n(1 - p)
\end{aligned}
$$

Calcular $C_{avg}(n)$ é normalmente mais difícil

- Em alguns algoritmos, $C_{avg} \ll C_{worst}$

Outra possibilidade de análise: eficiência amortizada

Agenda

Análise assintótica

Parar comparar odens de crescimento:
O (big oh), Ω (big omega), Θ (big theta)

Considerando:

- $t(n)$ e $g(n)$ funções não-negativas definidas sobre $\mathbb{N}$
- $t(n)$ = tempo de execução do algoritmo (i.e., $C(n)$)
- $g(n)$ = função para comparar eficiência

Informalmente:

- $O(g(n))$ = funções com ordem de crescimento $\leq$ a $g(n)$
- $\Omega(g(n))$ = funções com ordem de crescimento $\geq$ a $g(n)$
- $\Theta(g(n))$ = funções com ordem de crescimento $=$ a $g(n)$

Análise assintótica

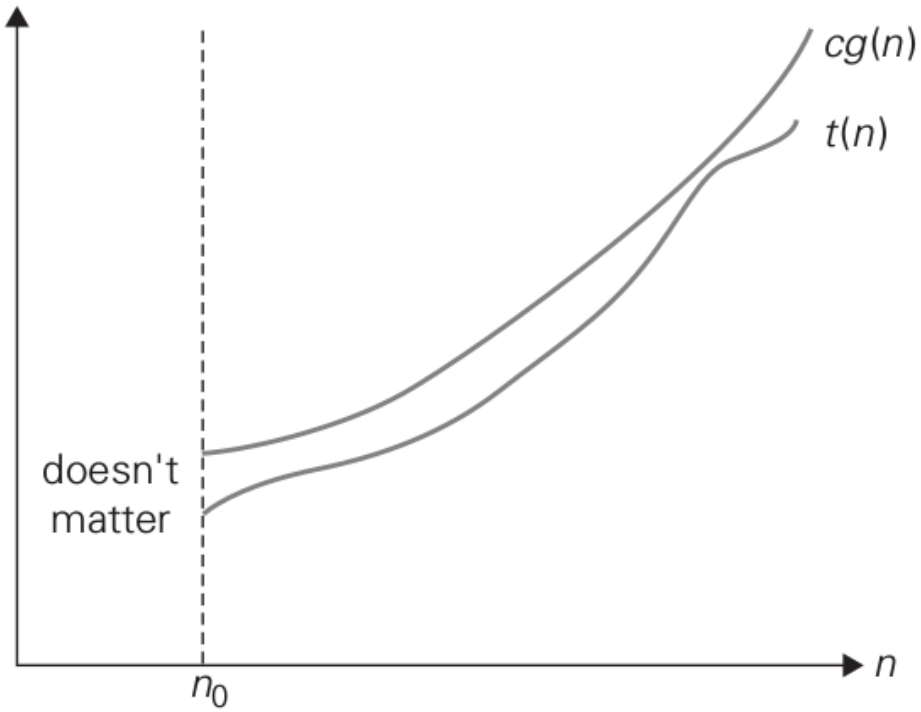
Exemplos (considerando $n \to \infty$):

- $n \in O(n^2)$
- $100n + 5 \in O(n^2)$
- $\frac{1}{2}n(n-1) \in O(n^2)$
- $n^3 \notin O(n^2)$
- $n^3 \in \Omega(n^2)$
- $\frac{1}{2}n(n-1) \in \Omega(n^2)$
- $100n + 5 \notin \Omega(n^2)$
- $100n + 5 \notin \Theta(n^2)$
- $100n + 5 \in \Theta(n)$

Análise assintótica[2]

Definição $O(n)$:

- $t(n) \in O(g(n))$ se existem c positivo e $n_0 \geq 0$ tal que $t(n) \leq cg(n)$ para todo $n \geq n_0$



