

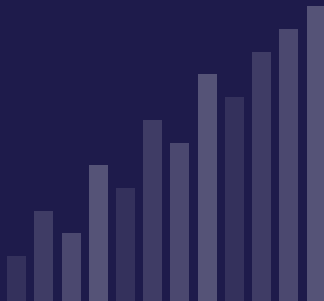
AULA 5 · BLOCO 1: FUNDAMENTOS, ORDENAÇÃO E CUSTO

Notação assintótica: O , Ω e

Θ

Capítulo 3

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Definir Θ , O , Ω , o e ω e interpretá-las graficamente
- ▶ Provar relações assintóticas exibindo constantes c e n_0
- ▶ Ordenar funções comuns pela taxa de crescimento
- ▶ Estimar a ordem de crescimento a partir de medições (log-log e razão de dobra)



Teoria

Definições, gráficos da Figura 3.1, propriedades e funções comuns



Prática em C

Quanto cabe em 1 segundo para cada $f(n)$



Experimento

Quatro algoritmos, quatro ordens de crescimento num só gráfico



Conexões

Ciência de dados, complexidade, bioinformática e SLAs

As três notações principais

$\Theta(g) = \{f : \exists c_1, c_2, n_0 > 0, 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \forall n \geq n_0\}$ $O(g)$: só o limite superior $f \leq cg$. $\Omega(g)$: só o limite inferior $f \geq cg$.

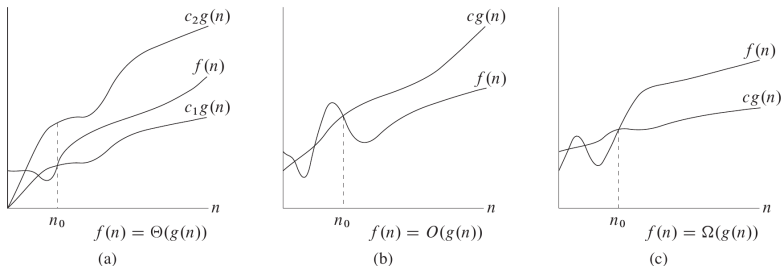


Figura 3.1 de Cormen et al. (2009) — (a) Θ : f entre c_1g e c_2g . (b) O : f abaixo de cg . (c) Ω : f acima de cg . Sempre a partir de n_0 .

Provando com constantes

Exemplo: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$.

Precisamos de $c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$. Dividindo por n^2 : $c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$.

- ▶ Lado direito: vale para todo $n \geq 1$ com $c_2 = \frac{1}{2}$.
- ▶ Lado esquerdo: para $n \geq 7$, $\frac{1}{2} - \frac{3}{7} = \frac{1}{14}$; basta $c_1 = \frac{1}{14}$.
- ▶ Logo $c_1 = \frac{1}{14}$, $c_2 = \frac{1}{2}$, $n_0 = 7$.

Contraexemplo: $6n^3 \neq \Theta(n^2)$: precisaríamos de $6n^3 \leq c_2 n^2$, ou seja, $n \leq c_2/6$, o que falha para n grande.

Teorema 3.1: $f = \Theta(g)$ se e somente se $f = O(g)$ e $f = \Omega(g)$.

Notação como ferramenta

- ▶ O no **pior caso** limita *toda* entrada: "a inserção é $O(n^2)$ ".
- ▶ Ω no **melhor caso** limita toda entrada por baixo: "a inserção é $\Omega(n)$ ".
- ▶ "A inserção é $\Theta(n^2)$ " é falso para toda entrada; vale para o **pior caso**.
- ▶ o e ω são limites **estritos**: $2n = o(n^2)$, mas $2n^2 \neq o(n^2)$. Formalmente, $f = o(g) \iff \lim f/g = 0$.
- ▶ Em equações: $2n^2 + \Theta(n) = \Theta(n^2)$ significa "alguma função anônima de $\Theta(n)$ ".

Propriedades: transitividade, reflexividade, simetria (Θ) e simetria transposta ($f = O(g) \iff g = \Omega(f)$).

A escada das funções comuns

$$1 \prec \lg^* n \prec \lg \lg n \prec \lg n \prec \sqrt{n} \prec n \prec n \lg n \prec n^2 \prec n^3 \prec 2^n \prec n! \prec n^n$$

- ▶ Todo polinômio cresce mais devagar que toda exponencial: $n^b = o(a^n)$ para $a > 1$.
- ▶ Todo polilogaritmo cresce mais devagar que todo polinômio: $\lg^b n = o(n^a)$ para $a > 0$.
- ▶ A base do logaritmo não importa: $\log_a n = \Theta(\log_b n)$.
- ▶ Stirling: $\lg(n!) = \Theta(n \lg n)$, resultado que voltará no limite inferior da ordenação.



