ACH2002

Aula 4

Análise assintótica

Aula passada

- Problema, algoritmo e programa
- Prova de corretude de algoritmos iterativos:
  – Prova por indução de invariantes
- Análise de complexidade (ex: tempo)
  – Testes empíricos
  – Análise numérica do insertion-sort

Prova por indução

Ex: prova por indução que $1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Base: P é verdadeira para n = 1: $1 = \frac{1(1+1)}{2}$

Hipótese: P é verdadeira para um dado n qualquer: $1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Passo: Dado que P vale para n, P é verdadeira para n+1:

$$
\begin{aligned}
1 + 2 + 3 + \cdots + n + n + 1 &= \frac{n(n+1)}{2} + n + 1 \\
&= \frac{n^2 + n + 2n + 2}{2} \\
&= \frac{n^2 + 3n + 2}{2} \\
&= \frac{(n+1)(n+2)}{2}
\end{aligned}
$$

Note que um loop é uma série...

Prova de corretude por indução

```
int Max (int v[], int n) {
   int j, x;
   x = v[0];
   for (j = 1; j < n; j++)
      /* x é um elemento máximo de v[0..j-1] */
      if (x < v[j])   x = v[j];
   return x;
}
```

x, o valor retornado, é o elemento máximo do vetor

Base: no início da primeira iteração (j = 1), x é o elemento máximo de v[0]

Hipótese: assuma que é verdadeiro para um j < n, que x é o elemento máximo de v[0..j-1]

Passo da indução: se x é o elemento máximo de v[0..j-1] no início da iteração para o valor j, então na próxima instrução (que é única no loop):

- se x < v[j], x será substituído por v[j], o que o torna o elemento máximo de v[0,j] no início da próxima iteração (valor j+1)

- se x >= v[j], x não mudará de valor, pois continua sendo o elemento máximo de v[0,j] no início da próxima iteração (valor j+1)

Aula passada

- Problema, algoritmo e programa
- Prova de corretude de algoritmos iterativos:
  – Prova por indução de invariantes
- Análise de complexidade (ex: tempo)
  – Testes empíricos
  – Análise numérica do insertion-sort

Aula passada

- Problema, algoritmo e programa
- Prova de corretude de algoritmos iterativos:
  – Prova por indução de invariantes
- Análise de complexidade (ex: tempo)
  – Testes empíricos
  – Análise numérica do insertion-sort

Função de custo de um algoritmo

- engloba o custo de tempo de cada instrução e o número de vezes que cada instrução é executada

- Exemplo: *insertion-sort(A)* (entrada: array A que tem tamanho n)

	custo	vezes
para j = 2 **até** tamanho[A] **faça**	c_1	n
chave = A[j] // "número a inserir"	c_2	n−1
// ordenando elementos à esquerda	0	n−1
i = j - 1	c_4	n−1
enquanto i > 0 **e** A[i] > chave **faça**	c_5	$\sum_{j=2}^{n} t_j$
A[i+1] = A[i]	c_6	$\sum_{j=2}^{n} (t_j - 1)$
i = i -1	c_7	$\sum_{j=2}^{n} (t_j - 1)$
fim enquanto		
A[i+1] = chave	c_8	n−1
fim para		

t_j – número de vezes que a linha é executada para um dado j (depende do j)

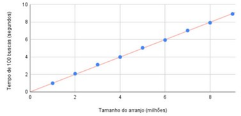
- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

		custo	vezes
1	**para** j = 2 **até** tamanho[A] **faça**	c_1	n
2	chave = A[j]	c_2	$n-1$
3	// ordenando elementos à esquerda	0	$n-1$
4	i = j - 1	c_4	$n-1$
5	**enquanto** i > 0 **e** A[i] > chave **faça**	c_5	$\sum_{j=2}^{n} t_j$
6	A[i+1] = A[i]	c_6	$\sum_{j=2}^{n}(t_j-1)$
7	i = i -1	c_7	$\sum_{j=2}^{n}(t_j-1)$
8	**fim enquanto**		
9	A[i+1] = chave	c_8	$n-1$
10	**fim para**		

Melhor caso: vetor já ordenado (A[i] ≤ chave na linha 5 ➔ $t_j=1$ para $j=2,3,\ldots,n$)

$T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) = (c_1+c_2+c_4+c_5+c_8)n - (c_2+c_4+c_5+c_8)$

Tempo de execução, neste caso, pode ser expresso como *an + b* para constantes **a** e **b** que dependem dos custos de instrução c_i ➔ **função linear** de *n*



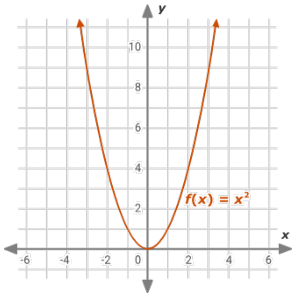
- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8 (n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ t_j=j para j=2,3,...,n)

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

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n \; -(c_2+c_4+c_5+c_8)$$



- Tempo de execução, neste caso, pode ser expresso como *$an^2 + bn + c$* para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de *n*

Exercícios (Indução matemática)

1. Prove que $1^2 + 2^2 + 3^2 + ... + n^2 = (2n^3 + 3n^2 + n)/6$, $\forall n \geq 1$
2. Prove que $1 + 3 + 5 + ... + 2n - 1 = n^2$, $\forall n \geq 1$
3. Prove que $1^3 + 2^3 + 3^3 + ... + n^3 = (n^4 + 2n^3 + n^2)/4$, $\forall n \geq 1$
4. Prove que $1^3 + 3^3 + 5^3 + ... + (2n - 1)^3 = 2n^4 – n^2$, $\forall n \geq 1$
5. Prove que $1 + 2 + 2^2 + 2^3 + ... + 2^n = 2^{n+1} - 1$, $\forall n \geq 0$
6. Prove que $2^n \geq n^2$; , $\forall n \geq 4$
7. Prove que $\frac{1}{1} - \frac{1}{2} + \frac{1}{3} - \ldots + \frac{1}{2n-1} - \frac{1}{2n} = \frac{1}{n+1} + \frac{1}{n+2} + \ldots + \frac{1}{2n}$
8. Prove que a soma dos cubos de três números naturais positivos sucessivos é divisível por 9.
9. Prove que todo número natural n > 1 pode ser escrito como o produto de primos (indução forte).
10. Prove que todo número natural positivo pode ser escrito como a soma de diferentes potências de 2 (indução forte).

Desculpem, os dois últimos ainda não estudamos…

Conseguiram fazer os demais?

