

Projeto e Análise de Algoritmos I

Pedro Hokama

- [cls] Algoritmos: Teoria e Prática (Terceira Edição) Thomas H. Cormen, Charles Eric Leiserson, Ronald Rivest e Clifford Stein.
 - [timr] Algorithms Illuminated Series, Tim Roughgarden
 - Desmistificando Algoritmos, Thomas H. Cormen.
 - Algoritmos, Sanjoy Dasgupta, Christos Papadimitriou e Umesh Vazirani
 - Stanford Algorithms
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt7Q0xr1PIAriY5623cKiH7V>
<https://www.youtube.com/playlist?list=PLXFMmlk03Dt5EMI2s2WQBsLsZl7A5HEK6>
 - Conjunto de Slides dos Professores Cid C. de Souza, Cândida N. da Silva, Orlando Lee, Pedro J. de Rezende
 - Conjunto de Slides do Professores Cid C. de Souza para a disciplina MO420
- Qualquer erro é de minha responsabilidade.

1 / 34

2 / 34

Princípios da Análise de Algoritmos

- O que fizemos no caso do MergeSort foi uma análise de Pior Caso, ou seja, qualquer que seja a entrada sabemos que o algoritmo executaram em tempo $\leq 6n \log_2 n + 6n$.
- Esse limite também é aplicado se um adversário tentasse atribuir números de forma a deixar o algoritmo lento.
- Essa análise é particularmente interessante por não precisar entender a aplicação do problema, padrões de entrada e ela é usualmente mais útil e mais fácil que as alternativas:
 - ▶ Análise de Caso Médio (Exige assumir alguma distribuição da entrada, ter conhecimento do domínio, mais difícil de ser feita)
 - ▶ Desempenho em *Benchmarks*
 - ▶ Análise de Melhor Caso (inútil na maior parte do tempo)

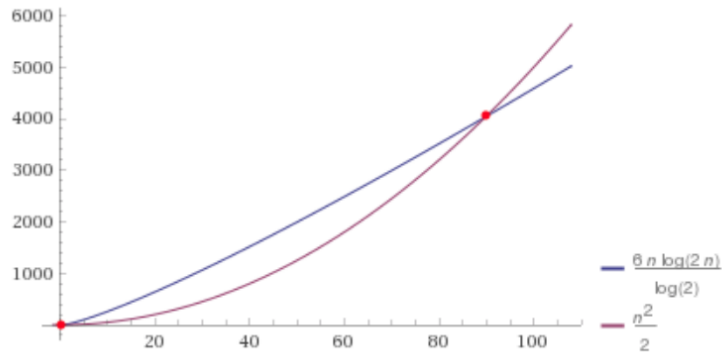
3 / 34

Princípios da Análise de Algoritmos

- Iremos fazer uma Análise **Assintótica** dos algoritmos, o que significa que estamos interessados no comportamento deles para instâncias **grandes**.
- Isso nos permite dizer que um algoritmo é mais rápido que outro assumindo que o tamanho n da instância é suficientemente grande.
- Por exemplo, podemos dizer com segurança que um algoritmo que executa em $6n \log_2 n + 6n$ é mais rápido que um que executa em $\frac{1}{2}n^2$
- Note que isso pode não ser verdade para n pequeno, mas a partir de algum $n = n_0$ sempre será verdade.
- De fato, para n pequeno, tanto faz o algoritmo que você use. Estamos interessados em resolver problemas grandes!

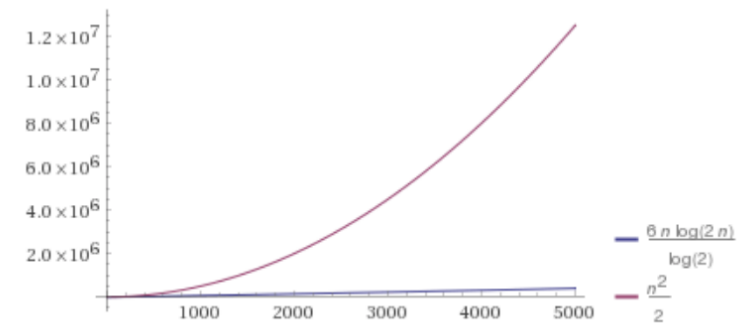
4 / 34

Plot:



5 / 34

Plot:



6 / 34

Princípios da Análise de Algoritmos

- Você poderia imaginar que com o avanço dos hardwares, bastaria eu usar um computador melhor.
- Na verdade quanto maior o poder computacional, MAIOR é a disparidade entre algoritmos eficientes.
- Podemos pensar no tamanho do problema que podemos resolver com computadores mais potentes usando diferentes algoritmos.
- Suponha que você tem dois algoritmos para um problema.

