Fundamentos da Análise da Eficiência de Algoritmos

Notação Assintótica

Aula 2
Alessandro L. Koerich

Pontifícia Universidade Católica do Paraná (PUCPR)

Ciência da Computação – 7º Período
Engenharia de Computação – 5º Período

Programa do PA

Introdução
- 1. Resolução de Problemas e Tipos de Problemas

Fundamentos da Análise da Eficiência de Algoritmos
- 2. Fundamentos
- 3. Notação Assintótica e Classe de Eficiência
- 4. Análise Matemática de Algoritmos
- 5. Análise Empírica de Algoritmos

Técnicas de Projeto de Algoritmos
- 6. Força Bruta
- 7. Dividir & Conquistar
- 8. Decrementar & Conquistar
- 9. Transformar & Conquistar
- 10. Compromisso Tempo-Espaço
- 11. Programação Dinâmica
- 12. Estratégia Gulosa
- 13. *Backtracking* & *Branch and Bound*
- 14. Algoritmos Aproximados

Limitações
- 15. Teorema do Limite Inferior
- 16. Árvores de Decisão
- 17. Problemas P, NP e NPC

Plano de Aula

- Introdução
- Estrutura da Análise de Algoritmos
- Crescimento de Funções e Notação Assintótica
- Comparação de Funções
- Resumo

Introdução

- Como avaliar um algoritmo?
- Dado um algoritmo, podemos perguntar se:
  - Ele de fato fornece uma solução para o problema em questão ?
  - Quão eficiente é o algoritmo ?
  - Qual o espaço em memória necessário para a execução?
  - Existem maneiras melhores de se resolver o problema?

Introdução

- Assim, um dos principais objetivos é o de desenvolver habilidades para avaliar algoritmos
- Existem vários critérios sobre os quais podemos avaliar um algoritmo.
- Quais são eles?

Análise de Algoritmos

- Critérios
  - Correção
  - Eficiência temporal
  - Eficiência espacial
  - Otimalidade
- Dois métodos:
  - Análise teórica
  - Analise empírica

Introdução

Abordagem Experimental

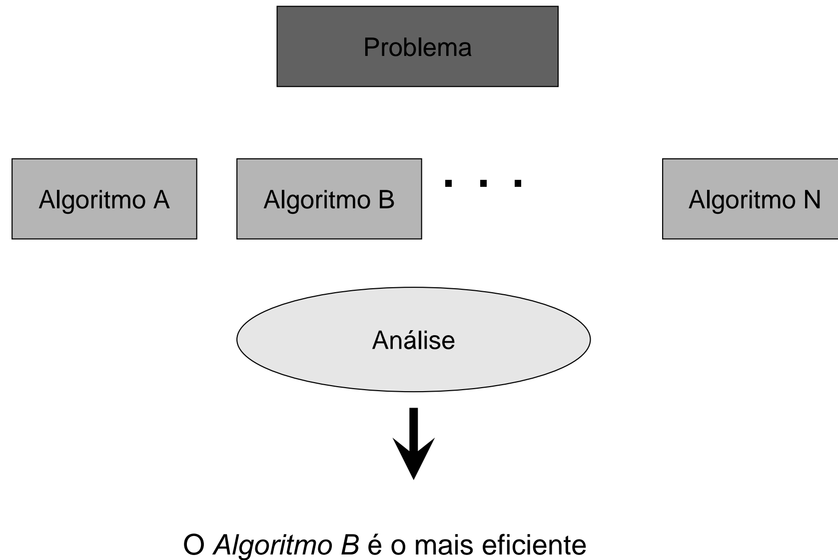
- Implementar o algoritmo e executar o programa para um conjunto de dados de teste
- Porém
  - não podemos testar todas as possíveis entradas.
  - podemos esquecer algum caso em que o algoritmo falha ou caso em que o desempenho do algoritmo é particularmente bom ou ruim
  - Os resultados dependem de aspectos de implementação

Introdução

Abordagem Teórica

- Estudar o algoritmo em termos gerais e tentar prever aspectos gerais do seu comportamento:
  - Correção: Ele fornece uma solução válida para o problema ?
  - Eficiência: Quanto tempo ele gasta ? Quanto de memória ele usa ?