Aula de hoje

- Análise assintótica (de complexidade)
- Análise formal (matemática) de algoritmos

Análise assintótica

- Primeiro: consciência que a complexidade (tempo, memória, etc) normalmente depende do tamanho da entrada

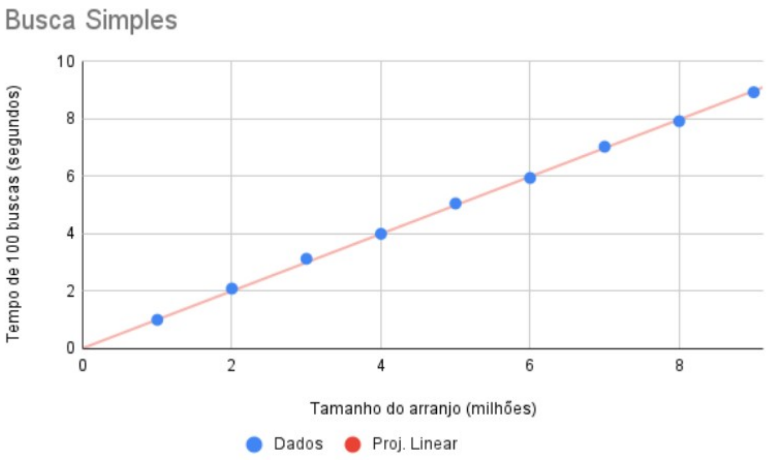
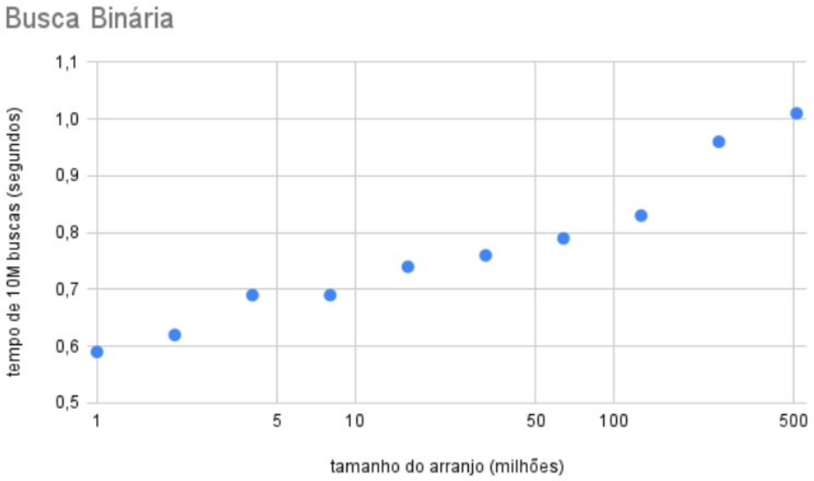


Figura 2.2: Gráfico ilustrando a hipótese de que o tempo de processamento da busca sequencial segue a função linear $t(x) = 0,997x$.



$$t(x) = a.log_2(x) + b = 0,043.log_2(x) + 0,529$$

Análise de algorimos

- Em geral:
  - tempo de execução de um algoritmo é fixo para uma determinada entrada
  - analisamos apenas o **pior caso** dos algoritmos:
    - é um limite superior sobre o tempo de execução de qualquer entrada;
    - pior caso ocorre com muita frequência para alguns algoritmos. Exemplo: registro inexistente em um banco de dados;
    - muitas vezes, o ***caso médio*** é quase tão ruim quanto o pior caso

- Nas análises anteriores, foram feitas algumas simplificações em relação às constantes, chegando à função linear e à função quadrática
- **Taxa de crescimento** ou **ordem de crescimento**:
  - considera apenas o termo inicial de uma fórmula (exemplo: an^2), pois os termos de mais baixa ordem são relativamente insignificantes para grandes valores de ***n***;
  - ignora o coeficiente constante do termo inicial (a) também por ser menos significativo para grandes entradas;
  - Portanto, dizemos que: a ordenação por inserção, por exemplo, tem um tempo de execução do pior caso igual a $\Theta(n^2)$ (*lê-se "theta de n ao quadrado"*);
  - Em geral, consideramos um algoritmo mais eficiente que outro se o tempo de execução do seu pior caso apresenta uma ordem de crescimento mais baixa.

Complexidade? Assintótica?

- **Complexidade**

  (*cs*) *sf* (*complexo+dade*) Qualidade do que é complexo.

  - **Complexo**

    (*cs*) *adj* (*lat complexu*) **1** Que abrange ou encerra muitos elementos ou partes. **2** Que pode ser considerado sob vários pontos de vista. **3** Complicado.

Complexidade? Assintótica?

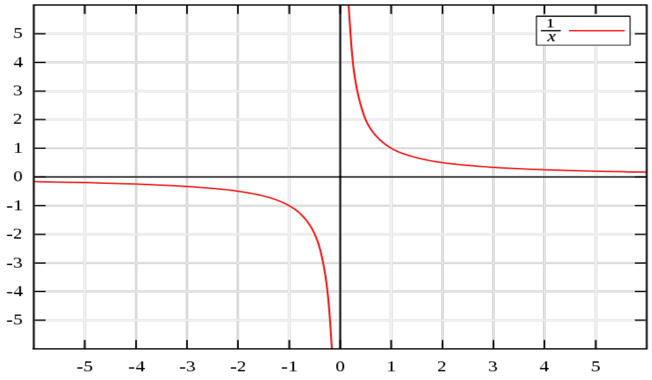
- **Assintótico**

adj (*assíntota+ico*[2]) *Geom* **1** Pertencente ou relativo à assíntota. **2** Qualificativo do espaço compreendido entre uma curva e a sua assíntota. **3** Diz-se da direção paralela de uma assíntota. *Var: assimptótico.*

- **Assíntota**

sf (*gr asýmptotos*) *Geom* Linha reta que se aproxima indefinidamente de uma curva sem nunca poder tocá-la. *Var: assímptota.*

A função *f(x)*=1/*x* tem como assíntotas os eixos coordenados.



***(Fonte:** http://pt.wikipedia.org/wiki/Assímptota)*

Crescimento Assintótico de Funções

- Escolha do algoritmo não é um problema crítico quando ***n*** é pequeno.
  - O problema é quando ***n*** cresce.
- Por isso, é usual analisar o comportamento das funções de custo quando ***n*** é bastante grande:
  - analisa-se o comportamento assintótico das funções de custo;
  - representa o *limite do comportamento da função de custo quando* ***n*** *cresce.*

Crescimento Assintótico de Funções

- Eficiência assintótica dos algoritmos:
  - estuda a maneira como o tempo de execução de um algoritmo aumenta com o tamanho da entrada no limite, à medida que o tamanho da entrada aumenta indefinidamente (sem limitação)
  - em geral, um algoritmo que é assintoticamente mais eficiente será a melhor escolha para toda as entradas, exceto as pequenas.

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

Comportamento Assintótico

- Supondo uma máquina que execute 1 milhão (10^6) de operações por segundo

$$\overbrace{\hspace{10cm}}^{n}$$

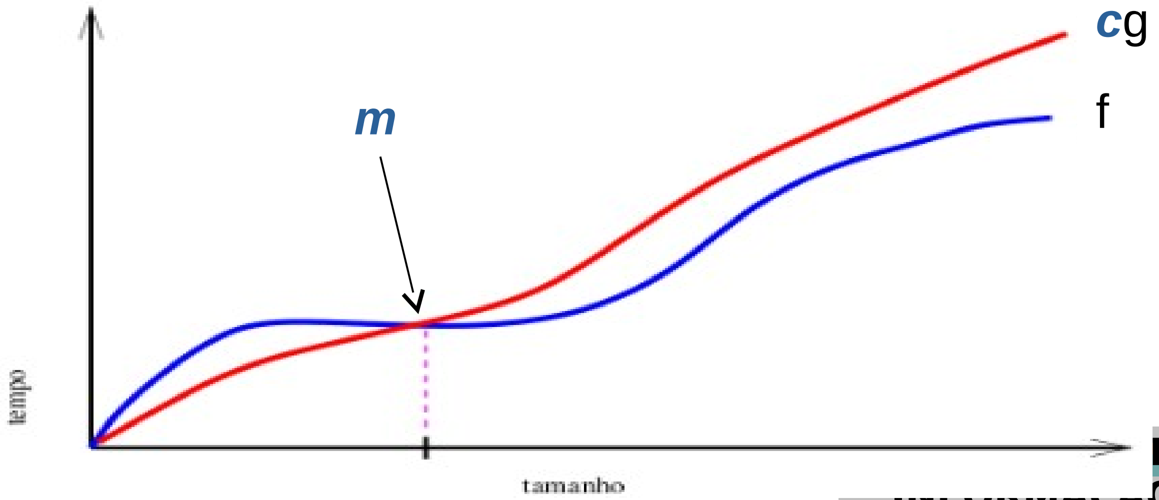
Função de custo	10	20	30	40	50	60
n	0,00001s	0,00002s	0,00003s	0,00004s	0,00005s	0,00006s
n^2	0,0001s	0,0004s	0,0009s	0,0016s	0,0025s	0,0036s
n^3	0,001s	0,008s	0,027s	0,064s	0,125s	0,216s
n^5	0,1s	3,2s	24,3s	1,7min	5,2min	12,96min
2^n	0,001s	1,04s	17,9min	12,7dias	35,7 anos	366 séc.
3^n	0,059s	58min	6,5anos	3855séc.	10^8séc.	10^{13}séc.