Algoritmo A	Algoritmo B
n	n^2

- Suponha que você fez um grande investimento e comprou um computador 4 vezes mais potente. Com o algoritmo A você pode resolver um problema 4 vezes maior, enquanto com o algoritmo B você só pode resolver um problema 2 vezes maior.

7 / 34

Análise Assintótica

- É a linguagem que os cientistas da computação (sérios) usam para discutir o desempenho em alto nível de algoritmos.
- É fundamental para o vocabulário do cientista da computação. Quando alguém diz “O MergeSort executa em $\Theta(n \log n)$, e o InsertionSort executa em $\Theta(n^2)$ ” o que exatamente ele está dizendo?
- A análise assintótica é uma ferramenta adequada pois:
 - ▶ É simples o bastante para suprimir detalhes de arquitetura/linguagem/compilador.
 - ▶ Mas é complexa o bastante para permitir a comparação entre diferentes algoritmos, especialmente em instâncias grandes.

8 / 34

Análise Assintótica

Ideia geral

Suprimir fatores constantes e termos de ordens inferiores.

- Por exemplo em:

$$6n \log_2 n + 6n$$

$6n$ é um termo de ordem inferior, 6 é constante então resultaria em:

$$n \log n$$

Exercício: A base do log também não importa. Por que?

- Então quando dizemos que o tempo de execução do MergeSort é $O(n \log n)$, ou de maneira geral quando dizemos que um algoritmo é $O(f(n))$. Estamos dizendo que depois de eliminar os termos de ordem inferior e constantes acabamos apenas com $f(n)$.

9 / 34

Exemplos

Exemplo de um laço

Algoritmo 1: Procura

Entrada: Um vetor A de tamanho n e um inteiro t

Saída: Verdadeiro se A contém t , Falso caso contrário

```
1 para  $k$  de 1 até  $n$  faça
2   se  $A[k] == t$  então
3     devolva Verdadeiro;
4 devolva Falso;
```

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

- O tempo de execução do exemplo depende por exemplo se t está ou não em A , e se t estiver na primeira posição? Estamos interessados no pior caso!
- Quantas operações estamos fazendo nas linhas 1 e 2? Uma, duas, três? Isso é constante e é suprimido pela notação O .

11 / 34

Análise assintótica de funções quadráticas - termos de menor ordem

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

n	$3n^2 + 10n + 50$	$3n^2$
64	12978	12288
128	50482	49152
512	791602	786432
1024	3156018	3145728
2048	12603442	12582912
4096	50372658	50331648
8192	201408562	201326592
16384	805470258	805306368
32768	3221553202	3221225472

- 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.

10 / 34

Exemplos

Exemplo de dois laços consecutivos

Algoritmo 2: Procura2

Entrada: Dois vetores A e B de tamanho n e um inteiro t

Saída: Verdadeiro se A ou B contém t , Falso caso contrário

```
1 para  $k$  de 1 até  $n$  faça
2   se  $A[k] == t$  então devolva Verdadeiro;
3 para  $k$  de 1 até  $n$  faça
4   se  $B[k] == t$  então devolva Verdadeiro;
5 devolva Falso;
```

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

- Evidentemente, nesse problema a entrada é na verdade de tamanho $2n$, então o algoritmo não seria $O(2n)$? Essa constante é suprimida na notação O .

12 / 34

Exemplos

Exemplo de dois laços aninhados

Algoritmo 3: ProcuraComum

Entrada: Dois vetores A e B de tamanho n

Saída: Verdadeiro se A ou B têm um numero em comum, Falso caso contrário

```
1 para  $j$  de 1 até  $n$  faça
2   para  $k$  de 1 até  $n$  faça
3     se  $A[j] == B[k]$  então devolva Verdadeiro;
4 devolva Falso;
```

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

- Pior caso ✓. Constantes ✓.
- Nesse caso se o tamanho dos vetores dobrar, o tempo de execução quadruplica!

13 / 34

Notação O

- Big Oh, Ôzão, O grande.
- Notação O é utilizada em funções definidas nos inteiros positivos.
- Seja $T(n)$ uma função sobre $n = 1, 2, 3, \dots$
- $T(n) : \mathbb{Z}_+^* \rightarrow \mathbb{R}$
- Pergunta: Quando podemos dizer que $T(n) = O(f(n))$?
- Resposta: Se eventualmente, para um n suficientemente grande, $T(n)$ é limitado superiormente por uma alguma constante vezes $f(n)$.