Introdução



Análise de Algoritmos

- Analisar = Prever os recursos que um algoritmo necessitará. Exemplo:
  - Memória
  - Largura de banda de comunicação
  - Hardware
- Objetivo: Prever o comportamento, especialmente o tempo de execução sem implementá–lo em uma plataforma específica

Estrutura da Análise

- Como analisar a eficiência de algoritmos?
- Existem dois tipos de eficiência: **eficiência temporal** e **eficiência espacial**.
  - A **eficiência temporal** indica quão rápido um algoritmo em questão é executado.
  - A **eficiência espacial** está relacionada com o espaço extra que o algoritmo necessita.

Estrutura da Análise

- Antigamente, ambos recursos – tempo e espaço – eram valiosos.
- Atualmente, a quantidade extra de espaço requerida por um algoritmo não é tão importante, ainda que exista uma diferença entre memória principal, secundária e *cache*.
- Porém, o tempo continua sendo importante, pois, problemas cada vez mais complexos são tratados → abordaremos somente eficiência temporal

Tamanho da Entrada

- Quase todos os algoritmos levam mais tempo para ser executados sobre entrada maiores.
- Assim, é obvio investigar a eficiência de algoritmos como função do parâmetro ***n*** que indica o tamanho da entrada do algoritmo.

Tempo de Execução?

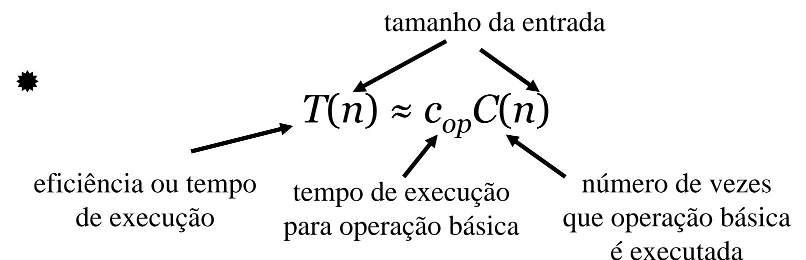
- Por que não utilizar unidade "física" de tempo?
  - Dependência do *hardware*, qualidade da implementação, compilador, etc.
- Métrica que não dependa de fatores externos. Alternativas:
  - Contar o número de vezes que cada operação do algoritmo é realizada
  - Identificar a operação mais importante do algoritmo (operação básica), a operação que mais contribui para o tempo de execução total e contar o número de vezes que esta operação é realizada.

Eficiência de Algoritmos

- Geralmente, a operação que consome mais tempo está no laço (*loop*) mais interno do algoritmo.

Eficiência de Algoritmos

- Eficiência temporal é analisada determinando o número de repetições de operações básicas com uma função do tamanho da entrada
- Operação básica: a operação que contribui mais para o tempo de execução do algoritmo.
-



Eficiência de Algoritmos

- Atenção, pois, $C(n)$ não contém informação sobre operações que não são básicas.
- $C(n)$ é calculado com aproximação
- A constante c_{op} também é uma aproximação.

Eficiência de Algoritmos

- Na análise da eficiência de algoritmos, ignoramos constantes multiplicativas e nos concentramos na ordem de crescimento da contagem da operação básica.