Comportamento Assintótico – Resumindo...

- Se ***f(n)*** é a função de complexidade de um algoritmo A
  - O comportamento assintótico de f (n) representa o limite do comportamento do custo (complexidade) de A quando ***n*** cresce.
- A análise de um algoritmo (função de complexidade)
  - Geralmente considera apenas algumas operações elementares ou mesmo uma operação elementar (e.g., o número de comparações).
- A complexidade assintótica relata crescimento assintótico das operações elementares.

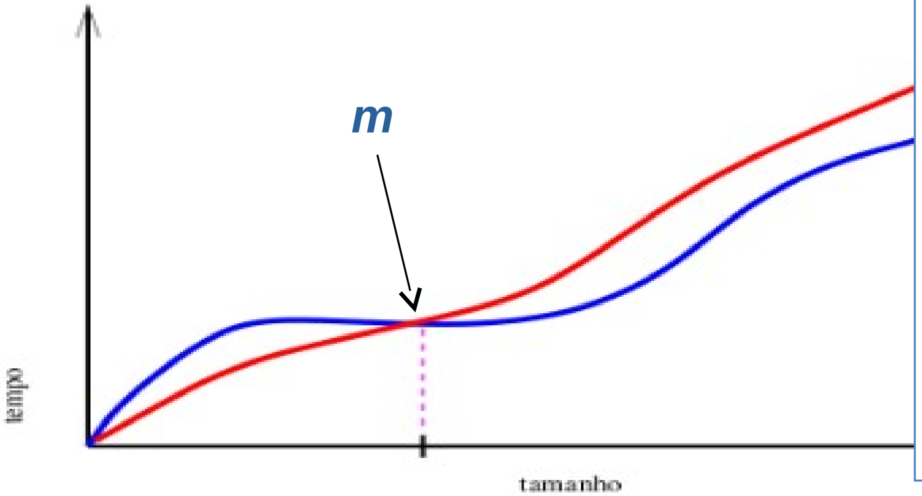
Relacionamento Assintótico

- Definição:
  - Uma função ***g(n)*** domina assintoticamente outra função ***f(n)*** se existem duas constantes positivas ***c*** e ***m*** tais que, para $n \geq m$, tem-se $|f(n)| \leq c \cdot |g(n)|$.



Relacionamento Assintótico

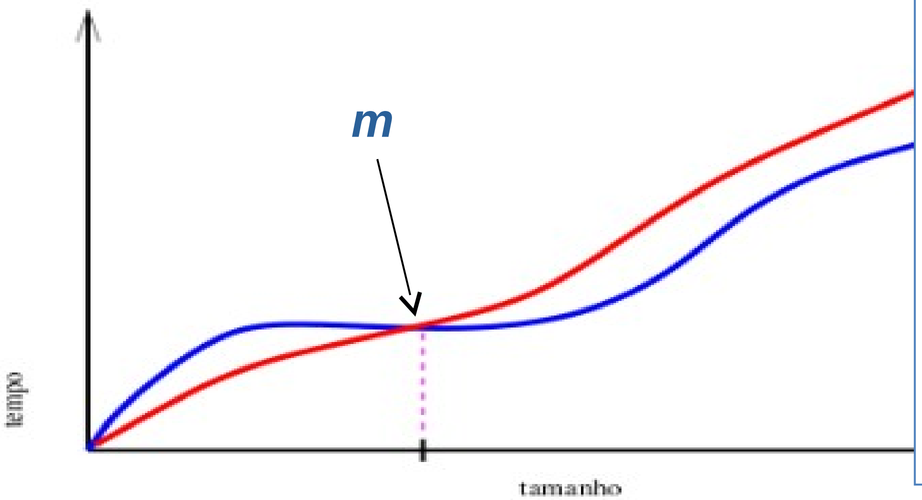
- Definição:
  - Uma função ***g(n)*** domina assintoticamente outra função ***f(n)*** se existem duas constantes positivas ***c*** e ***m*** tais que, para $n \geq m$, tem-se $|f(n)| \leq c \cdot |g(n)|$.



- Exemplo: $g(n) = n$ e $f(n) = n^2$
- Alguém domina alguém?

Relacionamento Assintótico

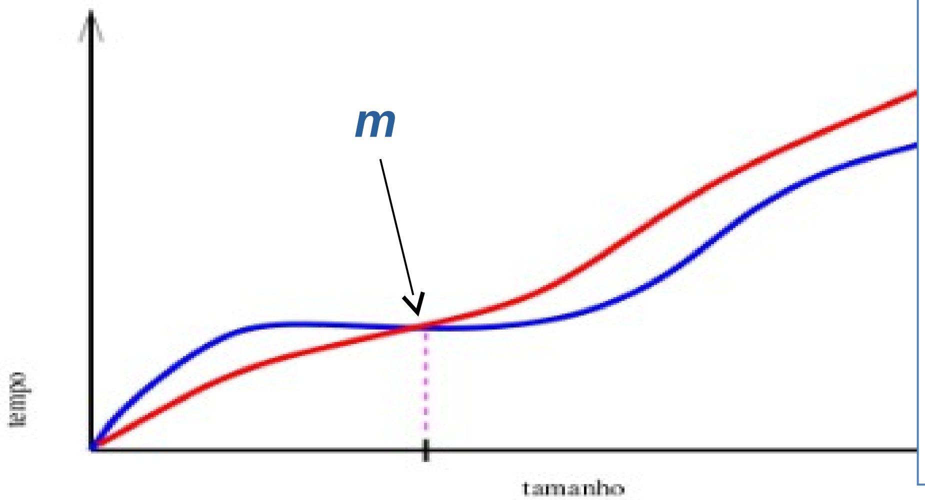
- Definição:
  - Uma função ***g(n)*** domina assintoticamente outra função ***f(n)*** se existem duas constantes positivas ***c*** e ***m*** tais que, para $n \geq m$, tem-se $|f(n)| \leq c \cdot |g(n)|$.



- Exemplo: $g(n) = n$ e $f(n) = n^2$
- Alguém domina alguém?
  - $|n| \leq |n^2|$ para todo $n \in N$
- Para c = ? e m = ? $\Rightarrow$ $|g(n)| \leq |f(n)|$
- Portanto, f (n) domina assintoticamente g(n).

Relacionamento Assintótico

- Definição:
  - Uma função ***g(n)*** domina assintoticamente outra função ***f(n)*** se existem duas constantes positivas ***c*** e ***m*** tais que, para $n \geq m$, tem-se $|f(n)| \leq c \cdot |g(n)|$.



- Exemplo: $g(n) = n$ e $f(n) = n^2$
- Alguém domina alguém?
  - $|n| \leq |n^2|$ para todo $n \in N$

Não necessariamente precisa mostrar os primeiros valores (c=13 e m=5 também vale para a prova)

- Para $c = 1$ e $m = 0 \Rightarrow |g(n)| \leq |f(n)|$
- Portanto, f (n) domina assintoticamente g(n).