15 / 34

Exemplo de dois laços aninhados

Algoritmo 4: ProcuraComum

Entrada: Um vetor A de tamanho n

Saída: Verdadeiro se A tem números duplicados, Falso caso contrário

```
1 para  $j$  de 1 até  $n$  faça
2   para  $k$  de  $j + 1$  até  $n$  faça
3     se  $A[j] == A[k]$  então devolva Verdadeiro;
4 devolva Falso;
```

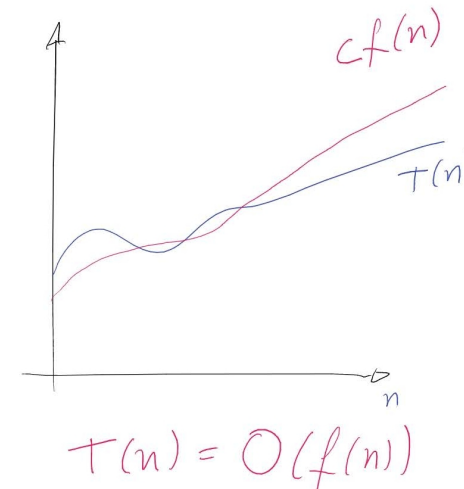
- Dessa vez, obviamente, procuramos no vetor A ao invés de B .
- Ao invés de olhar todas combinações de j e k , olhamos apenas aquelas em que $k \geq j + 1$.
- Isso é válido pois comparar, por exemplo, o primeiro com o terceiro elemento é a mesma coisa que comparar o terceiro com o primeiro. Então diminuimos nossas comparações pela metade!

Qual o tempo de execução desse algoritmo?

- a $O(1)$
- b $O(\log n)$
- c $O(n)$
- d $O(n^2)$

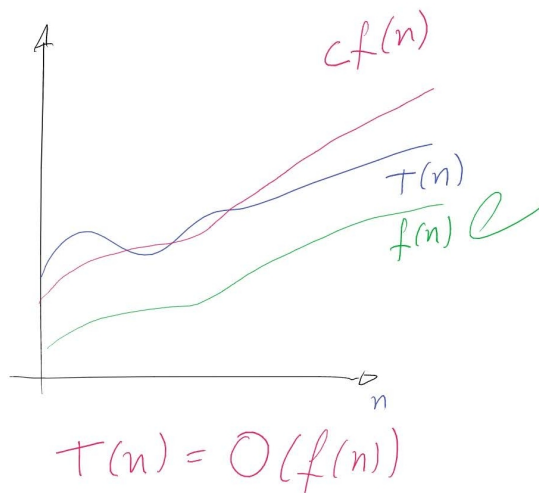
Exercício: É possível fazer esse algoritmo mais eficiente? Como?

14 / 34



- $T(n) = O(f(n))$ se quanto multiplicado por alguma constante c tal que $c \cdot f(n)$ está sempre acima de $T(n)$

16 / 34



- É válido mesmo se $f(n)$ ou $c' \cdot f(n)$ esteja sempre abaixo de $T(n)$, o importante é existir uma constante que a deixe sempre acima.

17 / 34

Exemplo

Teorema

Se $T(n) = a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0$ então $T(n) = O(n^k)$

- Para provar precisamos exigir constantes c e $n_0 > 0$ tais que $T(n) \leq c \cdot f(n)$ para todo $n \geq n_0$.
- Vamos então escolher $n_0 = 1$ e $c = |a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|$.

$$\begin{aligned}
 T(n) &= a_k n^k + a_{k-1} n^{k-1} + \dots + a_1 n + a_0 \\
 &\leq |a_k| n^k + |a_{k-1}| n^{k-1} + \dots + |a_1| n + |a_0| \\
 &\leq |a_k| n^k + |a_{k-1}| n^k + \dots + |a_1| n^k + |a_0| n^k \text{ (vál. para } n > n_0) \\
 &= (|a_k| + |a_{k-1}| + \dots + |a_1| + |a_0|) n^k \\
 &= c n^k
 \end{aligned}$$

portanto, $T(n) \leq c n^k$ logo, $T(n) = O(n^k)$ $\square$

19 / 34

Formalizando o O

Definição

$T(n) = O(f(n))$ se e somente se existem constantes $c, n_0 > 0$ tais que

$$T(n) \leq c \cdot f(n)$$

para todo $n \geq n_0$.