Tamanho da Entrada e Operação Básica

Problema	Medida do Tamanho da Entrada	Operação Básica
Busca por uma chave em uma lista de n itens	Número de itens na lista	Comparação de chaves
Multiplicação de duas matrizes de números de pontos flutuantes	Dimensões das matrizes	Multiplicação de Ponto Flutuante
Calcular a^n	n	Multiplicação de Ponto Flutuante
Grafos	Número de vértices e/ou arestas	Visitar um vértice ou atravessar uma aresta

Ordens 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}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

- A função logarítmica cresce mais lentamente
- As funções exponenciais e fatoriais crescem rapidamente

Crescimento de Funções

- Ordem de crescimento do tempo de execução de um algoritmo
  - Caracterização simples da eficiência do algoritmo
  - Comparar o desempenho relativo de algoritmos alternativos
- Em geral podemos determinar o tempo de execução exato de um algoritmo, porém.... a precisão extra não vale o esforço de calculá–lo.

Crescimento de Funções

Para entradas grandes:

- As constantes multiplicativas e
- Os termos de mais baixa ordem

são dominados pelos efeitos do tamanho da entrada

Melhor Caso, Caso Médio, Pior Caso

- Para alguns algoritmos, eficiência depende não somente do tipo da entrada, mas também das especificidades de uma entrada em particular.
- Exemplo: Algoritmo Busca Seqüencial

Exemplo: Busca Seqüencial

- Problema: Dado uma lista de n elementos e um chave de busca K, encontre um elemento igual a K, se presente.
- Algoritmo: Varrer a lista e comparar os elementos sucessivos com K até encontrar um elemento similar (busca bem sucedida) ou percorrer toda a lista (busca mal sucedida)
  - Pior caso ?
  - Melhor caso ?
  - Caso médio ?

Algoritmo Busca Seqüencial

```
ALGORITHM SequentialSearch(A[0..n − 1], K)
//Searches for a given value in a given array by sequential search
//Input: An array A[0..n − 1] and a search key K
//Output: Returns the index of the first element of A that matches K
//        or −1 if there are no matching elements
i ← 0
while i < n and A[i] ≠ K do
    i ← i + 1
if i < n return i
else return −1
```


Melhor Caso, Caso Médio, Pior Caso

Para alguns algoritmos, eficiência depende do tipo da entrada:

- Pior Caso: $W(n)$ – máximo sobre entradas de tamanho n
- Melhor Caso: $B(n)$ – mínimo sobre entradas de tamanho n
- Caso Médio: $A(n)$ – "média" sobre entradas de tamanho n
  - Número de vezes que a operação básica será executada sobre entrada típica
  - Não é a média entre pior melhor casos
  - Número esperado de repetições das operações básicas

Notação Assintótica

- A análise da eficiência de algoritmos se concentra na ordem de crescimento da operação básica de um algoritmo, como o principal indicador de sua eficiência.
- Para comparar e classificar estas ordens de crescimento, são utilizadas notações assintóticas.
- Objetivo: Resumir o comportamento de um algoritmo em fórmulas simples e de fácil compreensão

Taxa de Crescimento Assintótica

Um modo de comparar funções, que ignora fatores constantes e entradas de tamanho pequeno

- $O(g(n))$: classe de funções $f(n)$ que crescem não mais rapidamente que $g(n)$
- $\Theta(g(n))$: classe de funções $f(n)$ que cresce na mesma taxa que $g(n)$
- $\Omega(g(n))$: classe de funções $f(n)$ que crescem pelo menos tão rapidamente quanto $g(n)$

Notação Assintótica

- Eficiência Assintótica
  - Maneira como o tempo de execução de um algoritmo aumenta com o tamanho da entrada no limite, a medida que a entrada aumenta indefinidamente
- Em geral:
  - Um algoritmo que é assintoticamente mais eficiente será a melhor escolha para todas as entradas (exceto as muito pequenas)

Notação Assintótica

- Notação Θ (Theta)
- Notação O (O maiúsculo)
- Notação Ω (Ômega maiúsculo)
- Notação o (o minúsculo)
- Notação w (ômega minúsculo)

Notação Θ

- Para uma dada função $g(n)$ denotamos por $\Theta(g(n))$ o conjunto de funções:

  $\Theta(g(n))=\{f(n)$: 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 $n \geq n_0\}$