Relacionamento Assintótico

- $g(n) = n$ e $f(n) = -n^2$
- Alguém domina alguém?
  - ???

Relacionamento Assintótico

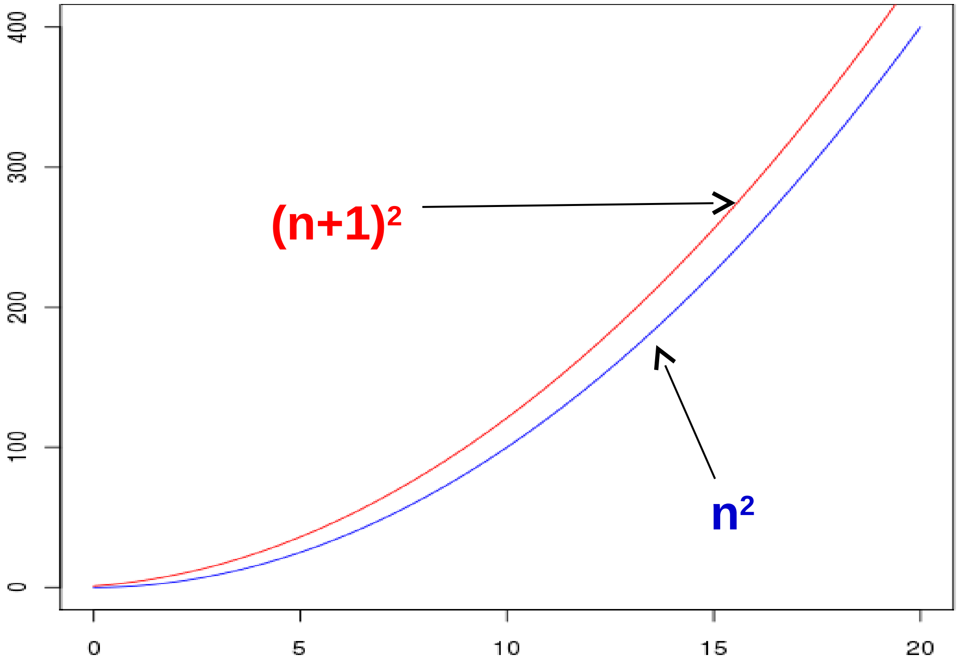
- $g(n) = n$ e $f(n) = -n^2$
- Alguém domina alguém?
  - $|n| \leq |-n^2|$ para todo $n \in N$.
    - Por ser módulo, o sinal não importa
  - Para $c = 1$ e $m = 0 \Rightarrow |g(n)| \leq |f(n)|$.
- Portanto, $f(n)$ domina assintoticamente $g(n)$.

Relacionamento Assintótico

- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - ???

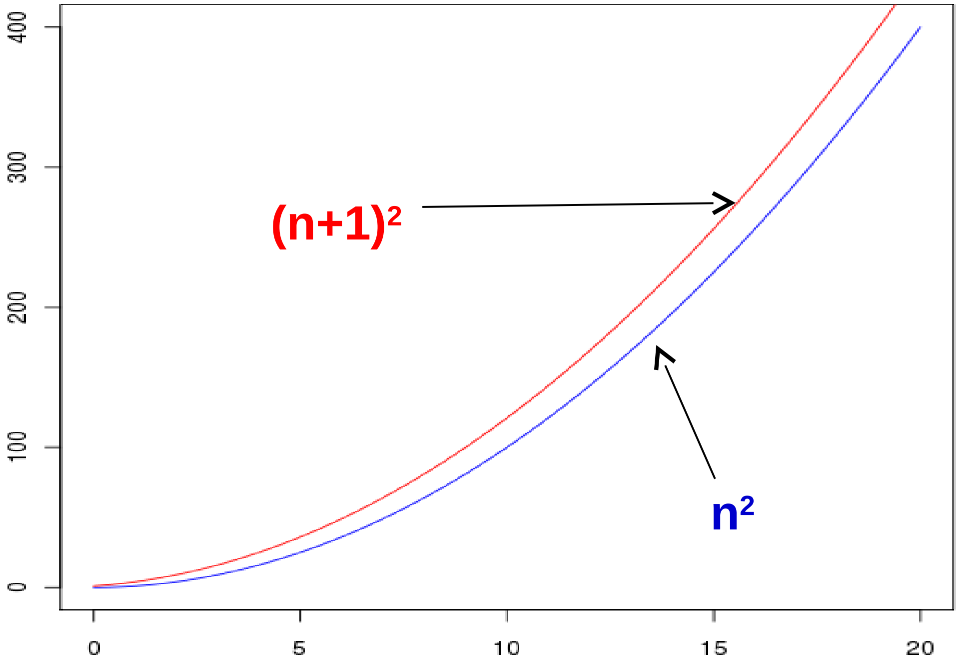
Relacionamento Assintótico

- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - Vamos colocar em um gráfico



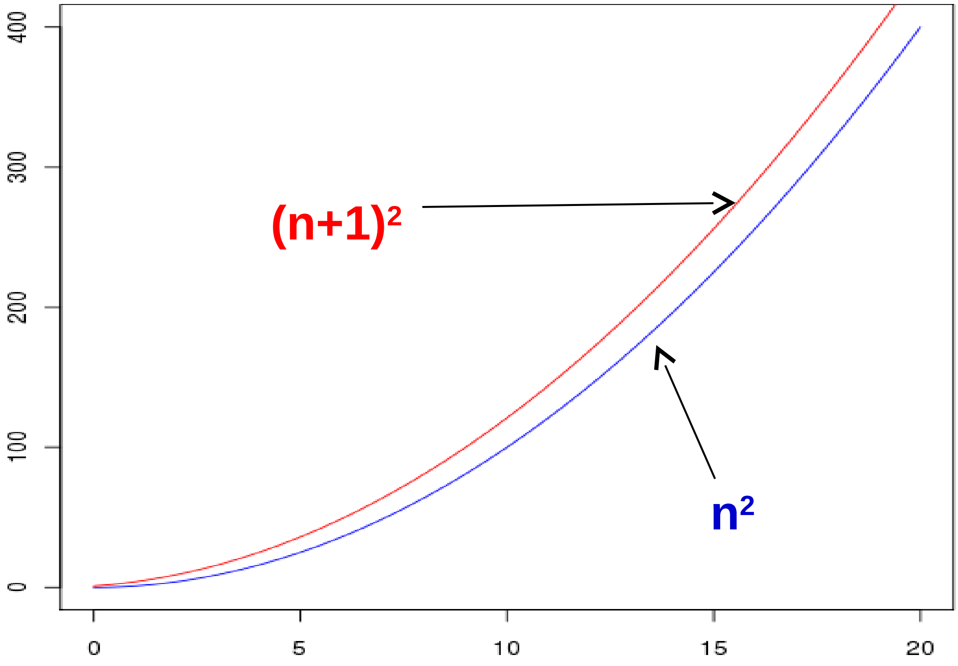
Relacionamento Assintótico

- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - Vamos colocar em um gráfico
  - $|n^2| \leq |(n+1)^2|$, para $n \geq 0$, $c = 1$
  - g(n) domina f(n)

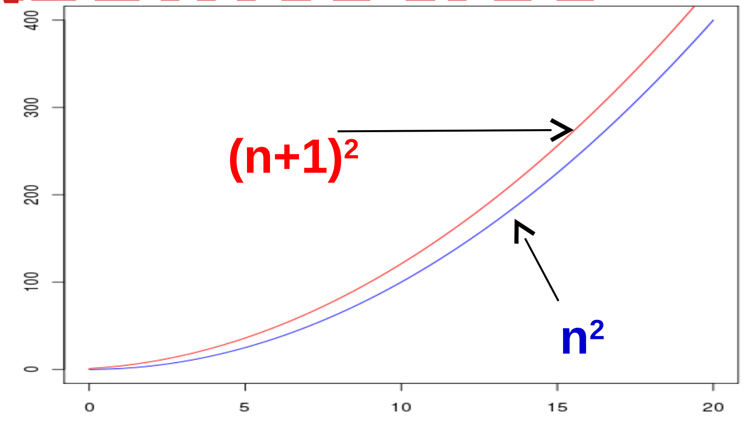


Relacionamento Assintótico

- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - Será somente isso?
  - Não há como f(n) dominar g(n)?
    - ???