$o(n)$ = para todo c

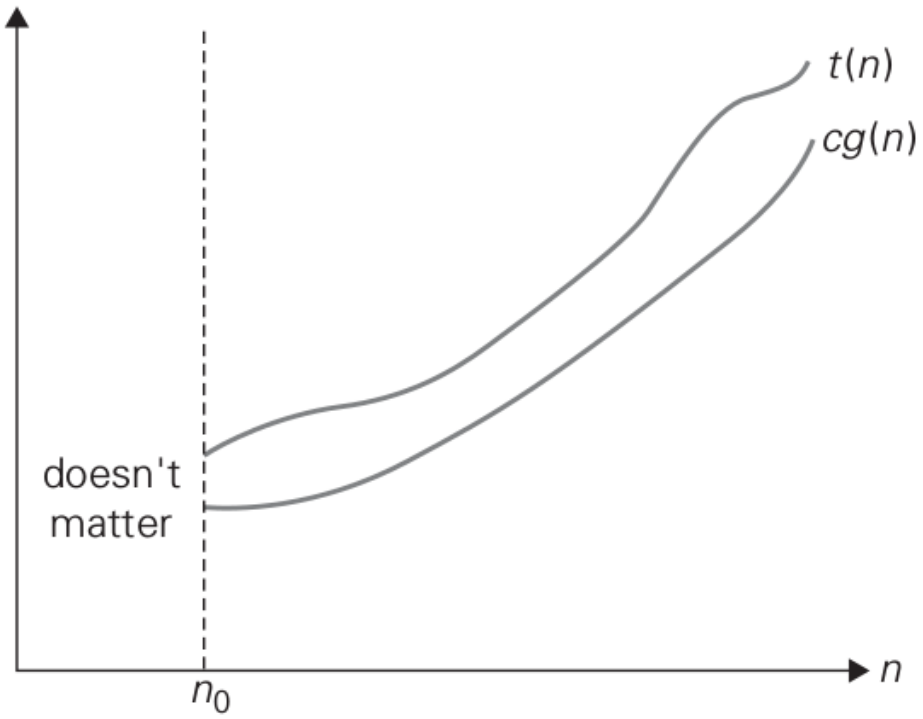
Exemplo:
$100n + 5 \leq 100n + 5n$
(para todo $n \geq 1$) $= 105n$
Logo, $100n + 5 \in O(n)$

[2] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Análise assintótica[3]

Definição $\Omega(n)$:

- $t(n) \in \Omega(g(n))$ se existem c positivo e $n_0 \geq 0$ tal que $t(n) \geq cg(n)$ para todo $n \geq n_0$



$\omega(n)$ = para todo c

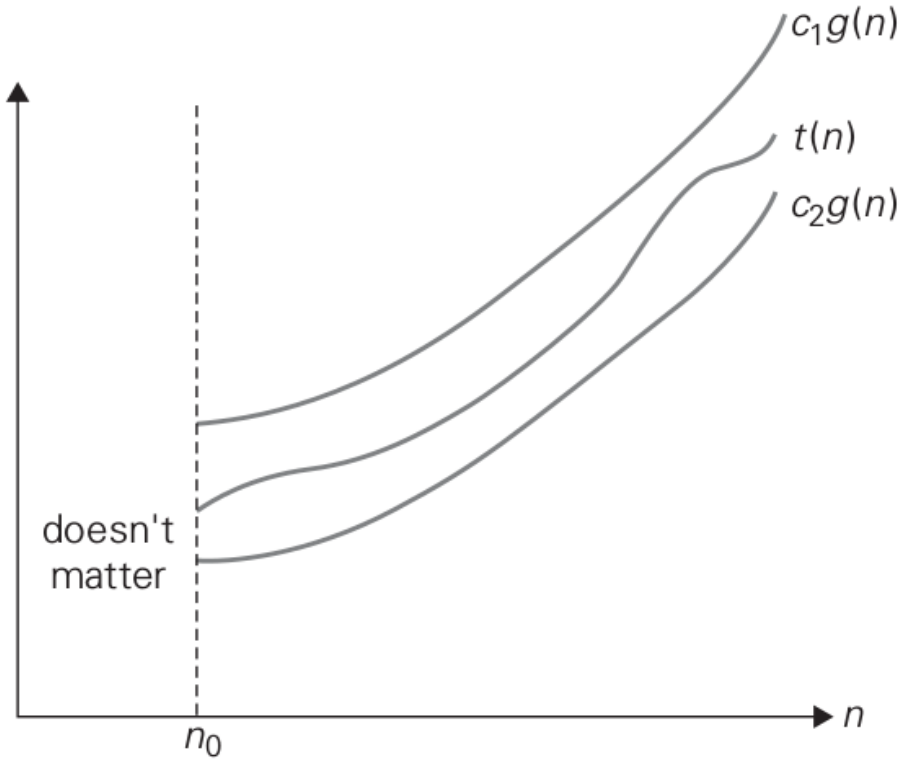
Exemplo:
$n^3 \geq n^2$ (para todo $n \geq 0$)
Logo, $n^3 \in \Omega(n^2)$