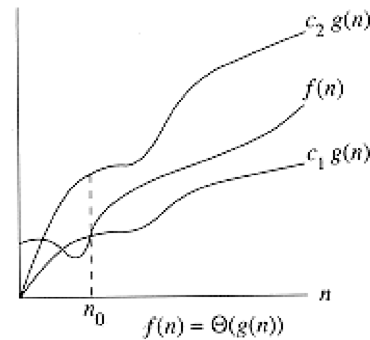
- Uma função $f(n)$ pertence ao conjunto $\Theta(g(n))$ se existem constantes positivas c_1 e c_2 tais que ela possa ser "imprensada" entre $c_1 g(n)$ e $c_2 g(n)$, para um valor de n suficientemente grande.

Notação Θ

- Como $\Theta(g(n))$ é um conjunto, poderíamos escrever "$f(n) \in \Theta(g(n))$" para indicar que $f(n)$ é um membro de (ou pertence a) $\Theta(g(n))$
- Em vez disso, geralmente escreveremos $f(n) = \Theta(g(n))$

Notação Θ

- A figura abaixo apresenta as funções $f(n)$ e $g(n)$ onde $f(n) = \Theta(g(n))$



- Para todos os valores de n à direita de n_o, o valor de $f(n)$ reside em $c_1 g(n)$ ou acima dele, e em $c_2 g(n)$ ou abaixo deste valor

Notação Θ

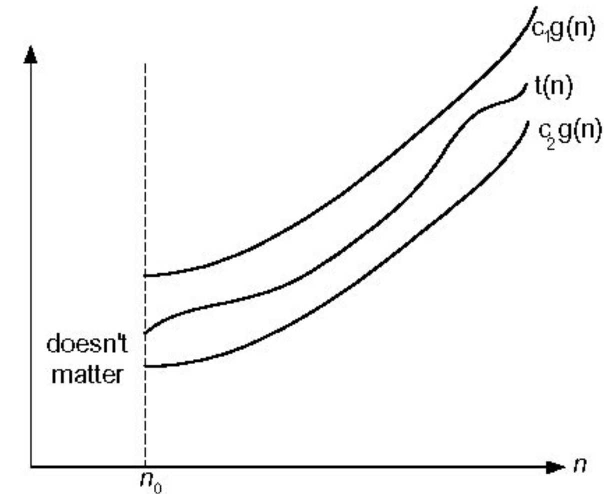


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Notação Θ

- Em outras palavras, para todo $n \geq n_o$, a função $f(n)$ é igual a $g(n)$ dentro de um fator constante.
- Dizemos que $g(n)$ é um limite assintóticamente restrito para $f(n)$.
- A definição de $\Theta(g(n))$ exige que todo membro $f(n) \in \Theta(g(n))$ seja assintóticamente não negativo, isto é, que $f(n)$ seja não negativo sempre que n for suficientemente grande.

Notação Θ

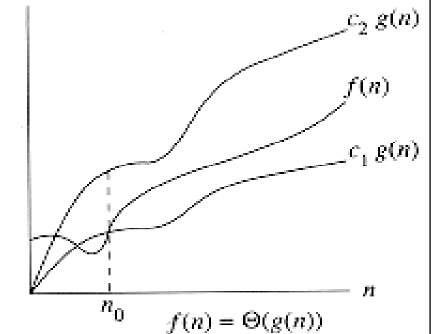
- Os termos de mais baixa ordem de uma função assintoticamente positiva podem ser ignorados na determinação de limites assintoticamente restritos.
- Eles são insignificantes para grandes valores de n.

Notação Θ

- Qualquer constante é um polinômio de grau 0
- Qualquer função constante pode ser expressa como $\Theta(n^0)$ ou $\Theta(1)$

Notação *O*

- Como vimos, a notação Θ limita assintoticamente uma função acima e abaixo.



- Quando temos apenas um limite assintótico superior, usamos a notação *O*.

Notação *O*

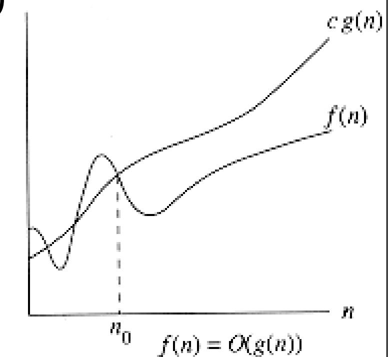
- Para uma dada função $g(n)$ denotamos por $O(g(n))$ o conjunto de funções:

  $O(g(n))=\{f(n)$: existem constantes positivas c e n_o tais que $0 \le f(n) \le c\,g(n)$ para todo $n \ge n_o\}$

- Usamos a notação *O* para dar um limite superior sobre uma função, dentro de um fator constante.

Notação *O*

- A figura abaixo apresenta as funções $f(n)$ e $g(n)$ onde $f(n) = O(g(n))$



- Para todos os valores de n à direita de n_o, o valor de $f(n)$ está em ou abaixo de $g(n)$.

Notação O

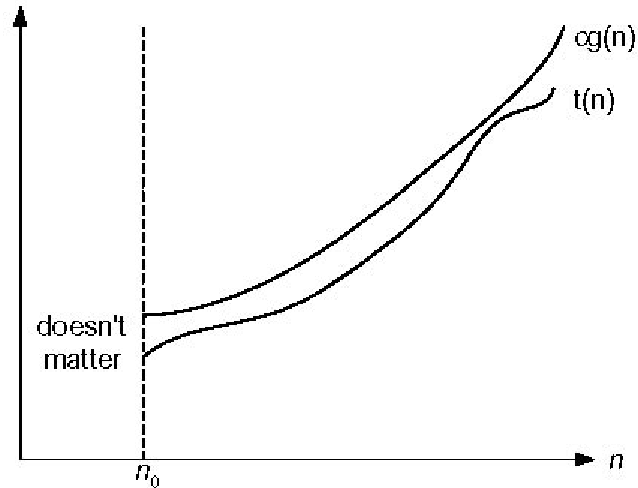


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

Notação O

- Para indicar que uma função $f(n)$ é um membro de $O(g(n))$, escrevemos $f(n) = O(g(n))$
- Note que $f(n) = \Theta(g(n))$ implica $f(n) = O(g(n))$, pois a notação Θ é uma noção mais forte que a notação O.

Notação O

- Quando escrevemos $f(n) = O(g(n))$, estamos simplesmente afirmando que:

  algum múltiplo constante de $g(n)$ é um limite assintótico superior sobre $f(n)$ sem qualquer menção sobre o quanto um limite superior é restrito.

Notação O

- Usando a notação O podemos descrever a estrutura de um algoritmo apenas inspecionando sua estrutura global.
- Ex: Algoritmo de Ordenação por Inserção
  - A estrutura de *loop* duplamente aninhado produz um limite superior $O(n^2)$ sobre o tempo de execução do pior caso.

Notação Ω

- Como vimos, a notação O fornece um limite assintótico superior sobre uma função.
- A notação Ω fornece um limite assintótico inferior.

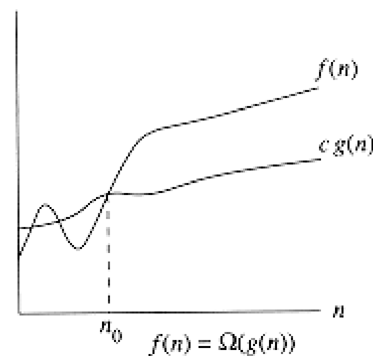
Notação Ω

- Para uma dada função $g(n)$ denotamos por $\Omega(g(n))$ o conjunto de funções:

  $\Omega(g(n)) = \{ f(n)$: existem constantes positivas c e n_o tais que $0 \le cg(n) \le f(n)$ para todo $n \ge n_o \}$

Notação Ω

- A figura abaixo apresenta as funções $f(n)$ e $g(n)$ onde $f(n) = \Omega(g(n))$



- Para todos os valores de n à direita de n_o, o valor de $f(n)$ está em ou acima de $g(n)$.