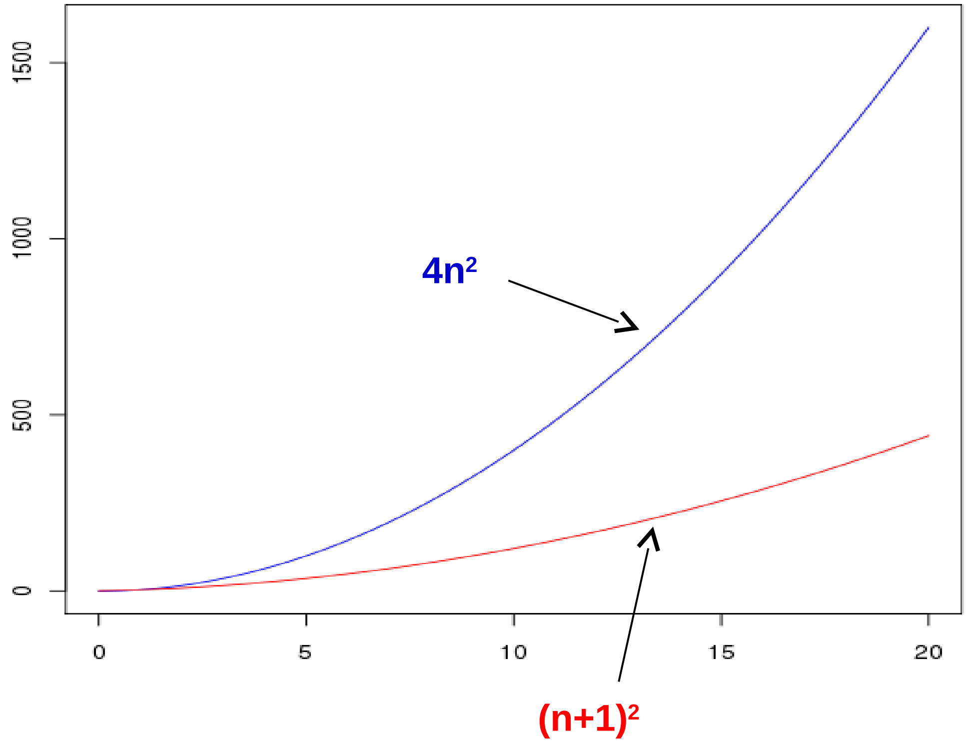
Relacionamento Assintótico



- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - Não há como f(n) dominar g(n)?
    - Lembre que a definição envolve também uma constante.
    - Suponha que queremos $g(n) \leq cf(n)$
    - Então $|(n+1)^2| \leq |cn^2|$
    - Mas, para isso, basta que $|(n+1)^2| \leq |(\sqrt{c}\, n)^2|$,
      - ou $|n+1| \leq |\sqrt{c}\, n|$
    - Se $\sqrt{c} = 2$, ou seja, $c=4$, isso é verdade

Relacionamento Assintótico

- $g(n) = (n+1)^2$ e $f(n) = n^2$
- Alguém domina alguém?
  - $|(n+1)^2| \leq |4n^2|$, para $n \geq 0$
  - f(n) domina g(n), para $n \geq 1$
- Nesse caso, dizemos que f(n) e g(n) dominam assintoticamente uma a outra.



Notação *O*

- Knuth(1971) * criou a notação ***O*** (lê-se "O grande") para expressar que ***g(n)*** domina assintoticamente ***f(n)***
  - Escreve-se ***f(n) = O(g(n))*** e lê-se: "***f(n) é da ordem no máximo g(n)***".
- Para que serve isto para o Bacharel em Sistemas de Informação?

*Knuth, D.E. (1971) "Mathematical Analysis of Algorithms". *Proceedings IFIP Congress 71, vol. 1, North Holland, Amsterdam, Holanda, 135-143.*

Notação *O*

- Para que serve isto para o Bacharel em Sistemas de Informação?
  - Muitas vezes calcular a função de complexidade exata ***f(n)*** de um algoritmo é complicado.
  - É mais fácil determinar que ***f(n)*** é ***O(g(n))***, isto é, que assintoticamente ***f(n)*** cresce no máximo como ***g(n)***.

Notação O

- Definição:

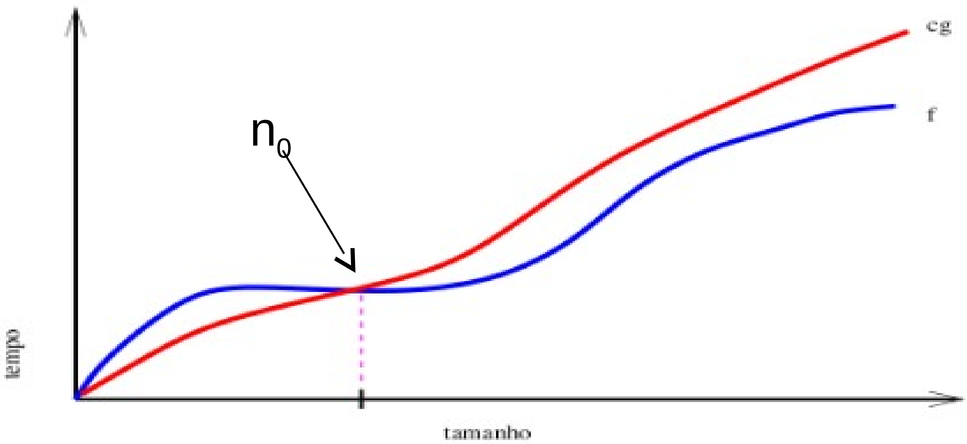
Conjunto de funções dominadas por g(n)

  - $O(g(n)) = \{f(n)$: existem constantes positivas $\boldsymbol{c}$ e $\boldsymbol{n_0}$ tais que $0 \leq f(n) \leq cg(n)$, para todo $\boldsymbol{n} \geq \boldsymbol{n_0}\}$

funções assintoticamente não negativas

- Informalmente, dizemos que, se $f(n) \in O(g(n))$, então f(n) cresce no máximo tão rapidamente quanto g(n).

cg é um limite superior de f



Notação *O*

- Definição:
  - $O(g(n)) = \{f(n)$: existem constantes positivas c e n_0 tais que $0 \leq f(n) \leq cg(n)$, para todo $n \geq n_0\}$
  - $\frac{3}{2}n^2 - 2n \in O(n^2)$?
    - ???

Notação *O*

- Definição:
  - $O(g(n))$ = {f(n): existem constantes positivas ***c*** e ***n₀*** tais que $0 \le f(n) \le cg(n)$, para todo $n \ge n_0$}
  - $\frac{3}{2}n^2 - 2n \in O(n^2)$?

$$\left|\frac{3}{2}n^2 - 2n\right| \le \left|c\,n^2\right|$$

- ???

Notação *O*

- Definição:
  - $O(g(n))$ = {f(n): existem constantes positivas ***c*** e ***n₀***
    tais que $0 \le f(n) \le cg(n)$, para todo $n \ge n_0$}
  - $\frac{3}{2}n^2 - 2n \in O(n^2)$?

$$\left|\frac{3}{2}n^2 - 2n\right| \le \left|c\,n^2\right|$$

- Fazendo c = 3/2, teremos $\left|\frac{3}{2}n^2 - 2n\right| \le \left|\frac{3}{2}n^2\right|$, para n ≥

Notação *O*

- Definição:
  - $O(g(n))$ = {f(n): existem constantes positivas ***c*** e $\boldsymbol{n_0}$ tais que $0 \leq f(n) \leq cg(n)$, para todo $\boldsymbol{n \geq n_0}$}
  - $\frac{3}{2}n^2 - 2n \in O(n^2)$?

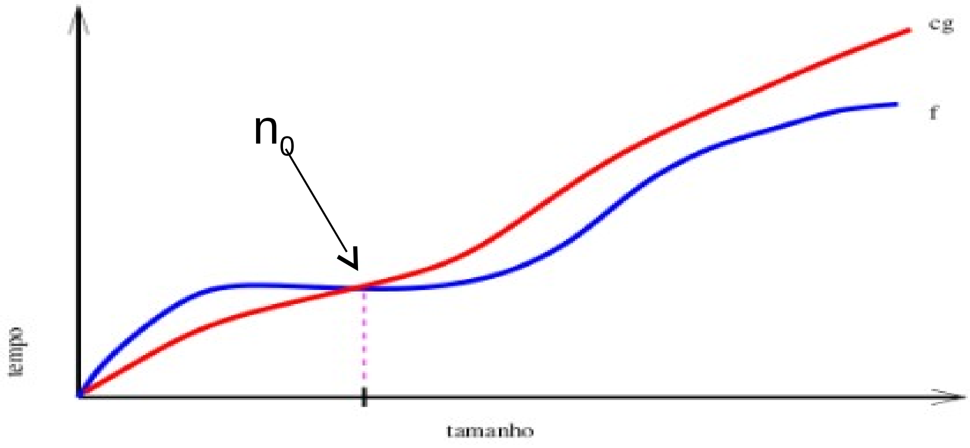
$$\left|\frac{3}{2}n^2 - 2n\right| \leq \left|c\,n^2\right|$$

Qualquer $n_0 >= 0$ vale

- Fazendo c = 3/2, teremos $\left|\frac{3}{2}n^2 - 2n\right| \leq \left|\frac{3}{2}n^2\right|$, para n ≥ 2
- Outras constantes podem existir, mas o que importa é que <u>existe alguma escolha para as constantes</u>

Notação *O*

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



Notação *O*

- Com a notação ***O*** podemos descrever frequentemente o tempo de execução de um algoritmo apenas inspecionando a estrutura global do algoritmo.

- Exemplo: estrutura de laço duplamente aninhado no algoritmo *insertion-sort* (visto anteriormente) produz um limite superior $O(n^2)$ no pior caso:
  - custo de **uma iteração** do laço interno é limitado superiormente por $O(1)$ (constante)
  - índices i e j são no máximo n
  - laço interno é executado no máximo uma vez para cada um dos n^2 pares de valores correspondentes a i e j

```
para j = 2 até tamanho[A] faça
  chave = A[j]
// ordenando elementos à esquerda
  i = j - 1
  enquanto i > 0 e A[i] > chave faça
     A[i+1] = A[i]
     i = i -1
   fim enquanto
   A[i+1] = chave
fim para
```

Operações com a notação *O*

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times f(n) &= O(f(n)), \text{ c é uma constante} \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Operações com a notação *O*

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times f(n) &= O(f(n)), \text{ c é uma constante} \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Particularmente útil para analisar algoritmos (sequências de trechos de código)

Operações com a notação *O*

- A regra $O(f(n)) + O(g(n)) = O(\max(f(n),g(n)))$ pode ser usada para calcular o tempo de execução de uma sequência de trechos de um programa
  - Suponha 3 trechos: $O(n)$, $O(n^2)$ e $O(n\log n)$
  - Qual o tempo de execução do algoritmo como um todo?
    - ???

Operações com a notação O

- A regra $O(f(n)) + O(g(n)) = O(\max(f(n),g(n)))$ pode ser usada para calcular o tempo de execução de uma sequência de trechos de um programa
  - Suponha 3 trechos: $O(n)$, $O(n^2)$ e $O(n\log n)$
  - Qual o tempo de execução do algoritmo como um todo?
    - Lembre-se que o tempo de execução é a soma dos tempos de cada trecho
    - $O(n) + O(n^2) + O(n\log n) = \max(O(n), O(n^2), O(n\log n)) = O(n^2)$

Ex: InsertionSort é O de quanto?

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n-(c_2+c_4+c_5+c_8)$$

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Ex: InsertionSort é O(n²)

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n-(c_2+c_4+c_5+c_8)$$

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Ex: InsertionSort é $O(n^2)$

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

$$T(n)=c_1 n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n \;-(c_2+c_4+c_5+c_8)$$

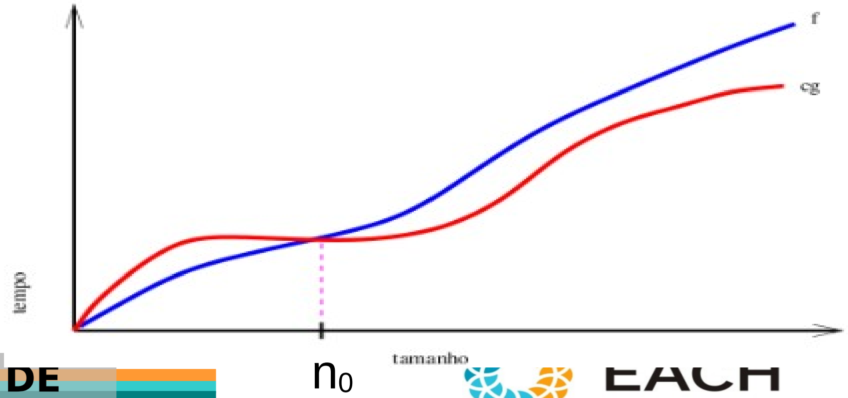
Também é $O(n^k)$ para k > 2...

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Notação Ω

- Definição:
  - $\Omega(g(n)) = \{f(n)$: existem constantes positivas $\boldsymbol{c}$ e $\boldsymbol{n_0}$ tais que $0 \leq cg(n) \leq f(n)$, para todo $\boldsymbol{n} \geq \boldsymbol{n_0}\}$

    funções assintoticamente não negativas
- Informalmente, dizemos que, se $f(n) \in \Omega(g(n))$, então f(n) cresce no mínimo tão lentamente quanto g(n).
  - Note que se $f(n) \in O(g(n))$ define um limite superior para f(n), $\Omega(g(n))$ define um limite inferior



Notação Ω

- Definição:
  - $\Omega(g(n)) = \{f(n)$: existem constantes positivas $\boldsymbol{c}$ e $\boldsymbol{n_0}$ tais que $0 \leq cg(n) \leq f(n)$, para todo $\boldsymbol{n} \geq \boldsymbol{n_0}\}$
  - $$\frac{3}{2}n^2 - 2n \in \Omega(n^2) \quad ?$$
    - ???

Notação Ω

- Definição:
  - Ω(g(n)) = {f(n): existem constantes positivas ***c*** e ***n_0*** tais que 0 ≤ cg(n) ≤ f(n), para todo ***n*** ≥ ***n_0***}
  - $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$?
    - Fazendo c = ? teremos $\left| c\,n^2 \right| \le \left| \frac{3}{2}n^2 - 2n \right|$, para n ≥ ?

0 ≤ cg(n) ≤ f(n)

Notação Ω

- Definição:
  - Ω(g(n)) = {f(n): existem constantes positivas ***c*** e ***n₀***
    tais que 0 ≤ cg(n) ≤ f(n), para todo ***n*** ≥ ***n₀***}
  - $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$?
    - Fazendo c = ? teremos $\left| c\,n^2 \right| \leq \left| \frac{3}{2}n^2 - 2n \right|$, para n ≥ ?

0 ≤ cg(n) ≤ f(n) $\left(\frac{3}{2} - c\right)n^2 - 2n >= 0$

Notação Ω

- Definição:
  - Ω(g(n)) = {f(n): existem constantes positivas ***c*** e $\boldsymbol{n_0}$ tais que $0 \le cg(n) \le f(n)$, para todo $\boldsymbol{n \ge n_0}$}
  - $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$?
    - Fazendo c = 1/2, teremos $\left|\frac{1}{2}n^2\right| \le \left|\frac{3}{2}n^2 - 2n\right|$, para n ≥ ?

$0 \le cg(n) \le f(n)$

$\left(\frac{3}{2} - c\right)n^2 - 2n >= 0$

$n^2 - 2n >= 0$

Notação Ω

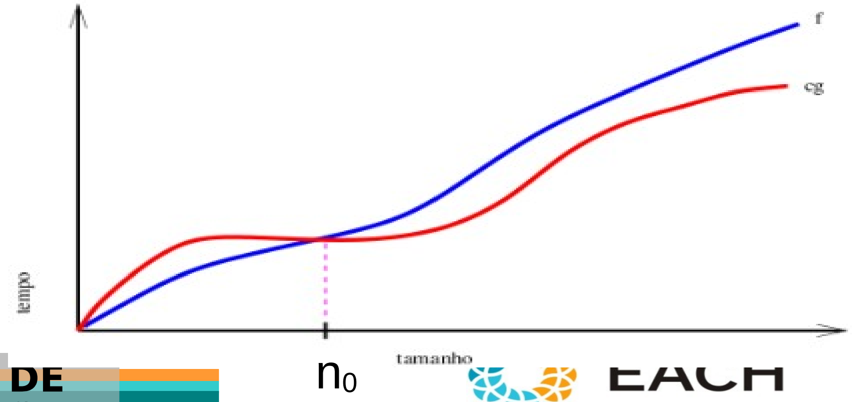
- Definição:
  - Ω(g(n)) = {f(n): existem constantes positivas ***c*** e ***n_0*** tais que 0 ≤ cg(n) ≤ f(n), para todo ***n*** ≥ ***n_0***}
  - $\frac{3}{2}n^2 - 2n \in \Omega(n^2)$?
    - Fazendo c = 1/2, teremos $\left|\frac{1}{2}n^2\right| \leq \left|\frac{3}{2}n^2 - 2n\right|$, para n ≥ 2

0 ≤ cg(n) ≤ f(n) $\left(\frac{3}{2} - c\right)n^2 - 2n >= 0$

$n^2 - 2n >= 0$

Notação Ω

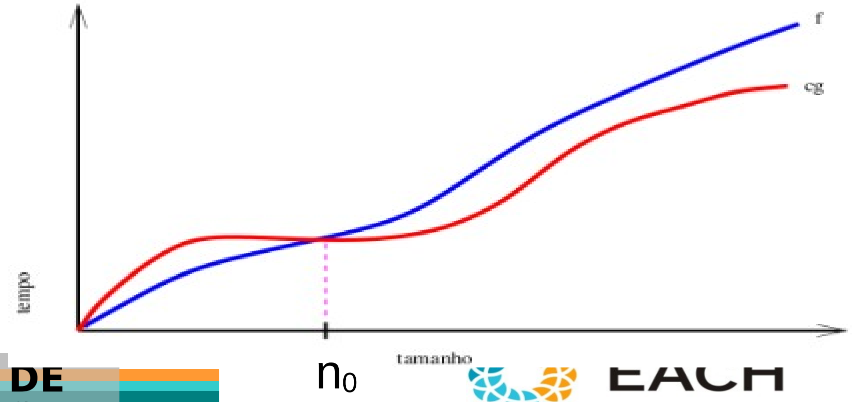
Ex: Qualquer algoritmo de ordenação é Ω(n). Por quê?



Notação Ω

Ex: Qualquer algoritmo de ordenação é $\Omega(n)$.
Porque no mínimo tem que conferir cada posição do array...

Mas é sempre interessante dar uma avaliação mais precisa



Ex: InsertionSort é Ω de quanto?

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ t_j=j para j=2,3,...,n)

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

$$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n-(c_2+c_4+c_5+c_8)$$

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Ex: InsertionSort é $\Omega(n^2)$

(por conta do pior caso)

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

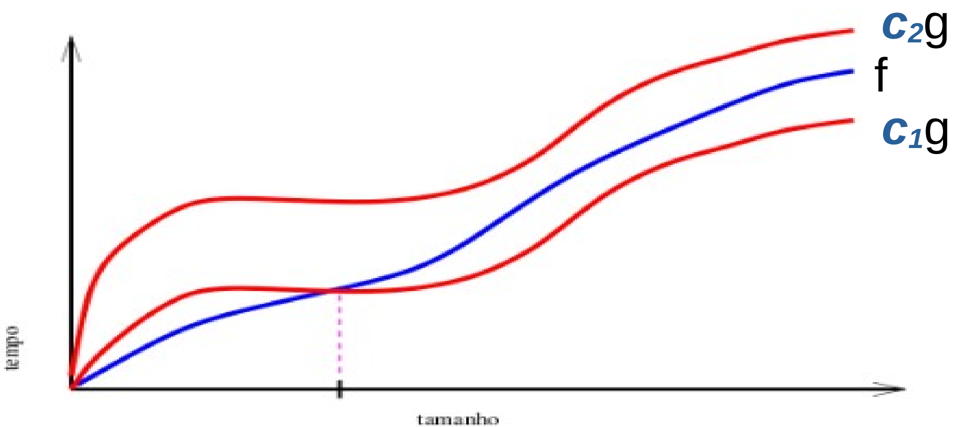
$$T(n)=c_1n+c_2(n-1)+c_4(n-1)+c_5\frac{n(n-1)}{2}-1+c_6\frac{n(n-1)}{2}+c_7\frac{n(n-1)}{2}+c_8(n-1)=$$

$$\left(\frac{c_5}{2}+\frac{c_6}{2}+\frac{c_7}{2}\right)n^2+\left(c_1+c_2+c_4+\frac{c_5}{2}-\frac{c_6}{2}-\frac{c_7}{2}+c_8\right)n\ -(c_2+c_4+c_5+c_8)$$

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Notação Θ

- Definição:
  - $\Theta(g(n)) = \{f(n)$: existem constantes positivas $\boldsymbol{c_1}$, $\boldsymbol{c_2}$ e $\boldsymbol{n_0}$ tais que $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, para todo $\boldsymbol{n} \geq \boldsymbol{n_0}\}$
- Informalmente, dizemos que, se $f(n) \in \Theta(g(n))$, então f(n) cresce tão rapidamente quanto g(n).