[3] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Análise assintótica[4]

Definição $\Theta(n)$:

- $t(n) \in \Theta(g(n))$ se existem c_1 e c_2 positivos e $n_0 \geq 0$ tal que $c_2 g(n) \leq t(n) \leq c_1 g(n)$ para todo $n \geq n_0$



Exemplo:
$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2$
(para todo $n \geq 0$)

$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq$
$\frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n = \frac{1}{4}n^2$
(para todo $n \geq 2$)

Logo, $\frac{1}{2}n(n-1) \in \Theta(n^2)$

[4] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Análise assintótica

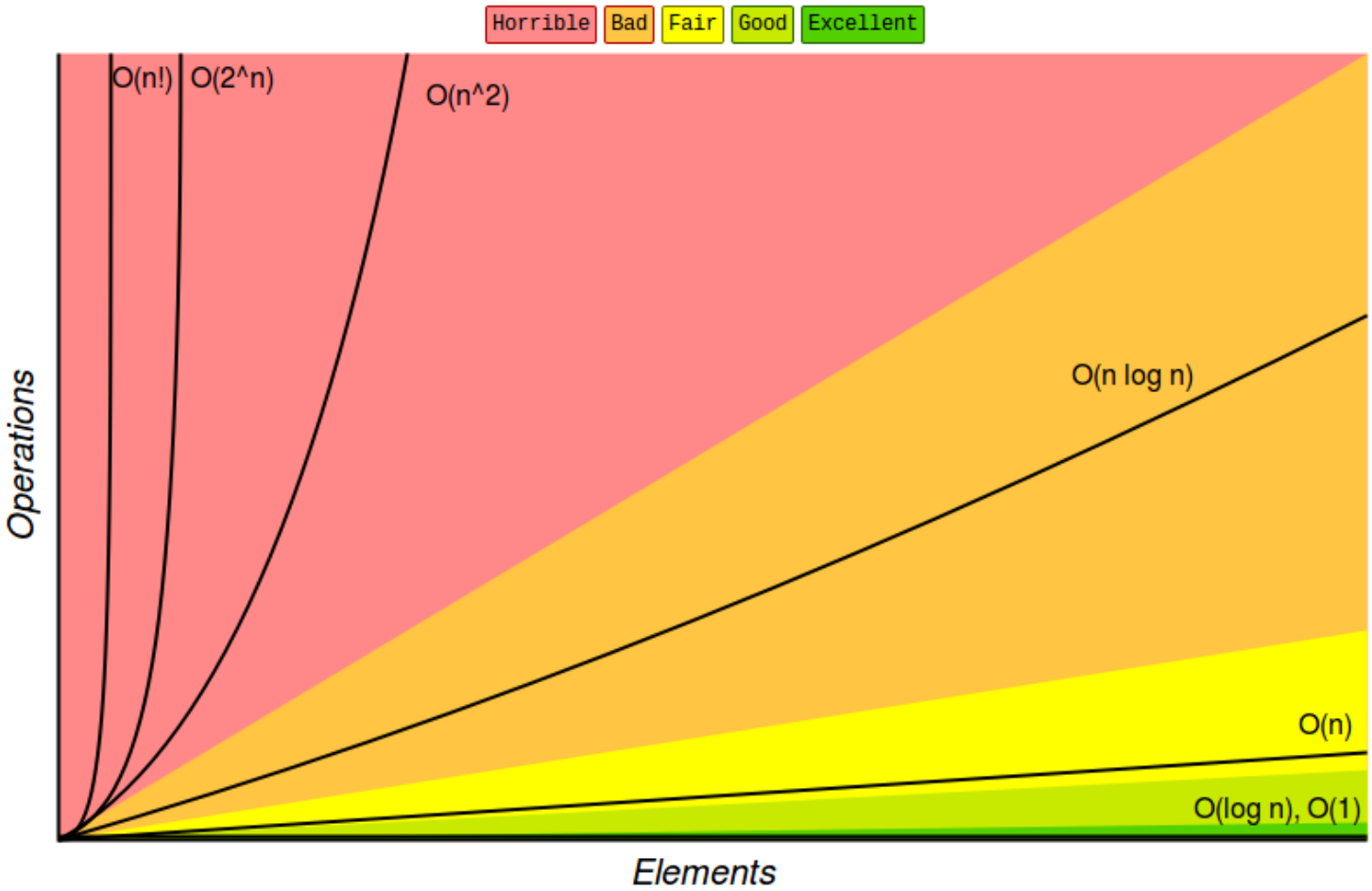
Teorema

- se $t_1(n) \in O(g_1(n)) \wedge t_2(n) \in O(g_2(n))$,
  então $t_1(n) + t_2(n) \in O(max(g_1(n), g_2(n)))$

Implicação prática:

- A eficiência de um algoritmo é determinada pela parte com maior ordem de crescimento.

Análise assintótica[5]



5 http://bigocheatsheet.com/

Análise assintótica[6]

Data Structure	Time Complexity								Space Complexit
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	O(1)	O(n)	O(n)	O(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	O(n)	O(n)	O(1)	O(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	O(1)	O(1)	O(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	O(log(n))	O(log(n))	O(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

6 http://bigocheatsheet.com/

Análise assintótica: comparando via limite

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & \text{ordem de } t(n) \text{ é menor do que ordem de } g(n) \\ c & \text{ordem de } t(n) \text{ é igual a ordem de } g(n) \\ \infty & \text{ordem de } t(n) \text{ é maior do que ordem de } g(n) \end{cases}$$

Exemplo:

$$\lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2} \lim_{n\to\infty} \frac{n^2-n}{n^2} = \frac{1}{2} \lim_{n\to\infty} (1 - \frac{1}{n}) = \frac{1}{2}$$

logo, $O(\frac{1}{2}n(n-1)) = O(n^2)$.

Análise assintótica: principais classes

1	constante
$log\ n$	logarítmica
n	linear
$n\ log\ n$	linear-rítmica
n^2	quadrática
n^3	cúbica
2^n	exponencial
$n!$	fatorial

Agenda

Funções não-recursivas

Procedimento geral:

1. Decidir parâmetro(s) indicando tamanho da(s) entrada(s)
2. Identificar operação básica do algoritmo
3. Verificar se $C(n)$ depende só de n
   - Análise do pior caso
   - Análise do melhor caso
   - Análise do caso médio
4. Definir somatário para $C(n)$
5. Estabelecer ordem de crescimento

Exemplo

Algoritmo: UniqueElements(A[0..n-1])

```
1  for i ← 0 to n − 2 do
2      for j ← i + 1 to n − 1 do
3          if A[i] = A[j] then return false;
4  return true;
```

Parâmetro do tamanho da entrada: n

Operação básica: comparação C

$C(n)$ não depende só de n

Exemplo

$$
\begin{aligned}
C_{worst}(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) \\
&= \sum_{i=0}^{n-2} (n-1-i) \\
&= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \\
&= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} \\
&= \frac{(n-1)n}{2} \\
&\approx \frac{1}{2} n^2 \in \Theta(n^2)
\end{aligned}
$$

Exemplo

$$
\begin{aligned}
C_{worst}(n) &= \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-2} ((n-1) - (i+1) + 1) \\
&= \sum_{i=0}^{n-2} (n-1-i) \\
&= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \\
&= (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2} \\
&= (n-1)^2 - \frac{(n-2)(n-1)}{2} \\
&= \frac{(n-1)n}{2} \\
&\approx \frac{1}{2} n^2 \in \Theta(n^2)
\end{aligned}
$$

Nem sempre dá para definir $C(n)$ como um somatório

Agenda

Funções recursivas

Procedimento geral:

1. Decidir parâmetro(s) indicando tamanho da(s) entrada(s)
2. Identificar operação básica do algoritmo
3. Verificar se $C(n)$ depende só de n
   - Análise do pior caso
   - Análise do melhor caso
   - Análise do caso médio
4. Definir relação de recorrência e condições iniciais para $C(n)$
5. Estabelecer ordem de crescimento

Exemplo

Algoritmo: F(n)

1 **if** $n = 0$ **then return** *1*;
2 **else return** $F(n-1) * n$;

Parâmetro do tamanho da entrada: n

Operação básica: multiplicação M

- Assumindo que M é constante para quaisquer dois números
- Assumindo que a multiplicação "domina" a adição

$M(n)$ depende só de n

Relação de recorrência:

- $M(n) = M(n-1) + 1$, para $n > 0$
- $M(0) = 0$

Exemplo: método de backward substitutions

$$
\begin{aligned}
M(n) &= M(n-1)+1 \\
&= (M(n-2)+1)+1 = M(n-2)+2 \\
&= (M(n-3)+1)+2 = M(n-3)+3 \\
&= \cdots \\
&= M(n-i)+i \\
&= \cdots \\
&= M(n-n)+n \\
&= M(0)+n \\
&= 0+n \\
&= n \in \Theta(n)
\end{aligned}
$$

Agenda

Análise amortizada

Amortizar (diluir) o custo no tempo (sequência de chamadas)

- Nem toda execução de uma operação terá o mesmo custo
- Calcular $C_{amort}(n) = \frac{C(n)}{n}$

Exemplo: *append* em arrays dinâmicos

- Se o array não está cheio, inserir valor no final
- Se o array está cheio,
  - Criar um array com o dobro da capacidade
  - Copiar os valores antigos
  - Inserir valor no final

Análise amortizada

Exemplo: *append* em arrays dinâmicos

append	tam. antigo	tam. atual	qtd. cópias
1	1	-	-
2	1	2	1
3	2	4	2
4	4	-	-
5	4	8	4
6	8	-	-
7	8	-	-
8	8	-	-
9	8	16	8

Análise amortizada

Exemplo: *append* em arrays dinâmicos

Quantidade de operações de cópia para 9 inserções:

- $1 + 2 + 4 + 8$

De forma geral, $2^n + 1$ inserções:

- $1 + 2 + 2^2 + 2^3 + \cdots + 2^n = 2^{n+1} - 1$

Custo total (inserções + cópias) para $2^n + 1$ (método da agregação):

- $(2^{n+1} - 1) + (2^n + 1) = 3 \cdot 2^n$

Amortizando (para $n \to \infty$):

- $\lim_{n\to\infty} \frac{3 \cdot 2^n}{2^n+1} = 3$, logo $C_{amort}(n) \in O(1)$