Notação Ω

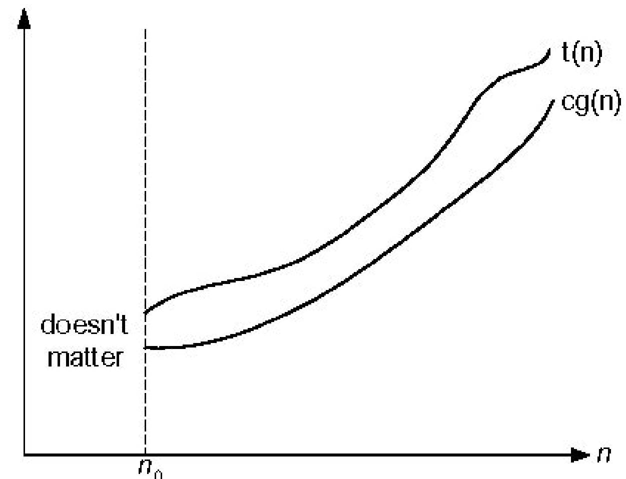
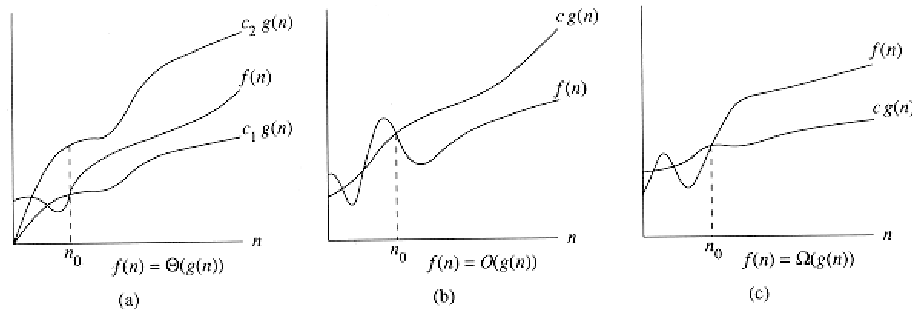


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

Notação Ω

- TEOREMA: Para duas funções quaisquer $f(n)$ e $g(n)$, temos $f(n) = \Theta(g(n))$ se e somente se $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$.



Notação Ω

- Considerando–se que a notação Ω descreve um limite inferior, quando a usamos para limitar o tempo de execução do melhor caso de um algoritmo, por implicação também limitamos o tempo de execução do algoritmo sobre entradas arbitrárias.
- Ex: o tempo de execução no melhor caso da ordenação por inserção é $\Omega(n)$ → o tempo de execução da ordenação por inserção é $\Omega(n)$.

Notação Ω

- Assim, o tempo de execução da ordenação por inserção recai entre $\Omega(n)$ e $O(n^2)$.
- Quando afirmamos que o tempo de execução de um algoritmo é $\Omega(g(n))$, queremos dizer que:
  - independentemente da entrada específica de tamanho n escolhida para cada valor de n, o tempo de execução sobre esta entrada é pelo menos uma constante vezes $g(n)$, para um valor de n suficientemente grande.

Notação *o*

- A notação *o* indica um limite superior que não é assintóticamente restrito.
- Para uma dada função $g(n)$ definimos $o(g(n))$ o conjunto de funções:

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

Notação o

- As definições da notação O e da notação o são semelhantes. A principal diferença é que em $f(n)=O(g(n))$, o limite $0 \leq f(n) \leq c\,g(n)$ se mantém válido para alguma constante $c > 0$,
- Mas, em $f(n)=o(g(n))$, o limite $0 \leq f(n) < c\,g(n)$ é válido para todas as constantes $c > 0$.
- Note que isto é exatamente o mesmo que a definição de O, exceto que "alguma constante" foi trocado para "para todas".

Notação o

- Intuitivamente, na notação o, a função $f(n)$ se torna insignificante em relação a $g(n)$ à medida que n se aproxima do infinito, isto é:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Notação ω

- A notação ω indica um limite inferior que não é assintóticamente restrito.
- Para uma dada função $g(n)$ definimos $\omega(g(n))$ o conjunto de funções:

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

Notação ω

- Por exemplo, $n^2/2 = \omega(n)$, mas $n^2/2 \neq \omega(n^2)$. A relação implica $f(n)=\omega(g(n))$ implica que:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \infty$$

se o limite existe. Isto é, $f(n)$ se torna arbitrariamente grande em relação a $g(n)$ à medida que n se aproxima do infinito.

Comparação de Funções

- Muitas das propriedades relacionais de números reais também se aplicam a comparações assintóticas:
  - Transitividade
  - Reflexividade
  - Simetria:
  - Simetria de Transposição

Comparação de Funções

- Pelo fato destas propriedade se manterem válidas para notações assintóticas ⇨ analogia entre a comparação assintótica de duas funções f e g e a comparação de dois números reais a e b:

$$f(n)=O(g(n)) \approx a \le b,$$
$$f(n)=\Omega(g(n)) \approx a \ge b,$$
$$f(n)=\Theta(g(n)) \approx a = b,$$
$$f(n)=o(g(n)) \approx a < b,$$
$$f(n)=\omega(g(n)) \approx a > b.$$

Comparação de Funções

- Dizemos que $f(n)$ é assintóticamente menor que $g(n)$ se $f(n) = o(g(n))$, pois

$f(n) = o(g(n)) \approx a < b$

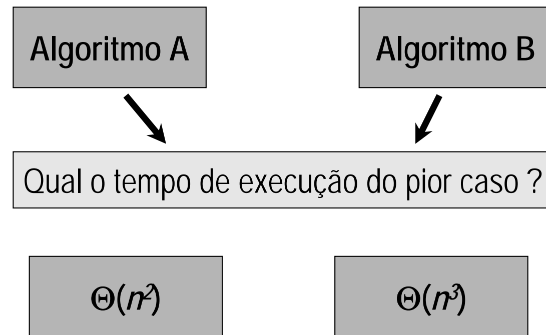
- Dizemos que $f(n)$ é assintóticamente maior que $g(n)$ se $f(n)=\omega(g(n))$, pois

$f(n) = \omega(g(n)) \approx a > b$

Funções Assintóticas Básicas

1	Constante
$\log n$	Logarítmica
n	Linear
$n \log n$	$n \log n$
n^2	Quadrática
n^3	Cúbica
2^n	Exponencial
$n!$	Fatorial

Resumo



Qual apresenta uma ordem de crescimento mais baixa ?

Atenção: esta avaliação pode ser incorreta para entradas pequenas !!!

Resumo

- Tanto eficiência temporal quanto espacial são medidas como funções do tamanho da entrada do algoritmo
- Eficiência temporal é medida contando o número de vezes que a operação básica do algoritmo é executada
- Eficiência espacial é medida contando o número de unidades de memória consumidas pelo algoritmo

Resumo

- As eficiências de alguns algoritmos podem ser significativamente diferentes para entradas do mesmo tamanho
- Para tais algoritmos precisamos distinguir entre as eficiências no pior caso, no caso médio e no melhor caso.
- O interesse principal é na ordem de crescimento do tempo de execução do algoritmo a medida que o tamanho da entrada tende a infinito.

Resumo

- As notações Θ, Ω e O são usadas para indicar e comparar as ordens de crescimento assintóticos de funções que expressam eficiências de algoritmos.
- As eficiências de um grande número de algoritmos recaem nas classes: *constante, logarítmica, linear, n–log–n, quadrática, cúbica* e *exponencial.*