- Então para provar que $T(n) = O(f(n))$ você precisa exibir c e n_0 que provem que $T(n) \leq c \cdot f(n)$ para todo $n \geq n_0$.
- Dizer que c e n_0 são constantes, significa que não dependem de n . (Tudo bem se c depender de n_0 e vice versa).
- (muito) Formalmente $O(f(n))$ é um conjunto de funções que podem ser limitadas por $f(n)$, então o correto seria dizer que $T(n) \in O(f(n))$. Mas dizer que $T(n) = O(f(n))$ é um abuso de notação comumente aceito e que facilita a manipulação dessas funções.

18 / 34

Teorema

Para todo $k \geq 1$, n^k não é $O(n^{k-1})$.

- Podemos provar por contradição, ou seja, assumimos que a premissa é verdadeira porém a conclusão é falsa e chegamos a algum absurdo.
- Supomos que existe um $k \geq 1$ e constantes $c, n_0 > 0$ tal que $n^k = O(n^{k-1})$, ou seja,

$$\begin{aligned}
 n^k &\leq c n^{k-1} & \forall n \geq n_0 \\
 n n^{k-1} &\leq c n^{k-1} & \forall n \geq n_0 \\
 \cancel{n} n^{k-1} &\leq \cancel{c} n^{k-1} & \forall n \geq n_0 \\
 n &\leq c & \forall n \geq n_0
 \end{aligned}$$

Como n pode ser todos os naturais maiores do que n_0 , não pode existir tal constante c que seja maior do que qualquer natural. Portanto chegamos a um absurdo e provamos que todo $k \geq 1$, n^k não é $O(n^{k-1})$. $\square$

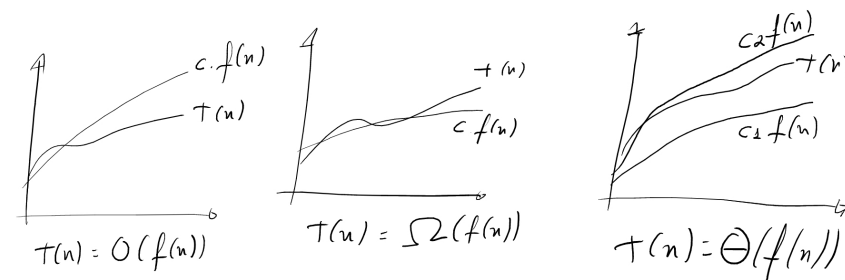
20 / 34

Analogia

- Intuitivamente, você pode comparar a notação O com uma relação de $\leq$. Já que dizer que $T(n)$ ser $O(f(n))$, quer dizer que: (existe constante c e n_0 tal que para todo n maior que n_0)

$$T(n) \leq cf(n)$$

- Nessa comparação a notação Ω seria como uma relação de $\geq$
- E a notação Θ seria como uma relação de $=$



21 / 34

Notação Ω

A notação Ω que define um limitante inferior. Formalmente:

Definição

$T(n) = \Omega(f(n))$ se e somente se existem constantes c , $n_0 > 0$ tais que

$$T(n) \geq c \cdot f(n)$$

para todo $n \geq n_0$.

Notação Θ

A notação Θ indica que uma função $T(n)$ é limitada inferiormente e superiormente por uma função $f(n)$. Formalmente:

Definição

$T(n) = \Theta(f(n))$ se e somente se existem constantes c_1 , c_2 e $n_0 > 0$ tais que

$$c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$$

para todo $n \geq n_0$.

Definição

$T(n) = \Theta(f(n))$ se e somente se $T(n) = O(f(n))$ e também se $T(n) = \Omega(f(n))$.

23 / 34

22 / 34

24 / 34

- A notação o (Little-Oh, ózinho, ó pequeno) é semelhante a notação O porém não assintoticamente justo.
- Informalmente pode-se pensar que se O está para uma relação do tipo $\leq$, enquanto o o é uma relação do tipo $<$.
- A notação ω (Little-omega, omeguinha, omega pequeno) é semelhante a notação Ω porém não assintoticamente justo.
- Informalmente pode-se pensar que Ω está para uma relação do tipo $\geq$, enquanto a ω é uma relação do tipo $>$.