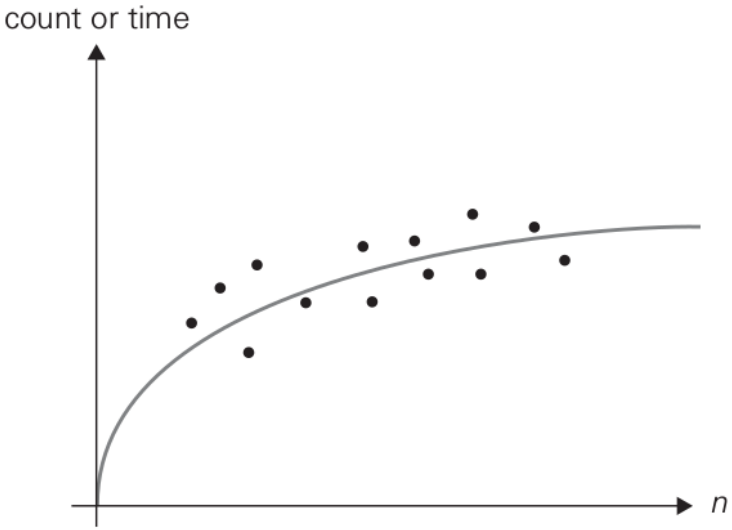
Agenda

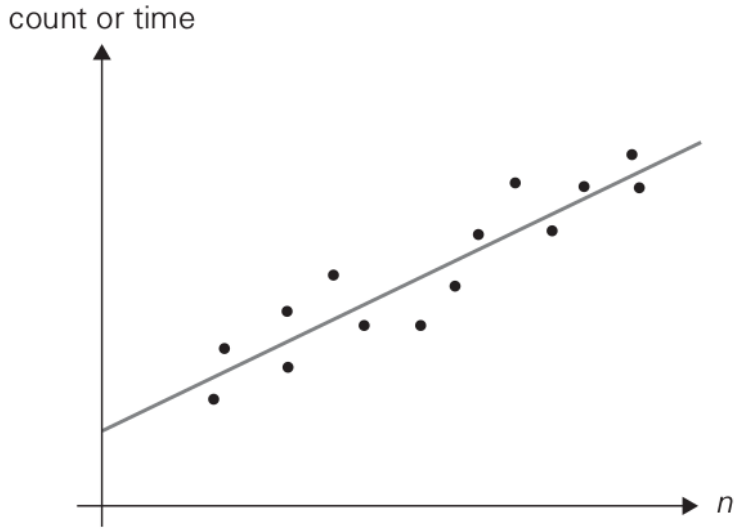
Análise empírica

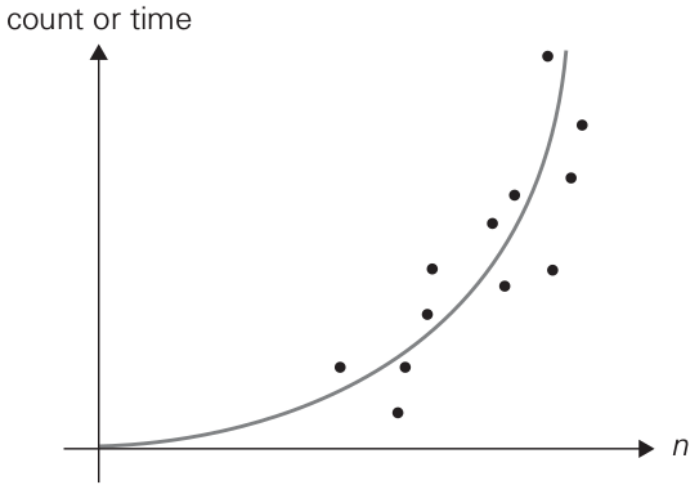
Procedimento geral:

1. Entender o propósito do experimento
2. Definir métrica de medição (contagem *vs.* tempo)
3. Definir as características das entradas
4. Implementar o algoritmo
5. Gerar entradas
6. Executar o algoritmo e coletar dados
7. Analisar os dados obtidos

Análise empírica







[7] Fonte: A. Levitin. Introduction to the Design and Analysis of Algorithms. 2011.

Análise empírica

Análise assintótica

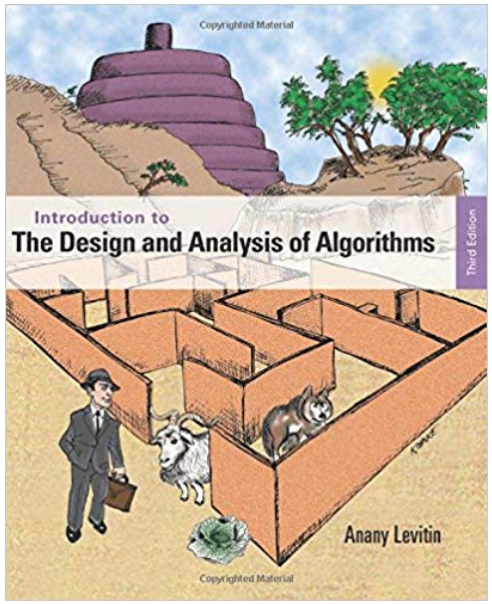
- + Independência de entradas específicas
- − Aplicabilidade limitada

Análise empírica

- + Aplicabilidade ilimitada
- − Dependência das entradas
- − Dependência da infraestrutura computacional

Agenda

Bibliografia + leitura recomendada



Capítulo 2 (pp. 41–95).
Anany Levitin.
Introduction to the Design and Analysis of Algorithms.
3a edição. Pearson. 2011.

ANÁLISE DE ALGORITMOS

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil