Projeto e Análise de Algoritmos

A. G. Silva, R. de Santiago

Baseado nos materiais de
Souza, Silva, Lee, Rezende, Miyazawa – Unicamp
Ribeiro – FCUP • Mariani – UFSC
Manber, Introduction to Algorithms (1989) – Livro

29 de março de 2019

Conteúdo programático

- Introdução (4 horas/aula)
- Notação Assintótica e Crescimento de Funções (4 horas/aula)
- Recorrências (4 horas/aula)
- Divisão e Conquista (12 horas/aula)
- Buscas (4 horas/aula)
- Grafos (4 horas/aula)
- Algoritmos Gulosos (8 horas aula)
- Programação Dinâmica (8 horas/aula)
- NP-Completude e Reduções (6 horas/aula)
- Algoritmos Aproximados e Busca Heurística (6 horas/aula)

Cronograma atualizado em 22mar

- **15mar** – Apresentação da disciplina. Introdução.
- **22mar** – *Prova de proficiência/validação.*
- **29mar** – Notação assintótica. Recorrências.
- **05abr** – Divisão e conquista. Multiplicação de inteiros.
- **12abr** – Ordenação sem restrição. Ordenação em tempo linear.
- **19abr** – *Dia não letivo.* Exercícios.
- **26abr** – Estatística de ordem.
- **03mai** – **Primeira avaliação.**
- **10mai** – Grafos. Buscas.
- **17mai** – Algoritmos gulosos.
- **24mai** – Algoritmos gulosos. Programação dinâmica.
- **31mai** – Programação dinâmica.
- **07jun** – NP-Completude e reduções.
- **14jun** – **Segunda avaliação.**
- **21jun** – *Dia não letivo.* Exercícios.
- **28jun** – **Avaliação substitutiva** *(opcional)*

Algoritmo

- Um algoritmo é um **método** para resolver um problema (computacional)
- Um algoritmo é uma **ideia** por trás de um programa e é independente de linguagem de programação, máquina, etc
- **Propriedades** de um algoritmo:

Correção

Deve resolver corretamente **todas as instâncias** do problema

Eficiência

O desempenho (**tempo** e **memória**) deve ser adequado

- Este curso é sobre a **concepção** e **análise** de algoritmos corretos e eficientes

Preocupações

- Importância da análise do tempo de execução

Predição

Quanto tempo um algoritmo precisa para resolver um problema? Qual a escala? Podemos ter garantias sobre o tempo de funcionamento?

Comparação

Um algoritmo A é melhor que um algoritmo B? Qual é a melhor forma de resolvermos um determinado problema?

- Estudaremos uma **metodologia** para responder a essas questões

Velocidade de computadores

Desempenho algorítmico $\times$ Velocidade de computação

Um algoritmo melhor em um computador mais lento **sempre vencerá** um algoritmo pior em um computador mais rápido, para instâncias suficientemente grandes

- O que realmente importa é a **taxa de crescimento** do tempo de execução!

Random Access Machine (RAM)

- Precisamos de um **modelo genérico** e **independente** de linguagem e de máquina.
- *Random Access Machine* (RAM)
  - Cada **operação simples** (ex.: $+$, $-$, $\leftarrow$, **If**) leva **1 passo**
  - Ciclos e procedimentos, por exemplo, não são instruções simples
  - Cada **acesso à memória** leva também **1 passo**
- Podemos medir o tempo de execução contando o número de passos como uma função do tamanho de entrada: $T(n)$
- Operações são **simplificadas**, mas isto é útil
  Ex.: a soma de dois inteiros não custa o mesmo que dividir dois reais mas, para uma visão global, esses valores específicos não são importantes

Tipos de análise de algoritmos

Pior caso (análise mais comum de ser feita):

- $T(n)$ = quantidade máxima de tempo para qualquer entrada de tamanho n

Caso médio (análise feita de vez em quando):

- $T(n)$ = tempo médio para qualquer entrada de tamanho n
- Implica em conhecimento sobre a distribuição estatística das entradas

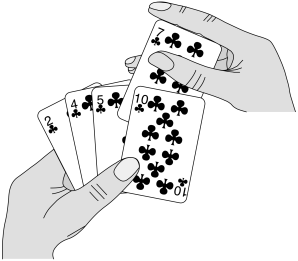
Melhor caso (apenas uma curiosidade):

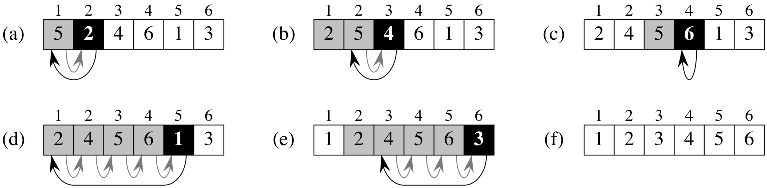
- Quando o algoritmo é rápido apenas para algumas das entradas

Ordenação por inserção *(revisão)*

Ordenação por inserção

revisão





Vamos contar ?

ORDENA-POR-INSERÇÃO(A, n)	Custo	Vezes
1 **para** $j \leftarrow 2$ **até** n **faça**	c_1	n
2 *chave* $\leftarrow A[j]$	c_2	$n-1$
3 $\triangleright$ Insere $A[j]$ em $A[1 \ldots j-1]$	0	$n-1$
4 $i \leftarrow j-1$	c_4	$n-1$
5 **enquanto** $i \geq 1$ **e** $A[i] >$ *chave* **faça**	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i \leftarrow i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] \leftarrow$ *chave*	c_8	$n-1$

A constante c_k representa o custo (tempo) de cada execução da linha k.

Denote por t_j o número de vezes que o teste no laço **enquanto** na linha 5 é feito para aquele valor de j.

Tempo de execução total

Logo, o tempo total de execução $T(n)$ de Ordena-Por-Inserção é a soma dos tempos de execução de cada uma das linhas do algoritmo, ou seja:

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

Como se vê, entradas de tamanho igual (i.e., mesmo valor de n), podem apresentar tempos de execução diferentes já que o valor de $T(n)$ depende dos valores dos t_j.

Melhor caso

O melhor caso de Ordena-Por-Inserção ocorre quando o vetor A já está ordenado. Para $j = 2, \ldots, n$ temos $A[i] \leq$ *chave* na linha 5 quando $i = j - 1$. Assim, $t_j = 1$ para $j = 2, \ldots, n$.

Logo,

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

Este tempo de execução é da forma $an + b$ para constantes a e b que dependem apenas dos c_i.

Portanto, **no melhor caso**, o tempo de execução é uma **função linear** no **tamanho da entrada**.

Pior Caso

Quando o vetor A está em ordem decrescente, ocorre o pior caso para Ordena-Por-Inserção. Para inserir a *chave* em $A[1 \dots j-1]$, temos que compará-la com todos os elementos neste subvetor. Assim, $t_j = j$ para $j = 2, \dots, n$.

Lembre-se que:

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

e

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

Pior caso – continuação

Temos então que

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right) n \\
&\quad - (c_2 + c_4 + c_5 + c_8)
\end{aligned}
$$

O tempo de execução no pior caso é da forma $an^2 + bn + c$ onde a, b, c são constantes que dependem apenas dos c_i.

Portanto, **no pior caso**, o tempo de execução é uma **função quadrática** no **tamanho da entrada**.

Complexidade assintótica de algoritmos

- Como dito anteriormente, na maior parte desta disciplina, estaremos nos concentrando na **análise de pior caso** e no **comportamento assintótico** dos algoritmos (instâncias de tamanho grande).
- O algoritmo Ordena-Por-Inserção tem como complexidade (de pior caso) uma função quadrática $an^2 + bn + c$, onde a, b, c são constantes absolutas que dependem apenas dos custos c_i.
- O estudo assintótico nos permite "jogar para debaixo do tapete" os valores destas constantes, i.e., aquilo que independe do tamanho da entrada (neste caso os valores de a, b e c).
- **Por que podemos fazer isso ?**

Análise assintótica de funções quadráticas

Considere a função quadrática $3n^2 + 10n + 50$:

n	$3n^2 + 10n + 50$	$3n^2$	Diferença percentual
64	12978	12288	5,32%
128	50482	49152	2,63%
512	791602	786432	0,65%
1024	3156018	3145728	0,33%
2048	12603442	12582912	0,16%
4096	50372658	50331648	0,08%
8192	201408562	201326592	0,04%
16384	805470258	805306368	0,02%
32768	3221553202	3221225472	0,01%

Como se vê, $3n^2$ é o termo dominante quando n é grande.

De um modo geral, podemos nos concentrar nos termos dominantes e esquecer os demais.

Notação assintótica

- Usando notação assintótica, dizemos que o algoritmo Ordena-Por-Inserção tem complexidade de tempo de pior caso $\Theta(n^2)$.
- Isto quer dizer duas coisas:
  - a complexidade de tempo é limitada (superiormente) assintoticamente por algum polinômio da forma an^2 para alguma constante a,
  - para todo n suficientemente grande, existe alguma instância de tamanho n que consome tempo pelo menos dn^2, para alguma contante positiva d.
- Mais adiante discutiremos em detalhes o uso da notação assintótica em análise de algoritmos.

Ordenação por intercalação

Ordenação por intercalação

Q que significa intercalar dois (sub)vetores ordenados?

Problema: Dados $A[p \ldots q]$ e $A[q{+}1 \ldots r]$ crescentes, rearranjar $A[p \ldots r]$ de modo que ele fique em ordem crescente.

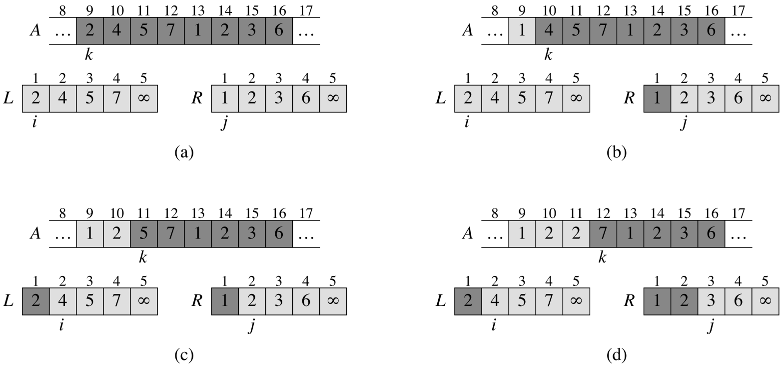
Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

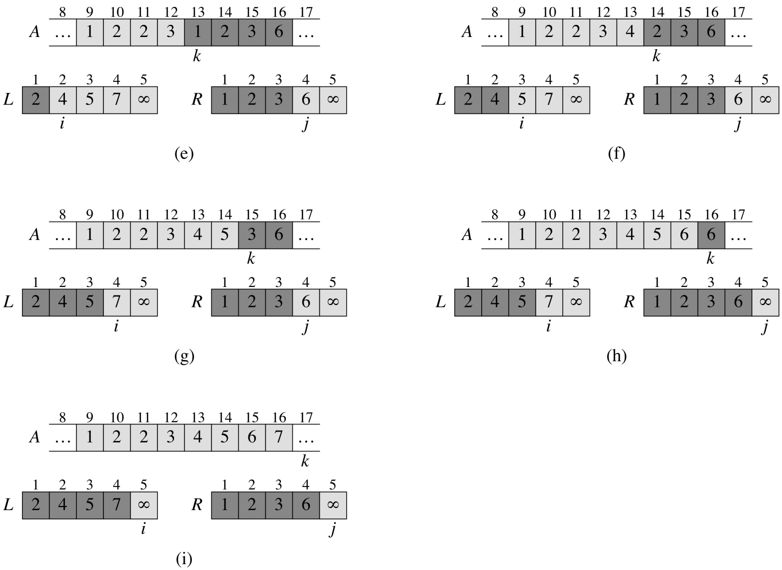
	p				q				r
A	11	22	33	44	55	66	77	88	99

Intercalação com sentinela



(a) (b) (c) (d)

Intercalação com sentinela



Intercalação com sentinela

INTERCALA(A, p, q, r)

$n_1 \leftarrow q - p + 1$
$n_2 \leftarrow r - q$
sejam $L[1..n_1 + 1]$ e $R[1..n_2 + 1]$ novos vetores
para $i \leftarrow 1$ **até** n_1 **faça**
    $L[i] \leftarrow A[p + i - 1]$
para $j \leftarrow 1$ **até** n_2 **faça**
    $R[j] \leftarrow A[q + j]$
$L[n_1 + 1] \leftarrow \infty$
$R[n_2 + 1] \leftarrow \infty$
$i \leftarrow 1$
$j \leftarrow 1$
para $k \leftarrow p$ **até** r **faça**
    se $L[i] \leq R[j]$ **então**
        $A[k] \leftarrow L[i]$
        $i \leftarrow i + 1$
    senão
        $A[k] = R[j]$
        $j \leftarrow j + 1$

Outro algoritmo de intercalação (sem sentinela)

Intercalação

	p				q				r
A	22	33	55	77	99	11	44	66	88
B									

Intercalação

	k								
A									
	i								j
B	22	33	55	77	99	88	66	44	11

Intercalação

		k							
A	11								
	i							j	
B	22	33	55	77	99	88	66	44	11

Intercalação

			k						
A	11	22							

		i						j	
B	22	33	55	77	99	88	66	44	11

Intercalação

				k					
A	11	22	33						
			i					j	
B	22	33	55	77	99	88	66	44	11

Intercalação

					k				
A	11	22	33	44					

			i				j		
B	22	33	55	77	99	88	66	44	11

Intercalação

						k			
A	11	22	33	44	55				
				i			j		
B	22	33	55	77	99	88	66	44	11

Intercalação

							k		
A	11	22	33	44	55	66			
				i		*j*			
B	22	33	55	77	99	88	66	44	11

Intercalação

								k	
A	11	22	33	44	55	66	77		
					i	j			
B	22	33	55	77	99	88	66	44	11

Intercalação

									k
A	11	22	33	44	55	66	77	88	

					$i = j$				
B	22	33	55	77	99	88	66	44	11

Intercalação

A	11	22	33	44	55	66	77	88	99
					j	i			
B	22	33	55	77	99	88	66	44	11

Intercalação

Pseudo-código

```
INTERCALA(A, p, q, r)
    para i ← p até q faça
        B[i] ← A[i]
    para j ← q + 1 até r faça
        B[r + q + 1 − j] ← A[j]
    i ← p
    j ← r
    para k ← p até r faça
        se B[i] ≤ B[j]
            então A[k] ← B[i]
                  i ← i + 1
            senão A[k] ← B[j]
                  j ← j − 1
```

Complexidade de Intercala

Entrada:

	p				q				r
A	22	33	55	77	99	11	44	66	88

Saída:

	p				q				r
A	11	22	33	44	55	66	77	88	99

Tamanho da entrada: $n = r - p + 1$

Consumo de tempo: $\Theta(n)$

Corretude de Intercala

Invariante principal de Intercala:

No começo de cada iteração do laço das linhas 7–12, vale que:

1. $A[p \dots k-1]$ está ordenado,
2. $A[p \dots k-1]$ contém todos os elementos de $B[p \dots i-1]$ e de $B[j+1 \dots r]$,
3. $B[i] \geq A[k-1]$ e $B[j] \geq A[k-1]$.

Exercício. Prove que a afirmação acima é de fato um invariante de INTERCALA.

Exercício. (fácil) Mostre usando o invariante acima que INTERCALA é correto.

Projeto por indução e algoritmos recursivos

> "To understand recursion, we must first understand recursion."
>
> *(anônimo)*

- Um algoritmo recursivo obtém a saída para uma instância de de um problema chamando a si mesmo para resolver instâncias menores deste mesmo problema (trata-se de um projeto por indução).
- A resolução por projeto de indução, deve reduzir um problema a subproblemas menores do mesmo tipo. E problemas suficientemente pequenos devem ser resolvidos de maneira direta.

Algoritmos recursivos

- O que é o paradigma de **divisão-e-conquista**?
- Como mostrar a corretude de um algoritmo recursivo?
- Como analisar o consumo de tempo de um algoritmo recursivo?
- O que é uma **fórmula de recorrência**?
- O que significa *resolver* uma fórmula de recorrência?

Recursão e o paradigma de divisão-e-conquista

- Algoritmos de divisão-e-conquista possuem as seguintes etapas em cada nível de recursão:
  1. **Problemas pequenos:** Quando os problemas são suficientemente pequenos, então o algoritmo recursivo deve resolver o problema de maneira direta.
  2. **Problemas que não são pequenos:**
     1. **Divisão:** o problema é dividido em subproblemas semelhantes ao problema original, porém tendo como entrada instâncias de tamanho menor.
     2. **Conquista:** cada subproblema é resolvido recursivamente a menos que o tamanho de sua entrada seja suficientemente "pequeno", quando este é resolvido diretamente.
     3. **Combinação:** as soluções dos subproblemas são combinadas para obter uma solução do problema original.

Exemplo de divisão-e-conquista: *Mergesort*

- Mergesort é um algoritmo para resolver o problema de ordenação e um exemplo clássico do uso do paradigma de divisão-e-conquista. (*to merge* = intercalar)
- Descrição do Mergesort em alto nível:
  1. **Divisão**: divida o vetor com n elementos em dois subvetores de tamanho $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$, respectivamente.
  2. **Conquista**: ordene os dois vetores recursivamente usando o Mergesort;
  3. **Combinação**: intercale os dois subvetores para obter um vetor ordenado usando o algoritmo Intercala.

Mergesort

Relembrando: o objetivo é rearranjar $A[p \ldots r]$, com $p \leq r$, em ordem crescente.

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

	p				q				r
A	66	33	55	44	99	11	77	22	88

Mergesort

Relembrando: o objetivo é rearranjar $A[p \ldots r]$, com $p \leq r$, em ordem crescente.

MERGESORT(A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGESORT(A, p, q)
4 MERGESORT$(A, q + 1, r)$
5 INTERCALA(A, p, q, r)

	p				q				r
A	33	44	55	66	99	11	77	22	88

Mergesort

Relembrando: o objetivo é rearranjar $A[p \dots r]$, com $p \leq r$, em ordem crescente.

MERGESORT(A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p+r)/2 \rfloor$
3 MERGESORT(A, p, q)
4 MERGESORT$(A, q+1, r)$
5 INTERCALA(A, p, q, r)

	p				q				r
A	33	44	55	66	99	11	22	77	88

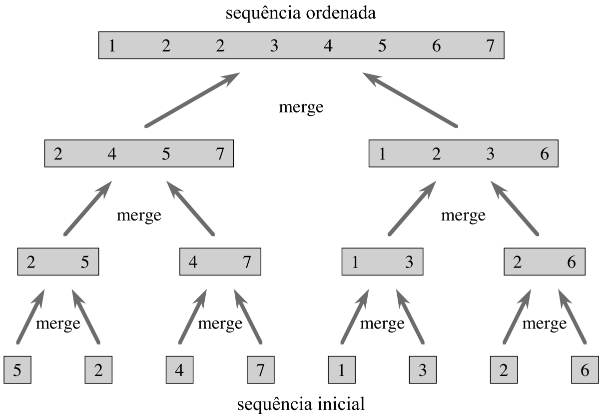
Relembrando: o objetivo é rearranjar $A[p \dots r]$, com $p \leq r$, em ordem crescente.

```
MERGESORT(A, p, r)
    se p < r
        então q ← ⌊(p + r)/2⌋
            MERGESORT(A, p, q)
            MERGESORT(A, q + 1, r)
            INTERCALA(A, p, q, r)
```

	p				q				r
A	11	22	33	44	55	66	77	88	99

Mergesort – exemplo do livro

- Exemplo do livro (CLRS)
- Visualização de cada “merge” do algoritmo



Corretude do Mergesort

```
MERGESORT(A, p, r)
    se p < r
        então q ← ⌊(p + r)/2⌋
              MERGESORT(A, p, q)
              MERGESORT(A, q + 1, r)
              INTERCALA(A, p, q, r)
```

O algoritmo está correto?

A corretude do algoritmo Mergesort apoia-se na corretude do algoritmo Intercala e pode ser demonstrada **por indução** em $n := r - p + 1$.

Aprenderemos como fazer provas por indução mais adiante.

Complexidade do Mergesort

MERGESORT(A, p, r)
se $p < r$
  então $q \leftarrow \lfloor (p + r)/2 \rfloor$
    MERGESORT(A, p, q)
    MERGESORT$(A, q + 1, r)$
    INTERCALA(A, p, q, r)

Qual é a complexidade de MERGESORT?

Seja $T(n) :=$ o consumo de tempo máximo (pior caso) em função de $n = r - p + 1$

Complexidade do Mergesort

```
MERGESORT(A, p, r)
1    se p < r
2        então q ← ⌊(p + r)/2⌋
3              MERGESORT(A, p, q)
4              MERGESORT(A, q + 1, r)
5              INTERCALA(A, p, q, r)
```

linha	consumo de tempo
1	?
2	?
3	?
4	?
5	?

$T(n) = ?$

Complexidade do Mergesort

MERGESORT(A, p, r)
1 **se** $p < r$
2 **então** $q \leftarrow \lfloor (p + r)/2 \rfloor$
3 MERGESORT(A, p, q)
4 MERGESORT$(A, q + 1, r)$
5 INTERCALA(A, p, q, r)

linha	consumo de tempo
1	$\Theta(1)$
2	$\Theta(1)$
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	$\Theta(n)$

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) + \Theta(2)$$

Complexidade do Mergesort

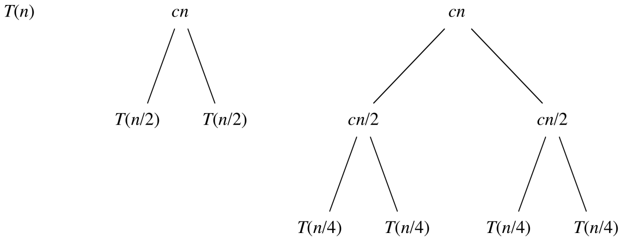
- Obtemos o que chamamos de fórmula de recorrência (i.e., uma fórmula definida em termos de si mesma).

$$T(1) = \Theta(1)$$
$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \quad \text{para } n = 2, 3, 4, \ldots$$

- Em geral, ao aplicar o paradigma de divisão-e-conquista, chega-se a um algoritmo recursivo cuja complexidade $T(n)$ é uma fórmula de recorrência.
- É necessário então resolver a recorrência! Mas, o que significa resolver uma recorrência?
- Significa encontrar uma "fórmula fechada" para $T(n)$.
- No caso, $T(n) = \Theta(n \lg n)$. Assim, o consumo de tempo do Mergesort é $\Theta(n \lg n)$ no pior caso.
- Veremos mais tarde como resolver recorrências.

Mergesort – árvore de recursão

- Árvore de recursão do Mergesort

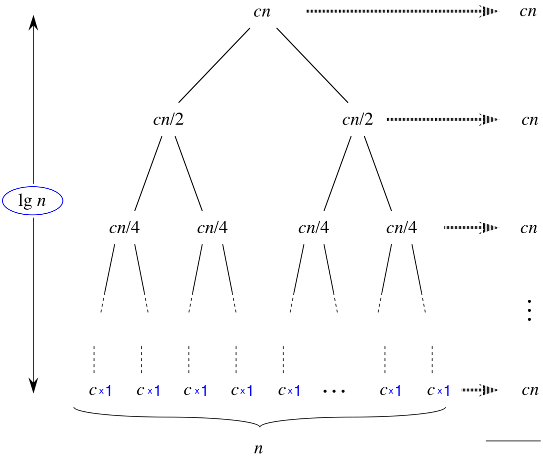


(a) (b) (c)

Mergesort – árvore de recursão

- Árvore de recursão do Mergesort

$$\frac{n}{2^i} = 1$$
$$2^i = n$$
$$\lg 2^i = \lg n$$
$$i = \lg n$$



Total: $cn \lg n + cn$

(d)

Crescimento de funções

Notação Assintótica

- Vamos expressar complexidade através de funções em variáveis que descrevam o tamanho de instâncias do problema. Exemplos:
  - Problemas de aritmética de precisão arbitrária: número de bits (ou bytes) dos inteiros.
  - Problemas em grafos: número de vértices e/ou arestas
  - Problemas de ordenação de vetores: tamanho do vetor.
  - Busca em textos: número de caracteres do texto ou padrão de busca.
- Vamos supor que funções que expressam complexidade são sempre positivas, já que estamos medindo número de operações.

Comparação de Funções

- 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	$\approx 1,26 \cdot 10^{30}$	$\approx 1,07 \cdot 10^{301}$	?	?	?

Análise assintótica

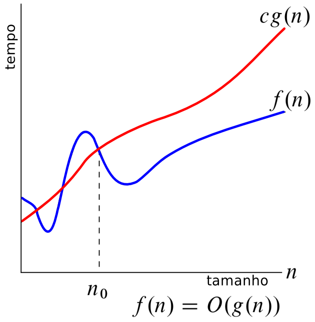
- Precisamos de uma ferramenta matemática para **comparar funções**
- Para a análise de algoritmo será feita uma **análise assintótica**:
  - Matematicamente: estudando o comportamento de **limites** ($n \to \infty$)
  - Computacionalmente: estudando o comportamento para entrada arbitrariamente grande ou descrevendo **taxa de crescimento**
- Para isso, uma **notação** específica é usada: O, Ω, Θ, o, ω
- O foco está nas **ordens de crescimento**

Classe 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 $g(n)$.



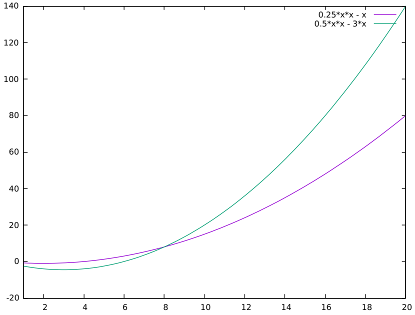
Classe O

Exemplo

$$f(n) = \frac{1}{4}n^2 - n$$
$$g(n) = n^2 - 6n$$

Valores de c e n_0 que satisfazem $f(n) \in O(g(n))$:

$$c = \frac{1}{2} \quad \text{e} \quad n_0 = 8$$

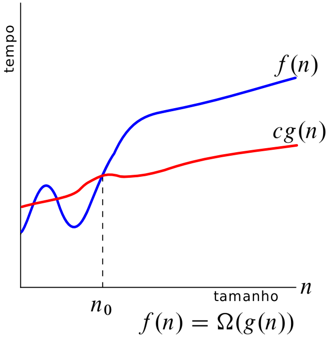


Classe Ω

Definição:

$\Omega(g(n)) = \{f(n) :$ existem constantes positivas c e n_0 tais que $0 \leq cg(n) \leq f(n)$, 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 $g(n)$.



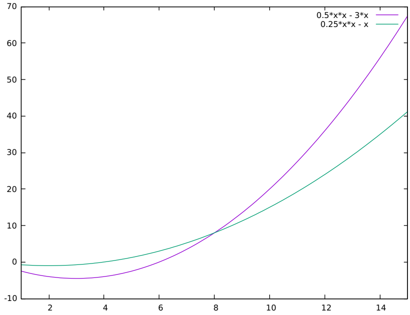
Classe Ω

Exemplo

$$f(n) = \frac{1}{2}n^2 - 3n$$
$$g(n) = \frac{1}{2}n^2 - 2n$$

Valores de c e n_0 que satisfazem $f(n) \in \Omega(g(n))$:

$$c = \frac{1}{2} \quad \text{e} \quad n_0 = 8$$

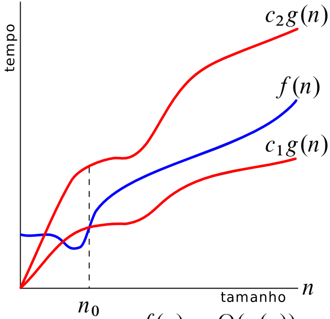


Classe Θ

Definição:

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

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



$f(n) = \Theta(g(n))$

Classe Θ

Definição:

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

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

Exemplo:

$\frac{1}{2}n^2 - 3n \in \Theta(n^2)$
Valores de c_1, c_2 e n_0 que satisfazem a definição são

$$c_1 = \frac{1}{14},\ c_2 = \frac{1}{2} \text{ e } n_0 = 7.$$

Classe o

Definição:

$o(g(n)) = \{f(n) :$ para toda constante positiva c, existe uma constante $n_0 > 0$ tal que $0 \leq f(n) < cg(n)$, para todo $n \geq n_0\}$.

Informalmente, dizemos que, se $f(n) \in o(g(n))$, então $f(n)$ cresce mais lentamente que $g(n)$.

Exemplo:

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

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

Classe ω

Definição:

$\omega(g(n)) = \{f(n) :$ para toda constante positiva c, existe uma constante $n_0 > 0$ tal que $0 \leq cg(n) < f(n)$, para todo $n \geq n_0.\}$

Informalmente, dizemos que, se $f(n) \in \omega(g(n))$, então $f(n)$ cresce mais rapidamente que $g(n)$.

Exemplo:

$\frac{1}{1000}n^2 \in \omega(n)$
Para todo valor de c, um n_0 que satisfaz a definição é

$$n_0 = \lceil 1000c \rceil + 1.$$

Notação assintótica – resumo

- $f(n) \in O(g(n))$ se houver constantes positivas n_0 e c tal que $f(n) \leq c\, g(n)$ para todo $n \geq n_0$
- $f(n) \in \Omega(g(n))$ se houver constantes positivas n_0 e c tal que $f(n) \geq c\, g(n)$ para todo $n \geq n_0$
- $f(n) \in \Theta(g(n))$ se houver constantes positivas n_0, c_1 e c_2 tal que $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ para todo $n \geq n_0$
- $f(n) \in o(g(n))$ se, para qualquer constante positiva c, existe n_0 tal que $f(n) < c\, g(n)$ para todo $n \geq n_0$
- $f(n) \in \omega(g(n))$ se, para qualquer constante positiva c, existe n_0 tal que $f(n) > c\, g(n)$ para todo $n \geq n_0$

Notação assintótica – analogia

Analogia entre duas funções f e g e dois números a e b:

- $f(n) \in O(g(n)) \quad \approx \quad a \leq b$
- $f(n) \in \Omega(g(n)) \quad \approx \quad a \geq b$
- $f(n) \in \Theta(g(n)) \quad \approx \quad a = b$
- $f(n) \in o(g(n)) \quad \approx \quad a < b$
- $f(n) \in \omega(g(n)) \quad \approx \quad a > b$

Definições equivalentes

$$f(n) \in o(g(n)) \text{ se } \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0.$$

$$f(n) \in O(g(n)) \text{ se } \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty.$$

$$f(n) \in \Theta(g(n)) \text{ se } 0 < \lim_{n\to\infty} \frac{f(n)}{g(n)} < \infty.$$

$$f(n) \in \Omega(g(n)) \text{ se } \lim_{n\to\infty} \frac{f(n)}{g(n)} > 0.$$

$$f(n) \in \omega(g(n)) \text{ se } \lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty.$$

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 $f(n) \in \omega(h(n))$.

Propriedades das Classes

Reflexividade:

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

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

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

Simetria:

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

Simetria Transposta:

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

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

Notação assintótica – algumas regras práticas

- **Multiplicação por uma constante:**
  $\Theta(c\ f(n)) = \Theta(f(n))$
  $99\ n^2 = \Theta(n^2)$

- **Mais alto expoente** de um polinômio
  $a_x n^x + a_{x-1} n^{x-1} + \cdots + a_2 n^2 + a_1 n + a_0$:
  $3\mathbf{n^3} - 5n^2 + 100 = \Theta(n^3)$
  $6\mathbf{n^4} - 20n^2 = \Theta(n^4)$
  $0.8\mathbf{n} + 224 = \Theta(n)$

- **Termo dominante:**
  $\mathbf{2^n} + 6n^3 = \Theta(2^n)$
  $\mathbf{n}! - 3n^2 = \Theta(n!)$
  $n \log n + 3\mathbf{n^2} = \Theta(n^2)$

Notação assintótica – dominância

Quando uma função é **melhor** que outra?

- Se queremos reduzir o tempo, funções "menores" são melhores
- Uma função domina sobre outra se, a medida que n cresce, a função continua "maior"
- Matematicamente: $f(n) \gg g(n)$ se $\lim_{n \to \infty} \frac{g(n)}{f(n)} = 0$

Relações de dominância

$$n! \gg 2^n \gg n^3 \gg n^2 \gg n \log n \gg n \gg \log n \gg 1$$

Notação assintótica – visão prática

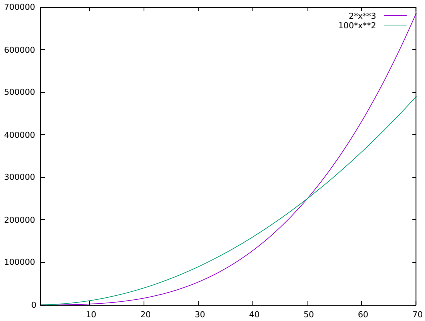
Se uma operação leva 10^{-9} segundos

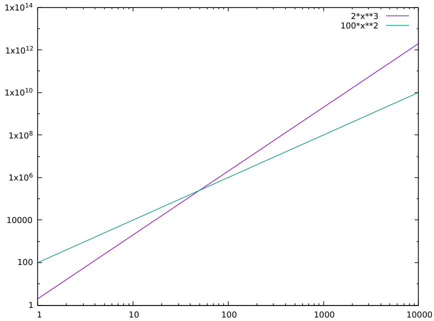
	$\log n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$
20	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	77 anos
30	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$1.07s$	
40	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	18.3 min	
50	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	13 dias	
100	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	10^{13} anos	
10^3	$< 0.01s$	$< 0.01s$	$< 0.01s$	$< 0.01s$	$1s$		
10^4	$< 0.01s$	$< 0.01s$	$< 0.01s$	$0.1s$	16.7 min		
10^5	$< 0.01s$	$< 0.01s$	$< 0.01s$	$10s$	11 dias		
10^6	$< 0.01s$	$< 0.01s$	$0.02s$	16.7 min	31 anos		
10^7	$< 0.01s$	$0.01s$	$0.23s$	1.16 dias			
10^8	$< 0.01s$	$0.1s$	$2.66s$	115 dias			
10^9	$< 0.01s$	$1s$	$29.9s$	31 anos			

Desenhando funções

- Comparando $2n^3$ com $100n^2$ usando o gnuplot:

```
gnuplot> plot [1:70] 2*x**3, 100*x**2
gnuplot> set logscale xy 10
gnuplot> plot [1:10000] 2*x**3, 100*x**2
```

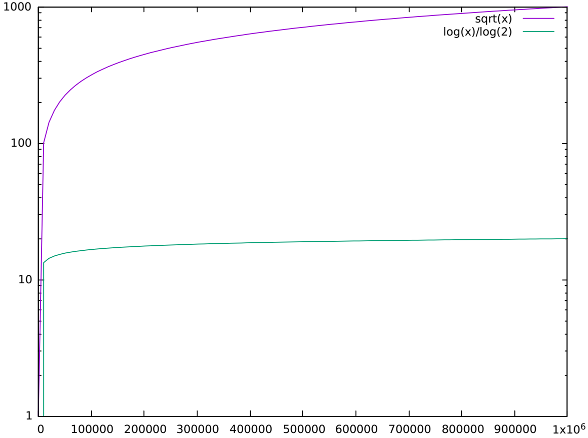




Desenhando funções

- Comparando $\sqrt{n}$ e $\log_2 n$:

```
gnuplot> set logscale y 10
gnuplot> plot [1:1000000] sqrt(x), log(x)/log(2)
```



Indução matemática

Na *Demonstração por Indução*, queremos demonstrar a validade de $P(n)$, uma propriedade P com um parâmetro natural n associado, para todo valor de n.
Há um número infinito de casos a serem considerados, um para cada valor de n. Demonstramos os infinitos casos de uma só vez:

- **Base da Indução:** Demonstramos $P(1)$.
- **Hipótese de Indução:** Supomos que $P(n)$ é verdadeiro.
- **Passo de Indução:** Provamos que $P(n+1)$ é verdadeiro, a partir da hipótese de indução.

Exemplo:

Prove que a soma dos n primeiros naturais ímpares é n^2.

Demonstração por Indução

Outra forma equivalente:

- **Base da Indução:** Demonstramos $P(1)$.
- **Hipótese de Indução:** Supomos que $P(n-1)$ é verdadeiro.
- **Passo de Indução:** Provamos que $P(n)$ é verdadeiro, a partir da hipótese de indução.

Exemplo:

Prove que a soma dos n primeiros naturais ímpares é n^2.

Demonstração por Indução

Às vezes queremos provar que uma proposição $P(n)$ vale para $n \geq n_0$ para algum n_0.

- **Base da Indução:** Demonstramos $P(n_0)$.
- **Hipótese de Indução:** Supomos que $P(n-1)$ é verdadeiro.
- **Passo de Indução:** Provamos que $P(n)$ é verdadeiro, a partir da hipótese de indução.

Exemplo:

Prove que todo inteiro $n \geq 2$ pode ser fatorado como um produto de primos.

Indução Fraca $\times$ Indução Forte

A *indução forte* difere da *indução fraca* (ou *simples*) apenas na suposição da hipótese.
No caso da indução forte, devemos supor que a propriedade vale para todos os casos anteriores, não somente para o anterior, ou seja:

- **Base da Indução:** Demonstramos $P(1)$.
- **Hipótese de Indução Forte:** Supomos que $P(k)$ é verdadeiro, para todo $1 \leq k < n$.
- **Passo de Indução:** Provamos que $P(n)$ é verdadeiro, a partir da hipótese de indução.

Exemplo:

Prove que todo inteiro $n \geq 2$ pode ser fatorado como um produto de primos.

Exemplo 1

Demonstre que a inequação

$$(1 + x)^n \geq 1 + nx$$

vale para todo natural n e real x tal que $(1 + x) > 0$.

Demonstração:

- A **base da indução** é $n = 1$. Nesse caso ambos os lados da inequação são iguais a $1 + x$, mostrando a sua validade. Isto encerra a prova do caso base.

Exemplo 1 (cont.)

- A **hipótese de indução** é: *Suponha que a inequação vale para n, isto é,* $(1+x)^n \geq 1+nx$ *para todo real* x *tal que* $(1+x) > 0$.
- O **passo de indução é:** *Supondo a h.i., vamos mostrar que a inequação vale para o valor* $n+1$*, isto é,* $(1+x)^{n+1} \geq 1+(n+1)x$ *para todo* x *tal que* $(1+x) > 0$. A dedução é simples:

$$\begin{aligned}
(1+x)^{n+1} &= (1+x)^n(1+x) \\
&\geq (1+nx)(1+x) \text{ (pela h.i. e } (1+x) > 0) \\
&= 1+(n+1)x+nx^2 \\
&\geq 1+(n+1)x \text{ (já que } nx^2 \geq 0)
\end{aligned}$$

A última linha mostra que a inequação vale para $n+1$, completando a demonstração. ■

Exemplo 2

Demonstre que o número T_n de regiões no plano criadas por n retas em posição geral é igual a

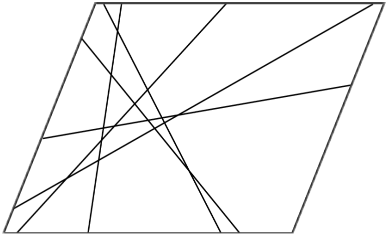
$$T_n = \frac{n(n+1)}{2} + 1.$$

Um conjunto de retas está em posição geral no plano se

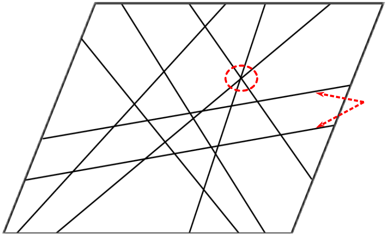
- todas as retas são concorrentes, isto é, não há retas paralelas e
- não há três retas interceptando-se no mesmo ponto.

Exemplo 2 (cont.)

Antes de prosseguirmos com a demonstração vejamos exemplos de um conjunto de retas que está em posição geral e outro que não está.



Em posição geral



Não estão em posição geral

Exemplo 2 (cont.)

Demonstração: A idéia que queremos explorar para o passo de indução é a seguinte: supondo que a fórmula vale para n, adicionar uma nova reta em posição geral e tentar assim obter a validade de $n+1$.

- A **base da indução** é, naturalmente, $n = 1$. Uma reta sozinha divide o plano em duas regiões. De fato,

  $$T_1 = (1 \times 2)/2 + 1 = 2.$$

  Isto conclui a prova para $n = 1$.

Exemplo 2 (cont.)

- A **hipótese de indução** é: *Suponha que* $T_n = (n(n+1)/2) + 1$ *para n.*
- O **passo de indução é:** *Supondo a h.i., vamos mostrar que para* $n+1$ *retas em posição geral vale que*

$$T_{n+1} = \frac{(n+1)(n+2)}{2} + 1.$$

Considere um conjunto L de $n+1$ retas em posição geral no plano e seja r uma dessas retas. Então, as retas do conjunto $L' = L \setminus \{r\}$ obedecem à hipótese de indução e, portanto, o número de regiões distintas do plano definidas por elas é $(n(n+1))/2 + 1$.

Exemplo 2 (cont.)

- Além disso, r intersecta as outras n retas em n pontos distintos. O que significa que, saindo de uma ponta de r no infinito e após cruzar as n retas de L', a reta r terá cruzado $n+1$ regiões, dividindo cada uma destas em duas outras.
- Assim, podemos escrever que

$$\begin{aligned} T_{n+1} &= T_n + n + 1 \\ &= \frac{n(n+1)}{2} + 1 + n + 1 \text{ (pela h.i.)} \\ &= \frac{(n+1)(n+2)}{2} + 1. \end{aligned}$$

Isso conclui a demonstração. ■

Exemplo 3

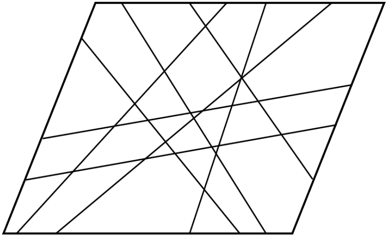
Definição:

Um conjunto de n retas no plano define regiões convexas cujas bordas são segmentos das n retas. Duas dessas regiões são *adjacentes* se as suas bordas se intersectam em algum segmento de reta não trivial, isto é contendo mais que um ponto.

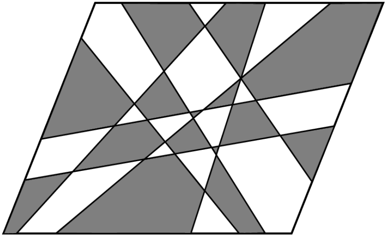
Uma *k-coloração* dessas regiões é uma atribuição de uma de k cores a cada uma das regiões, de forma que regiões adjacentes recebam cores distintas.

Exemplo 3 (cont.)

Veja exemplos dessas definições:



As regiões convexas



Uma 2-coloração do plano

Exemplo 3 (cont.)

Demonstre que para todo $n \geq 1$, existe uma 2-coloração das regiões formadas por n retas no plano.

Demonstração:

- A **base da indução** é, naturalmente, $n = 1$. Uma reta sozinha divide o plano em duas regiões. Atribuindo-se cores diferentes a essas regiões obtemos o resultado desejado.

  Isto conclui a prova para $n = 1$.

Exemplo 3 (cont.)

- A **hipótese de indução** é: *Suponha que sempre existe uma 2-coloração das regiões formadas por n retas no plano.*
- O **passo de indução é:** *Supondo a h.i., vamos exibir uma 2-coloração para as regiões formadas por* $n+1$ *retas no plano.*

  A demonstração do passo consiste em observar que a adição de uma nova reta *r* divide cada região atravessada por *r* em duas, e definir a nova 2-coloração da seguinte forma: as regiões em um lado de *r* mantém a cor herdada da hipótese de indução; as regiões no outro lado de *r* têm suas cores trocadas.

  Você é capaz de demonstrar que a 2-coloração obtida nesse processo obedece à definição?

Demonstração por Indução

Exemplos: Apesar da reconhecida validade dos seguintes somatórios, efetue provas por indução matemática da

1. Soma dos n termos de uma progressão aritmética (PA):

$$a_1 + (a_1 + r) + (a_1 + 2r) + \cdots + [a_1 + (n-1)r] \; =$$

$$= \sum_{i=0}^{n-1} (a_1 + i \cdot r) \; = \; \frac{n(a_1 + [a_1 + (n-1)r])}{2}$$

2. Soma dos n termos de uma progressão geométrica (PG):

$$a_1 + (a_1 \cdot q) + (a_1 \cdot q^2) + \cdots + (a_1 \cdot q^{n-1}) =$$

$$= \sum_{i=0}^{n-1} (a_1 \cdot q^i) = \frac{a_1(q^n - 1)}{q - 1}$$

Recorrências

Resolução de Recorrências

- Relações de recorrência expressam a complexidade de algoritmos recursivos como, por exemplo, os algoritmos de divisão e conquista.
- É preciso saber resolver as recorrências para que possamos efetivamente determinar a complexidade dos algoritmos recursivos.

Mergesort

```
MERGESORT(A, p, r)
    se p < r
        então q ← ⌊(p + r)/2⌋
              MERGESORT(A, p, q)
              MERGESORT(A, q + 1, r)
              INTERCALA(A, p, q, r)
```

Qual é a complexidade de MERGESORT?

Seja $T(n) :=$ o consumo de tempo máximo (pior caso) em função de $n = r - p + 1$

Complexidade do Mergesort

```
MERGESORT(A, p, r)
1  se p < r
2     então q ← ⌊(p + r)/2⌋
3           MERGESORT(A, p, q)
4           MERGESORT(A, q + 1, r)
5           INTERCALA(A, p, q, r)
```

linha	consumo de tempo
1	?
2	?
3	?
4	?
5	?

$$T(n) = ?$$

Complexidade do Mergesort

```
MERGESORT(A, p, r)
1    se p < r
2        então q ← ⌊(p + r)/2⌋
3              MERGESORT(A, p, q)
4              MERGESORT(A, q + 1, r)
5              INTERCALA(A, p, q, r)
```

linha	consumo de tempo
1	b_0
2	b_1
3	$T(\lceil n/2 \rceil)$
4	$T(\lfloor n/2 \rfloor)$
5	an

$$T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + (b_0 + b_1)$$

Resolução de recorrências

- Queremos resolver a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + an + b \quad \text{para } n \geq 2. \end{aligned}$$

- Resolver uma recorrência significa encontrar uma fórmula fechada para $T(n)$.
- Não é necessário achar uma solução exata. Basta encontrar uma função $f(n)$ tal que $T(n) \in \Theta(f(n))$.

Resolução de recorrências

Alguns métodos para resolução de recorrências:

- substituição
- iteração
- árvore de recorrência

Veremos também um resultado bem geral que permite resolver várias recorrências: Master theorem.

Método da substituição

- Idéia básica: "adivinhe" qual é a solução e prove por indução que ela funciona!
- Método poderoso mas nem sempre aplicável (obviamente).
- Com prática e experiência fica mais fácil de usar!

Exemplo

Considere a recorrência:

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \qquad \text{para } n \geq 2. \end{aligned}$$

Chuto que $T(n) \in O(n \lg n)$.

Mais precisamente, chuto que $T(n) \leq 3n \lg n$.

(Lembre que $\lg n = \log_2 n$.)

Exemplo

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\
&\leq 3 \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + 3 \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\leq 3 \left\lceil \frac{n}{2} \right\rceil \lg n + 3 \left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + n \\
&= 3 \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n - 3 \left\lfloor \frac{n}{2} \right\rfloor + n \\
&= 3n \lg n - 3 \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\leq 3n \lg n.
\end{aligned}
$$

(Yeeeeeeeessssss!)

Exemplo

- Mas espere um pouco!
- $T(1) = 1$ e $3.1.\lg 1 = 0$ e a base da indução não funciona!
- Certo, mas lembre-se da definição da classe $O(\ )$.

  Só preciso provar que $T(n) \leq 3n \lg n$ para $n \geq n_0$ onde n_0 é alguma constante.

  Vamos tentar com $n_0 = 2$. Nesse caso

  $$T(2) = T(1) + T(1) + 2 = 4 \leq 3.2.\lg 2 = 6,$$

  e estamos feitos.

Exemplo

- Certo, funcionou para $T(1) = 1$.
- Mas e se por exemplo $T(1) = 8$?

  Então $T(2) = 8 + 8 + 2 = 18$ e $3.2. \lg 2 = 6$.
  Não deu certo...
- Certo, mas aí basta escolher uma constante maior. Mostra-se do mesmo jeito que $T(n) \leq 10n \lg n$ e para esta escolha $T(2) = 18 \leq 10.2. \lg 2 = 20$.
- De modo geral, se o passo de indução funciona ($T(n) \leq cn \lg n$), é possível escolher c e a base da indução (n_0) de modo conveniente!

Como achar as constantes?

- Tudo bem. Dá até para chutar que $T(n)$ pertence a classe $O(n \lg n)$.
- Mas como descobrir que $T(n) \leq 3n \lg n$? Como achar a constante 3?
- Eis um método simples: suponha como hipótese de indução que $T(n) \leq cn \lg n$ para $n \geq n_0$ onde c e n_0 são constantes que vou tentar determinar.

Primeira tentativa

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\
&\le c\left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c\left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\le c\left\lceil \frac{n}{2} \right\rceil \lg n + c\left\lfloor \frac{n}{2} \right\rfloor \lg n + n \\
&= c\left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor\right) \lg n + n \\
&= cn \lg n + n
\end{aligned}
$$

(Hummm, não deu certo…)

Segunda tentativa

$$
\begin{aligned}
T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n \\
&\le c \left\lceil \frac{n}{2} \right\rceil \lg \left\lceil \frac{n}{2} \right\rceil + c \left\lfloor \frac{n}{2} \right\rfloor \lg \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\le c \left\lceil \frac{n}{2} \right\rceil \lg n + c \left\lfloor \frac{n}{2} \right\rfloor (\lg n - 1) + n \\
&= c \left(\left\lceil \frac{n}{2} \right\rceil + \left\lfloor \frac{n}{2} \right\rfloor \right) \lg n - c \left\lfloor \frac{n}{2} \right\rfloor + n \\
&= cn \lg n - c \left\lfloor \frac{n}{2} \right\rfloor + n \\
&\le cn \lg n.
\end{aligned}
$$

Para garantir a última desigualdade basta que $-c\lceil n/2 \rceil + n \le 0$ e $c = 3$ funciona. (Yeeeeeeeesssss!)

Completando o exemplo

Mostramos que a recorrência

$$\begin{array}{rcll} T(1) & = & 1 & \\ T(n) & = & T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + n & \text{para } n \geq 2. \end{array}$$

satisfaz $T(n) \in O(n \lg n)$.

Mas quem garante que $T(n)$ não é "menor"?

O melhor é mostrar que $T(n) \in \Theta(n \lg n)$.

Resta então mostrar que $T(n) \in \Omega(n \lg n)$. A prova é similar. (Exercício!)

Como chutar?

Não há nenhuma receita genérica para adivinhar soluções de recorrências. A experiência é o fator mais importante.

Felizmente, há várias idéias que podem ajudar.

Considere a recorrência

$$\begin{array}{rcll} T(1) & = & 1 & \\ T(n) & = & 2T(\lfloor n/2 \rfloor) + n & \text{para } n \geq 2. \end{array}$$

Ela é quase idêntica à anterior e podemos chutar que $T(n) \in \Theta(n \lg n)$. Isto de fato é verdade. (Exercício ou consulte o CLRS)

Como chutar?

Considere agora a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor + 17) + n \quad \text{para } n \geq 2. \end{aligned}$$

Ela parece bem mais difícil por causa do "17" no lado direito.

Intuitivamente, porém, isto não deveria afetar a solução. Para n **grande** a diferença entre $T(\lfloor n/2 \rfloor)$ e $T(\lfloor n/2 \rfloor + 17)$ não é tanta.

Chuto então que $T(n) \in \Theta(n \lg n)$. (Exercício!)

Truques e sutilezas

Algumas vezes adivinhamos corretamente a solução de uma recorrência, mas as contas aparentemente não funcionam! Em geral, o que é necessário é fortalecer a hipótese de indução.

Considere a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \quad \text{para } n \geq 2. \end{aligned}$$

Chutamos que $T(n) \in O(n)$ e tentamos mostrar que $T(n) \leq cn$ para alguma constante c.

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leq c\lceil n/2 \rceil + c\lfloor n/2 \rfloor + 1 \\ &= cn + 1. \end{aligned}$$

(Humm, falhou...)

E agora? Será que erramos o chute? Será que $T(n) \in \Theta(n^2)$?

Na verdade, adivinhamos corretamente. Para provar isso, é preciso usar uma hipótese de indução mais forte.

Vamos mostrar que $T(n) \leq cn - b$ onde $b > 0$ é uma constante.

$$\begin{aligned} T(n) &= T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 1 \\ &\leq c\lceil n/2 \rceil - b + c\lfloor n/2 \rfloor - b + 1 \\ &= cn - 2b + 1 \\ &\leq cn - b \end{aligned}$$

onde a última desigualdade vale se $b \geq 1$.
(Yeeeessss!)

Método da iteração

- Não é necessário adivinhar a resposta!
- Precisa fazer mais contas!
- Idéia: expandir (iterar) a recorrência e escrevê-la como uma somatória de termos que dependem apenas de n e das condições iniciais.
- Precisa conhecer limitantes para várias somatórias.

Método da iteração

Considere a recorrência

$$
\begin{aligned}
T(n) &= b && \text{para } n \leq 3,\\
T(n) &= 3T(\lfloor n/4 \rfloor) + n && \text{para } n \geq 4.
\end{aligned}
$$

Iterando a recorrência obtemos

$$
\begin{aligned}
T(n) &= n + 3T(\lfloor n/4 \rfloor)\\
&= n + 3(\lfloor n/4 \rfloor + 3T(\lfloor n/16 \rfloor))\\
&= n + 3(\lfloor n/4 \rfloor + 3(\lfloor n/16 \rfloor + 3T(\lfloor n/64 \rfloor)))\\
&= n + 3\lfloor n/4 \rfloor + 9\lfloor n/16 \rfloor + 27T(\lfloor n/64 \rfloor).
\end{aligned}
$$

Certo, mas quando devo parar?
O i-ésimo termo da série é $3^i \lfloor n/4^i \rfloor$. Ela termina quando $\lfloor n/4^i \rfloor \leq 3$, ou seja, $i \geq \log_4 n$.

Método da iteração

Como $\lfloor n/4^i \rfloor \leq n/4^i$ temos que

$$T(n) \leq n + 3n/4 + 9n/16 + 27n/64 + \cdots + 3^j b$$

$$\begin{aligned}
T(n) &\leq n + 3n/4 + 9n/16 + 27n/64 + \cdots + d.3^{\log_4 n} \\
&\leq n.(1 + 3/4 + 9/16 + 27/64 + \cdots) + dn^{\log_4 3} \\
&= n \sum_{i=0}^{\infty} \left(\frac{3}{4}\right)^i + dn^{\log_4 3} \\
&= 4n + dn^{\log_4 3}
\end{aligned}$$

pois $3^{\log_4 n} = n^{\log_4 3}$ e $\sum_{i=0}^{\infty} q^i = \frac{1}{1-q}$ para $0 < q < 1$.

Como $\log_4 3 < 1$ segue que $n^{\log_4 3} \in o(n)$ e logo, $T(n) \in O(n)$.

Método de iteração

- As contas ficam mais simples se supormos que a recorrência está definida apenas para potências de um número, por exemplo, $n = 4^i$.
- Note, entretanto, que a recorrência deve ser provada para todo natural suficientemente grande.
- Muitas vezes, é possível depois de iterar a recorrência, adivinhar a solução e usar o método da substituição!

Método de iteração

$$
\begin{array}{rcll}
T(n) & = & b & \text{para } n \leq 3, \\
T(n) & = & 3T(\lfloor n/4 \rfloor) + n & \text{para } n \geq 4.
\end{array}
$$

Chuto que $T(n) \leq cn$.

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/4 \rfloor) + n \\
&\leq 3c\lfloor n/4 \rfloor + n \\
&\leq 3c(n/4) + n \\
&\leq cn
\end{aligned}
$$

onde a última desigualdade vale se $c \geq 4$.
(Yeeessss!)

Resolução pelo método da iteração

- A ideia da resolução pelo **método da iteração** (ou **expansão telescópica**) é expandir a relação de recorrência até que possa ser detectado seu comportamento no caso geral.
- Passos para resolver um equação de recorrência:
  1. Copie a fórmula original
  2. Descubra o passo (se $T(n)$ estiver escrito em função de $T(n/2)$, a cada passo o parâmetro é dividido por 2)
  3. Isole as equações para "os próximos passos"
  4. Substitua os valores isolados na fórmula original
  5. Identifique a fórmula do i-ésimo passo
  6. Descubra o valor de i de forma a igualar o parâmetro de $T(x)$ ao parâmetro (valor de n) no caso base
  7. Substitua o valor de i na fórmula do i-ésimo caso
  8. Identifique a complexidade dessa fórmula
  9. Prove por indução que a equação foi corretamente encontrada

Resolução por expansão telescópica

Exemplo 1: $\boxed{T(n) = 2T(n/2) + 2}$ $\boxed{T(1) = 1}$

1. $T(n) = 2T(n/2) + 2$ *(fórmula original)*
2. $T(n)$ está escrito em função de $T(n/2)$
3. Isole as equações para $T(n/2)$ e $T(n/4)$:
   $T(n/2) = 2(T(n/4)) + 2$
   $T(n/4) = 2(T(n/8)) + 2$
4. Substitua $T(n/2)$ pelo valor que foi isolado acima e, em seguida, o mesmo para $T(n/4)$
   - *substituindo o valor isolado de* $T(n/2)$:
     $T(n) = 2(2(T(n/4)) + 2) + 2$
     $T(n) = 2^2 T(n/2^2) + 6$
   - *agora substituindo o valor de* $T(n/4)$:
     $T(n) = 2^2(2(T(n/8) + 2) + 6$
     $T(n) = 2^3 T(n/2^3) + 2^3 + 6$
     $T(n) = 2^3 T(n/2^3) + 2^4 - 2$

Resolução por expansão telescópica

Exemplo 1: $\boxed{T(n) = 2T(n/2) + 2}$ $\boxed{T(1) = 1}$

5. Identifique a fóruma do *i*-ésimo passo

   $$T(n) = 2^i T(n/2^i) + 2^{i+1} - 2$$

6. Descubra o valor de *i* de forma a igualar o parâmetro de $T(x)$ ao parâmetro (valor de *n*) no caso base

   $$T(n/2^i) \quad \Leftrightarrow \quad T(1)$$
   $$n/2^i = 1$$
   $$n = 2^i$$
   $$i = \lg(n)$$

7. Substitua o valor de *i* na fórmula do *i*-ésimo caso

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

8. Identifique a complexidade dessa fórmula

   $$\boxed{T(n) \in \Theta(n)}$$

Resolução por expansão telescópica

Exemplo 1: $\boxed{T(n) = 2T(n/2) + 2}$ $\boxed{T(1) = 1}$

9. Prova por indução
   - **Passo base:** para $n = 1$, o resultado esperado é 1
     $T(n) = 3n - 2 \quad = \quad 3 - 2 \quad = \quad 1 \qquad$ *(correto)*
   - **Passo indutivo:** por hipótese de indução, assumimos que a fórmula está correta para $n/2$, isto é, $T(n/2) = 3\ n/2\ -2$. Então, temos que verificar se $T(n) = 3n - 2$, sabendo-se que $T(n) = 2T(n/2) + 2$ e partindo da $H.I.$ que
     $T(n/2) = 3n/2 - 2$
     $T(n) = 2\ T(n/2)\ +2$
     $T(n) = 2\ (3n/2 - 2)\ +2$
     $T(n) = 2 \cdot 3 \cdot n/2 - 2 \cdot 2 + 2$
     $T(n) = 3n - 4 + 2$
     $T(n) = 3n - 2 \qquad$ *(passo indutivo provado)*
   - Demonstrado que $2T(n/2) + 2 = 3n - 2$ para $n \geq 1$

Resolução por expansão telescópica

Exemplo 2: $\boxed{T(n) = 2T(n-1) + 1}$ $\boxed{T(1) = 1}$

(Torre de Hanoi)

1. $T(n) = 2T(n-1) + 1$ *(fórmula original)*
2. $T(n)$ está escrito em função de $T(n-1)$
3. Isole as equações para $T(n-1)$ e $T(n-2)$:

   $T(n-1) = 2T(n-2) + 1$
   $T(n-2) = 2T(n-3) + 1$
4. Substitua $T(n-1)$ pelo valor que foi isolado acima e, em seguida, o mesmo para $T(n-2)$
   - *substituindo o valor isolado de $T(n-1)$:*
     $T(n) = 2(2T(n-2) + 1) + 1$
   - *agora substituindo o valor de $T(n-2)$:*
     $T(n) = 2^2T(n-2) + 2 + 1$
     $T(n) = 2^2(2T(n-3) + 1) + 2 + 1$
     $T(n) = 2^3T(n-3) + 2^2 + 2 + 1$
     $T(n) = 2^3T(n-3) + 2^3 - 1$

Resolução por expansão telescópica

Exemplo 2: $\boxed{T(n) = 2T(n-1) + 1}$ $\boxed{T(1) = 1}$

5. Identifique a fóruma do *i*-ésimo passo

$$T(n) = 2^i T(n-1) + 2^i - 1$$

6. Descubra o valor de *i* de forma a igualar o parâmetro de $T(x)$ ao parâmetro (valor de *n*) no caso base

$$T(n-i) \quad \Leftrightarrow \quad T(1)$$
$$n - i = 1$$
$$i = n - 1$$

7. Substitua o valor de *i* na fórmula do *i*-ésimo caso

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

8. Identifique a complexidade dessa fórmula

$$\boxed{T(n) \in \Theta(2^n)}$$

Resolução por expansão telescópica

Exemplo 2: $\boxed{T(n) = 2T(n-1) + 1}$ $\boxed{T(1) = 1}$

9. Prova por indução
   - **Passo base:** para $n = 1$, o resultado esperado é 1
     $T(n) = 2^n - 1 \quad = \quad 2 - 1 \quad = \quad 1$ *(correto)*

   - **Passo indutivo:** por hipótese de indução, assumimos que a fórmula está correta para $n-1$, isto é,
     $T(n-1) = 2^{n-1} - 1$. Então, temos que verificar se
     $T(n) = 2^n - 1$, sabendo-se que $T(n) = 2^n - 1$ e partindo da *H.I.* que $T(n-1) = 2^{n-1} - 1$
     $T(n) = 2\ T(n-1) + 1$
     $T(n) = 2\ (2^{n-1} - 1) + 1$
     $T(n) = 2^n - 2 + 1$
     $T(n) = 3^n - 1$ *(passo indutivo provado)*

   - Demonstrado que $2T(n-1) + 1 = 2^n - 1$ para $n \geq 1$

Resolução por expansão telescópica

- **Exercícios** – Repita o procedimento para as seguintes equações de recorrência:

1. $T(n) = 3T(n-1) + 1$
   $T(1) = 1$

2. $T(n) = 4T(n/2) + n$
   $T(1) = 1$

Árvore de recorrência

- Permite visualizar melhor o que acontece quando a recorrência é iterada.
- É mais fácil organizar as contas.
- Útil para recorrências de algoritmos de divisão-e-conquista.

Considere a recorrência

$$\begin{array}{rcll} T(n) & = & \Theta(1) & \text{para } n = 1, 2, 3, \\ T(n) & = & 3T(\lfloor n/4 \rfloor) + cn^2 & \text{para } n \geq 4, \end{array}$$

onde $c > 0$ é uma constante.

Costuma-se (CLRS) usar a notação $T(n) = \Theta(1)$ para indicar que $T(n)$ é uma constante.

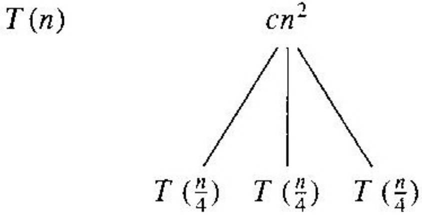
Simplificação

Vamos supor que a recorrência está definida apenas para potências de 4

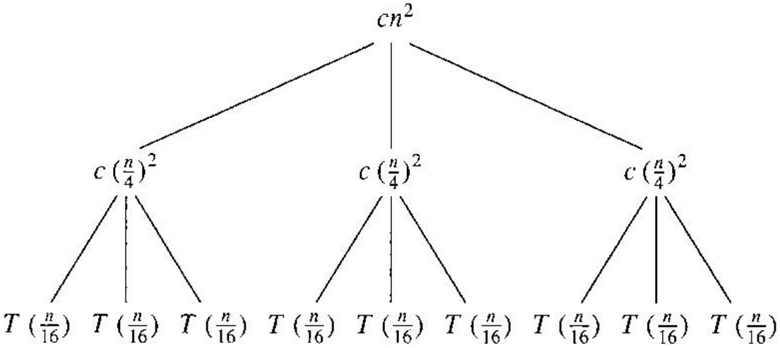
$$
\begin{aligned}
T(n) &= \Theta(1) && \text{para } n = 1,\\
T(n) &= 3T(n/4) + cn^2 && \text{para } n = 4, 16, \ldots, 4^i, \ldots.
\end{aligned}
$$

Isto permite descobrir mais facilmente a solução. Depois usamos o método da substituição para formalizar.

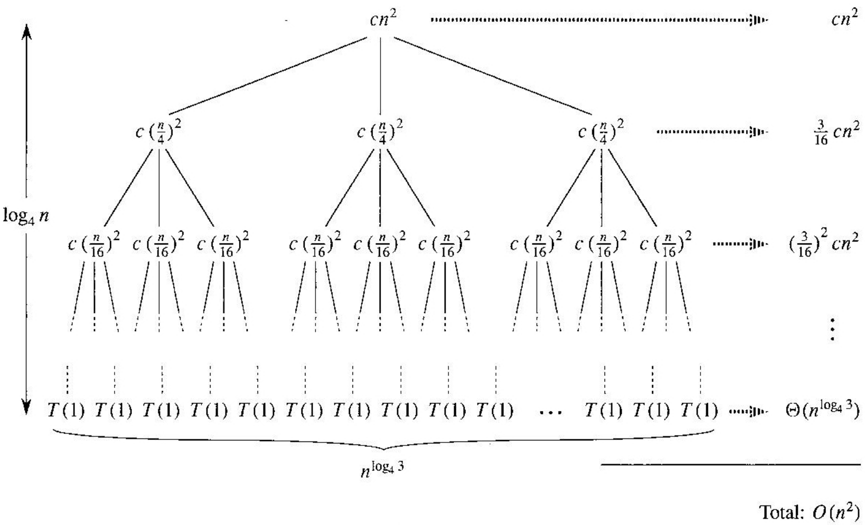
Árvore de recorrência



Árvore de recorrência



Árvore de recorrência



Árvore de recorrência

- O número de níveis é $\log_4 n + 1$.
- No nível i o tempo gasto (sem contar as chamadas recursivas) é $(3/16)^i cn^2$.
- No último nível há $3^{\log_4 n} = n^{\log_4 3}$ folhas. Como $T(1) = \Theta(1)$ o tempo gasto é $\Theta(n^{\log_4 3})$.

Logo,

$$\begin{aligned}
T(n) &= cn^2 + \frac{3}{16}cn^2 + \left(\frac{3}{16}\right)^2 cn^2 + \left(\frac{3}{16}\right)^3 cn^2 + \cdots + \\
&\quad + \left(\frac{3}{16}\right)^{\log_4 n - 1} cn^2 + \Theta(n^{\log_4 3}) \\
&= \sum_{i=0}^{\log_4 n - 1} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4 3}) \\
&\leq \sum_{i=0}^{\infty} \left(\frac{3}{16}\right)^i + \Theta(n^{\log_4 3}) = \frac{3}{16}cn^2 + \Theta(n^{\log_4 3}),
\end{aligned}$$

e $T(n) \in O(n^2)$.

Árvore de recorrência

Mas $T(n) \in O(n^2)$ é realmente a solução da recorrência original?

Com base na árvore de recorrência, chutamos que $T(n) \leq dn^2$ para alguma constante $d > 0$.

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/4 \rfloor) + cn^2 \\
&\leq 3d\lfloor n/4 \rfloor^2 + cn^2 \\
&\leq 3d(n/4)^2 + cn^2 \\
&= \frac{3}{16}dn^2 + cn^2 \\
&\leq dn^2
\end{aligned}
$$

onde a última desigualdade vale se $d \geq (16/13)c$.
(Yeeesssss!)

Árvore de recorrência

Resumo

- O número de nós em cada nível da árvore é o número de chamadas recursivas.
- Em cada nó indicamos o “tempo” ou “trabalho” gasto naquele nó que **não** corresponde a chamadas recursivas.
- Na coluna mais à direita indicamos o tempo total naquele nível que **não** corresponde a chamadas recursivas.
- Somando ao longo da coluna determina-se a solução da recorrência.

Vamos tentar juntos?

Eis um exemplo um pouco mais complicado.

Vamos resolver a recorrência

$$\begin{array}{rcll} T(n) & = & 1 & \text{para } n = 1, 2, \\ T(n) & = & T(\lceil n/3 \rceil) + T(\lfloor 2n/3 \rfloor) + n & \text{para } n \geq 3. \end{array}$$

Qual é a solução da recorrência?

Resposta: $T(n) \in O(n \lg n)$. (Resolvido em aula)

Recorrências com O à direita (CLRS)

Uma “recorrência”

$$T(n) = \Theta(1) \qquad \text{para } n = 1, 2,$$
$$T(n) = 3T(\lfloor n/4 \rfloor) + \Theta(n^2) \qquad \text{para } n \geq 3$$

representa todas as recorrências da forma

$$T(n) = a \qquad \text{para } n = 1, 2,$$
$$T(n) = 3T(\lfloor n/4 \rfloor) + bn^2 \qquad \text{para } n \geq 3$$

onde a e $b > 0$ são constantes.

As soluções exatas dependem dos valores de a e b, mas estão todas na mesma classe Θ.

A “solução” é $T(n) = \Theta(n^2)$, ou seja, $T(n) \in \Theta(n^2)$.

As mesmas observações valem para as classes O, Ω, o, ω.

Recorrência do Mergesort

Podemos escrever a recorrência de tempo do Mergesort da seguinte forma

$$\begin{array}{rcl} T(1) & = & \Theta(1) \\ T(n) & = & T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) \qquad \text{para } n \geq 2. \end{array}$$

A solução da recorrência é $T(n) = \Theta(n \lg n)$.

A prova é essencialmente a mesma do primeiro exemplo. (Exercício!)

Cuidados com a notação assintótica

A notação assintótica é muito versátil e expressiva. Entretanto, deve-se tomar alguns cuidados.

Considere a recorrência

$$\begin{aligned} T(1) &= 1 \\ T(n) &= 2T(\lfloor n/2 \rfloor) + n \qquad \text{para } n \geq 2. \end{aligned}$$

É similar a recorrência do Mergesort!

Mas eu vou “provar” que $T(n) = O(n)$!

Cuidados com a notação assintótica

Vou mostrar que $T(n) \leq cn$ para alguma constante $c > 0$.

$$\begin{aligned} T(n) &= 2T(\lfloor n/2 \rfloor) + n \\ &\leq 2c\lfloor n/2 \rfloor + n \\ &\leq cn + n \\ &= O(n) \qquad \Longleftarrow \textbf{ERRADO!!!} \end{aligned}$$

Por quê?

Não foi feito o passo indutivo, ou seja, não foi mostrado que $T(n) \leq cn$.

Teorema Master

- Veremos agora um resultado que descreve soluções para recorrências da forma

$$T(n) = aT(n/b) + f(n),$$

onde $a \geq 1$ e $b > 1$ são constantes.

- O caso base é omitido na definição e convenciona-se que é uma constante para valores pequenos.
- A expressão n/b pode indicar tanto $\lfloor n/b \rfloor$ quanto $\lceil n/b \rceil$.
- O Teorema Master **não** fornece a resposta para todas as recorrências da forma acima.

Teorema Master (Manber)

Teorema (Teorema Master (Manber))

Dada uma relação de recorrência da forma

$$T(n) = aT(n/b) + cn^k,$$

onde $a, b \in \mathbb{N}$, $a \geq 1$, $b \geq 2$, $c > 0$ *e* $k \geq 0$ *são constantes,*

$$T(n) \in \begin{cases} \Theta(n^{\log_b a}), & se\ a > b^k \\ \Theta(n^k \log n), & se\ a = b^k \\ \Theta(n^k), & se\ a < b^k \end{cases}$$

Teorema Master (Manber)

Prova: Por simplicidade, assumimos que $n = b^m$ de modo que n/b é sempre inteiro. Com isso temos

$$T(n) = aT(n/b) + cn^k$$

é equivalente a

$$T(n) = aT(b^{m-1}) + cb^{mk}$$

Vamos começar expandindo a relação de recorrência:

$$\begin{aligned}
T(n) &= aT(b^{m-1}) + cb^{mk} \\
&= a(aT(b^{m-2}) + cb^{(m-1)k}) + cb^{mk} \\
&= a^2 T(b^{m-2}) + cab^{(m-1)k} + cb^{mk} \\
&= a^3 T(b^{m-3}) + ca^2 b^{(m-2)k} + cab^{(m-1)k} + cb^{mk} \\
&= \ldots \\
&= a^m T(b^0) + ca^{m-1} b^k + ca^{m-2} b^2 k + \ldots + cab^{(m-1)k} + cb^{mk}
\end{aligned}$$

Teorema Master (Manber)

Assumindo que $T(1) = c$, ficamos com:

$$
\begin{aligned}
T(n) &= ca^m + ca^{m-1}b^k + ca^{m-2}b^{2k} + \ldots + cb^{mk} \\
&= c\sum_{i=0}^{m} a^{m-i}b^{ik} \\
&= ca^m \sum_{i=0}^{m} (b^k/a)^i.
\end{aligned}
$$

Na última linha podemos ver os casos do enunciado, com base em como séries geométricas se comprotam quando b^k/a é maior, menor ou igual a zero.

Teorema Master (Manber)

$$T(n) = ca^m \sum_{i=0}^{m} (b^k/a)^i.$$

Caso 1: $a > b^k$
Neste caso, o somatório $\sum_{i=0}^{m}(b^k/a)^i$ converge para uma constante. Daí, temos que $T(n) \in \Theta(ca^m)$. Como $n = b^m$, então $m = \log_b n$, consequentemente, $T(n) \in \Theta(n^{\log_b a})$.

Teorema Master (Manber)

$$T(n) = ca^m \sum_{i=0}^{m} (b^k/a)^i.$$

Caso 2: $a = b^k$
Como $b^k/a = 1$, temos $\sum_{i=0}^{m}(b^k/a)^i = m + 1$. Daí, temos que $T(n) \in \Theta(ca^m m)$. Como $m = \log_b n$ e $a = b^k$, então $ca^m m = cn^{\log_b a}\log_b n = cn^k \log_b n$, o que nos leva à conclusão que $T(n) \in \Theta(n^k \log_b n)$.

$$T(n) = ca^m \sum_{i=0}^{m} (b^k/a)^i.$$

Caso 3: $a < b^k$

Neste caso, a série não converge quando m vai para infinito, mas é possível calcular sua soma para um número finito de termos.

$$\begin{aligned} T(n) &= ca^m \sum_{i=0}^{m} (b^k/a)^i \\ &= ca^m \left(\frac{(b^k/a)^{m+1} - 1}{(b^k/a) - 1} \right). \end{aligned}$$

Desprezando as constantes na última linha da expressão acima e sabendo que $a^m \left(\frac{(b^k/a)^{m+1}-1}{(b^k/a)-1} \right) = b^{km}$ e $b^m = n$, concluímos que $T(n) \in \Theta(n^k)$. CQD

Teorema (Teorema Master (CLRS))

Sejam $a \geq 1$ *e* $b > 1$ *constantes, seja* $f(n)$ *uma função e seja* $T(n)$ *definida para os inteiros não-negativos pela relação de recorrência*

$$T(n) = aT(n/b) + f(n).$$

Então $T(n)$ *pode ser limitada assintoticamente da seguinte maneira:*

1. *Se* $f(n) \in O(n^{\log_b a - \epsilon})$ *para alguma constante* $\epsilon > 0$*, então* $T(n) \in \Theta(n^{\log_b a})$
2. *Se* $f(n) \in \Theta(n^{\log_b a})$*, então* $T(n) \in \Theta(n^{\log_b a} \log n)$
3. *Se* $f(n) \in \Omega(n^{\log_b a + \epsilon})$*, para alguma constante* $\epsilon > 0$ *e se* $af(n/b) \leq cf(n)$*, para alguma constante* $c < 1$ *e para* n *suficientemente grande, então* $T(n) \in \Theta(f(n))$

Resolução por Teorema Master

Exemplo 1: $\boxed{T(n) = 4T\left(\frac{n}{2}\right) + n}$

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

  $a = 4 \quad ; \quad b = 2$

  $\log_b a \;=\; \log_2 4 \;=\; 2$

  $f(n) \;=\; n$

  $f(n) \;\in\; O(n^{\log_b a - \epsilon}) \;=\; O(n^{2-\epsilon})$, sendo $\epsilon = 1$ $(\epsilon > 0)$

- Portanto, se encaixa no caso 1 do Teorema Master:

  $T(n) \;\in\; \Theta(n^{\log_b a}) \;=\; \Theta(n^2)$

Resolução por Teorema Master

Exemplo 2: $\boxed{T(n) = T\left(\frac{9n}{10}\right) + n}$

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

  $a = 1 \quad ; \quad b = \frac{10}{9} \quad ; \quad \log_b a \;=\; \log_{\frac{10}{9}} 1 \;=\; 0$

  $$\begin{aligned} f(n) &= n \\ f(n) &\in \Omega(n^{\log_b a+\epsilon}) = \Omega(n^{0+\epsilon}), \quad \text{sendo } \epsilon = 1\ (\epsilon > 0) \end{aligned}$$

- Será caso 3 se satisfizer a condição de regularidade: Para todo n, $af\left(\frac{n}{b}\right) = \frac{9n}{10} \leq \frac{9}{10}n = cf(n)$ para $c = \frac{9}{10} < 1$.

- Portanto, se encaixa no caso 3 do Teorema Master: $T(n) \in \Theta(f(n)) = \Theta(n)$

Resolução por Teorema Master

Exemplo 3: $\boxed{T(n) = 4T\left(\frac{n}{2}\right) + n^2}$

- $T(n) = aT\left(\frac{n}{b}\right) + f(n)$

  $a = 4 \quad ; \quad b = 2 \quad ; \quad \log_b a \;=\; \log_2 4 \;=\; 2$

  $$\begin{aligned} f(n) &= n^2 \\ f(n) &\in \Theta(n^{\log_b a}) = \Theta(n^2) \end{aligned}$$

- Portanto, se encaixa no caso 2 do Teorema Master:
  $T(n) \in \Theta(n^{\log_b a} \log n) = \Theta(n^2 \log n)$

Exemplos de Recorrências

Exemplos onde o Teorema Master se aplica:

- Caso 1:
  $T(n) = 9T(n/3) + n$
  $T(n) = 4T(n/2) + n \log n$
- Caso 2:
  $T(n) = T(2n/3) + 1$
  $T(n) = 2T(n/2) + (n + \log n)$
- Caso 3:
  $T(n) = T(3n/4) + n \log n$

Exemplos de Recorrências

Exemplos onde o Teorema Master **não se aplica**:

- $T(n) = T(n-1) + n$
- $T(n) = T(n-a) + T(a) + n$, ($a \geq 1$ inteiro)
- $T(n) = T(\alpha n) + T((1-\alpha)n) + n$, ($0 < \alpha < 1$)
- $T(n) = T(n-1) + \log n$
- $T(n) = 2T(\frac{n}{2}) + n \log n$