Prática em C

- ▶ Para 10^6 operações por segundo: qual o maior n para cada $f(n)$?
- ▶ Busca binária sobre o valor de n (a função é crescente)
- ▶ `tgamma(n+1)` calcula $n!$ para n real

Quanto cabe em 1 segundo (1/3)

```
1  #include <stdio.h>
2  #include <math.h>
3
4  /* Supondo `ops` operações por segundo, qual o maior n resolvido em 1 s para cada f(n)?
5     Busca binária sobre n: f é crescente, basta achar o último n com f(n) <= ops. */
6  double f(int qual, double n) {
7      switch (qual) {
8          case 0: return log2(n);
9          case 1: return sqrt(n);
10         case 2: return n;
11         case 3: return n * log2(n);
12         case 4: return n * n;
```

Quanto cabe em 1 segundo (2/3)

```
13     case 5: return n * n * n;
14     case 6: return pow(2, n);
15     default: return tgamma(n + 1);           //  $n! = \Gamma(n + 1)$ 
16 }
17 }
18
19 int main(void) {
20     double ops;
21     if (scanf("%lf", &ops) != 1 || ops < 1) return 1;
22     const char *nome[8] = {"lg n", "raiz de n", "n", "n lg n",
23                             "n^2", "n^3", "2^n", "n!"};
24     printf("%-10s maior n em 1 s com %.0e operações/s\n", "f(n)", ops);
```

Quanto cabe em 1 segundo (3/3)

```
25  [
26      for (int q = 0; q < 8; q++) {
27          double lo = 1, hi = 2;
28          while (hi < 1e300 && f(q, hi) <= ops) hi *= 2;    // dobra até passar
29          for (int it = 0; it < 200; it++) {                // depois, busca binária
30              double meio = (lo + hi) / 2;
31              if (f(q, meio) <= ops) lo = meio; else hi = meio;
32          }
33          if (lo > 1e15) printf("%-10s  %.2e\n", nome[q], lo);
34          else printf("%-10s  %.0f\n", nome[q], floor(lo));
35      }
36      return 0;
37  }
```



Experimento

- ▶ Busca binária, busca linear, merge sort e todos os pares
- ▶ n de 256 a 32.768, dobrando
- ▶ Em escala log-log, cada ordem é uma reta com inclinação própria

Descobrir a ordem de crescimento medindo

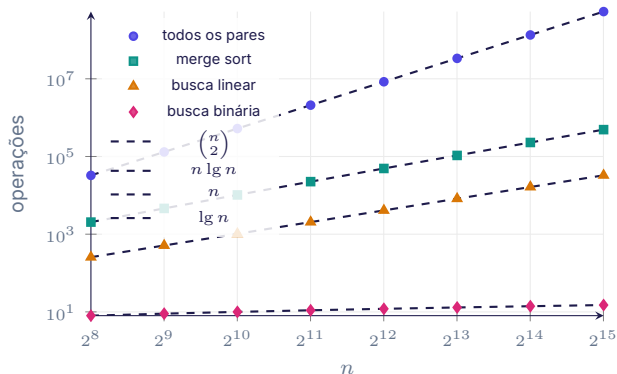
Hipótese. Contando a operação básica de cada algoritmo: busca binária $\approx \lg n$; busca linear (pior caso) $= n$; merge sort $= n \lg n$ (uma cópia por posição e nível); todos os pares $= n(n-1)/2$. Em log-log, as inclinações devem ser $\approx 0, 1$, um pouco acima de 1 e 2.

Método

1. Contador global ops na operação básica de cada algoritmo
2. $n = 2^8, \dots, 2^{15}$
3. Busca de um valor ausente (pior caso)
4. Comparação com a fórmula exata de cada um

EXPERIMENTO

Quatro ordens de crescimento em log-log



n	medido	razão
256,0	32.640	1,000
512,0	130.816	1,000
1.024	523.776	1,000
2.048	2.096.128	1,000
4.096	8.386.560	1,000
32.768	536.854.528	1,000

razão = medido $\div$ $\binom{n}{2}$

✓ desvio máx. 0,0%

Retas paralelas têm a mesma ordem; inclinação maior, ordem maior. A busca binária é quase horizontal.

EXPERIMENTO

O que o experimento mostrou

As contagens batem **exatamente** com $n(n-1)/2$, $n \lg n$, n e $\lg n$. Em escala log-log, a ordem de crescimento aparece como inclinação. Para estimar k a partir de duas medições: $k \approx \lg(T(2n)/T(n))$, a razão de dobra da aula 1.

>_ Reproduza: `gcc -O2 aula05_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Ciência de dados
- ▶ Teoria da complexidade
- ▶ Bioinformática
- ▶ Contratos de serviço (SLA)

Onde isso aparece na computação

Ciência de dados

O k -means custa $O(n k d i)$; um algoritmo $O(n^2)$ inviabiliza bases com milhões de linhas. A notação assintótica decide o que escala.

Teoria da complexidade

P contra NP pergunta se problemas verificáveis em tempo polinomial também são resolvíveis em tempo polinomial: um dos sete problemas do milênio.

Bioinformática

Alinhar duas sequências custa $\Theta(mn)$: para genomas de bilhões de bases, isso força heurísticas como BLAST.

Contratos de serviço (SLA)

Garantir resposta em 100 ms para qualquer carga exige saber como o tempo cresce com os dados; a ordem de crescimento entra no planejamento de capacidade.

Para casa

Exercícios

1. (3.1-1) Prove que $\max(f(n), g(n)) = \Theta(f(n) + g(n))$.
2. (3.1-2) Prove que $(n + a)^b = \Theta(n^b)$ para constantes a e $b > 0$.
3. (3.1-4) $2^{n+1} = O(2^n)$? E $2^{2n} = O(2^n)$?
4. (Problema 3-3) Ordene 10 funções da lista do livro por taxa de crescimento.
5. Rode `aula05_crescimento.c` com 10^9 operações por segundo e compare com 10^6 .

Leitura

Cormen et al., seções 3.1 e 3.2.

Código

josedihego.net/estruturas