25 / 34

Notação o

Definição

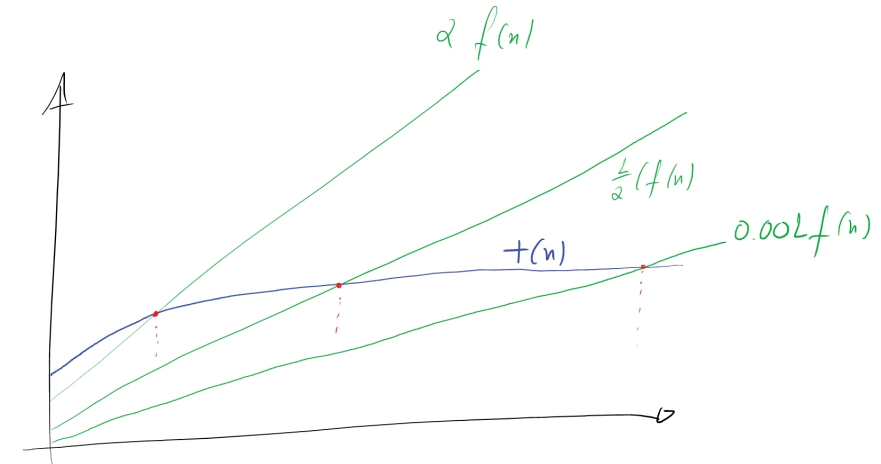
$T(n) = o(f(n))$ se e somente se para toda constante $c > 0$, existe um $n_0 > 0$ tal que

$$T(n) < c \cdot f(n)$$

para todo $n \geq n_0$.

Exercício: Provar que para todo $k \geq 1$, $n^{k-1} = o(n^k)$.

27 / 34



26 / 34

Notação ω

Definição

$T(n) = \omega(f(n))$ se e somente se para toda constante $c > 0$, existe um $n_0 > 0$ tal que

$$T(n) > c \cdot f(n)$$

para todo $n \geq n_0$.

28 / 34

Mais exemplos de Notação Assintótica

Exemplo

Provar que $2^{n+10} = O(2^n)$.

Precisamos exibir c e $n_0 > 0$ tais que

$$2^{n+10} \leq c2^n \text{ para todo } n \geq n_0.$$

Como escolher c e n_0 ?

$$2^{n+10} \leq c2^n$$

$$2^{10}2^n \leq c2^n$$

$$1024 \cdot 2^n \leq c2^n$$

Quando isso é verdade?

Escolhemos então algum $c \geq 1024$, e $n_0 \geq 1$.

29 / 34

Exemplo

Provar que 2^{10n} não é $O(2^n)$.

Suponha por absurdo (por contradição) que $2^{10n} = O(2^n)$ e portanto $2^{10n} \leq c2^n$ para alguma constante c e $n \geq n_0$. Então

$$2^{10n} \leq c2^n$$

$$2^{9n}2^n \leq c2^n$$

$$2^{9n}2^n \leq c2^n$$

$$2^{9n} \leq c$$

obviamente não existe uma constante que seja maior que 2^{9n} para todos os naturais n , portanto chegamos a uma contradição. Logo 2^{10n} não pode ser $O(2^n)$. $\square$

31 / 34

Exemplo

Provar que $2^{n+10} = O(2^n)$.

Precisamos exibir c e $n_0 > 0$ tais que

$$2^{n+10} \leq c2^n \text{ para todo } n \geq n_0.$$

Escolhemos então $c = 1024$, e $n_0 = 1$ e vamos mostrar que:

$$2^{n+10} \leq 1024 \cdot 2^n \text{ para todo } n \geq 1$$

$$2^{10}2^n \leq 1024 \cdot 2^n \text{ para todo } n \geq 1$$

$$1024 \cdot 2^n \leq 1024 \cdot 2^n \text{ para todo } n \geq 1$$

logo $2^{n+10} = O(2^n)$. $\square$

30 / 34

Exemplo

Para quaisquer dois pares de funções positivas, $f(n)$ e $g(n)$, provar que $\max\{f(n), g(n)\} = \Theta(f(n) + g(n))$.

Para mostrar que a afirmação é verdadeira, podemos escolher $c_1 = 1/2$, $c_2 = 1$ e $n_0 = 1$ e verificamos que:

$$\frac{1}{2}(f(n) + g(n)) \leq \max\{f(n), g(n)\} \leq f(n) + g(n) \quad \forall n \geq n_0$$

32 / 34

Usando limites

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

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

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

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

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

Exemplo

$$T(n) = \ln n \text{ e } f(n) = n^e.$$

$$\lim_{n \rightarrow \infty} \frac{\ln n}{n^e} = \lim_{n \rightarrow \infty} \frac{1/n}{e \cdot n^{e-1}} = 0.$$

portanto $\ln n = o(n^e)$. Obviamente $\ln n = O(n^e)$ também.