Notação Θ

- Definição:
  - Θ(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 $n \geq n_0$}
  - $\frac{3}{2}n^2 - 2n \in \Theta(n^2)$?
    - ???

Notação 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 $n \geq n_0$}
  - $\frac{3}{2}n^2 - 2n \in \Theta(n^2)$?
    - Fazendo $c_1 = 1/2$ e $c_2 = 3/2$ teremos $\left|\frac{1}{2}n^2\right| \leq \left|\frac{3}{2}n^2 - 2n\right| \leq \left|\frac{3}{2}n^2\right|$

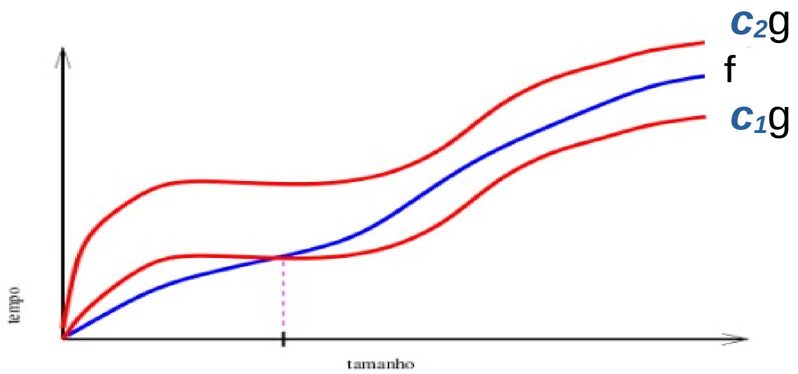
para $n \geq 2$

Notação Θ

- Mas, já vimos que:
  - $$\frac{3}{2}n^2 - 2n \in O(n^2) \rightarrow \left|\frac{3}{2}n^2 - 2n\right| \le \left|\frac{3}{2}n^2\right|$$
  - $$\frac{3}{2}n^2 - 2n \in \Omega(n^2) \rightarrow \left|\frac{1}{2}n^2\right| \le \left|\frac{3}{2}n^2 - 2n\right|$$ e ...
  - $$\frac{3}{2}n^2 - 2n \in \Theta(n^2) \rightarrow \left|\frac{1}{2}n^2\right| \le \left|\frac{3}{2}n^2 - 2n\right| \le \left|\frac{3}{2}n^2\right|$$
  - Será coincidência?
    - ???

Notação Θ

- Mas, já vimos que:
  - $$\frac{3}{2}n^2 - 2n \in O(n^2) \rightarrow \left|\frac{3}{2}n^2 - 2n\right| \le \left|\frac{3}{2}n^2\right|$$
  - $$\frac{3}{2}n^2 - 2n \in \Omega(n^2) \rightarrow \left|\frac{1}{2}n^2\right| \le \left|\frac{3}{2}n^2 - 2n\right|$$
  - $$\frac{3}{2}n^2 - 2n \in \Theta(n^2) \rightarrow \left|\frac{1}{2}n^2\right| \le \left|\frac{3}{2}n^2 - 2n\right| \le \left|\frac{3}{2}n^2\right|$$



- Será coincidência?
  - Não!
  - Se f(n) ∈ O(g(n)) e f(n) ∈ Ω (g(n)), então f(n) ∈ Θg(n))

EACH

Então InsertionSort é Θ (n²)

(por conta do pior caso)

- Tempo de execução do algoritmo = soma dos tempos de execução para cada instrução
- $T(n)=c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j - 1) + c_7\sum_{j=2}^{n}(t_j - 1) + c_8(n-1)$

- Pior caso: vetor em ordem inversa (deve comparar cada elemento A[j] com cada elemento do subarranjo ordenado A[j... j-1] ➔ $t_j=j$ para j=2,3,...,n)

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

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\frac{n(n-1)}{2} - 1 + c_6\frac{n(n-1)}{2} + c_7\frac{n(n-1)}{2} + c_8(n-1) =$$

$$\left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

- Tempo de execução, neste caso, pode ser expresso como $an^2 + bn + c$ para constantes **a**, **b** e **c** que dependem dos custos de instrução c_i ➔ **função quadrática** de n

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$f_1(n) = 2^\pi$$
$$f_2(n) = 2^n$$
$$f_3(n) = n \log n$$
$$f_4(n) = \log n$$
$$f_5(n) = 100n^2 + 150000n$$
$$f_6(n) = n + \log n$$
$$f_7(n) = n^2$$
$$f_8(n) = n$$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$\boxed{f_1(n) = 2^\pi}$$
$$f_2(n) = 2^n$$
$$f_3(n) = n \log n$$
$$f_4(n) = \log n$$
$$f_5(n) = 100n^2 + 150000n$$
$$f_6(n) = n + \log n$$
$$f_7(n) = n^2$$
$$f_8(n) = n$$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$\boxed{f_1(n) = 2^{\pi}}$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$f_3(n) = n \log n$

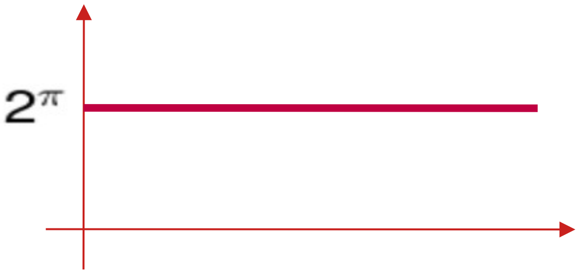
$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$f_7(n) = n^2$

$f_8(n) = n$



	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$\boxed{f_1(n) = 2^\pi} \quad = \Theta(1) \rightarrow \text{constante!!!}$$

$$f_2(n) = 2^n$$

$$f_3(n) = n \log n$$

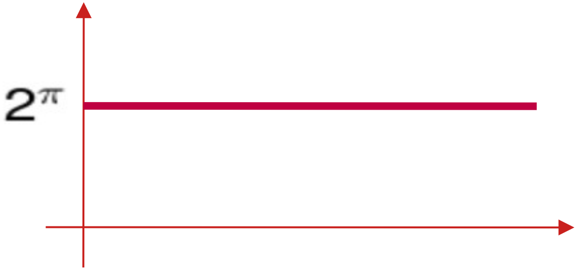
$$f_4(n) = \log n$$

$$f_5(n) = 100n^2 + 150000n$$

$$f_6(n) = n + \log n$$

$$f_7(n) = n^2$$

$$f_8(n) = n$$



	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	$\boxed{\Theta}$							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

$\boxed{\text{Prove que } 2^\pi = \Theta(2^\pi)}$

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$\boxed{f_1(n) = 2^{\pi}}$ $= \Theta(1) \rightarrow$ constante!!!

$f_2(n) = 2^n$

$f_3(n) = n \log n$

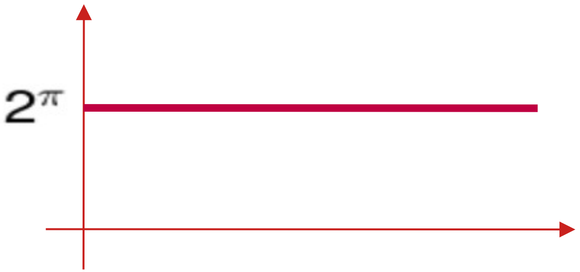
$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$f_7(n) = n^2$

$f_8(n) = n$



	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

Prove que $2^{\pi} = \Theta(2^{\pi})$

Existem c1, c2 e n0 constantes positivas tal que

$0 <= c_1 2^{\pi} <= 2^{\pi} <= c_2\, 2^{\pi}$ para $n >= n_0$

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$\boxed{f_1(n) = 2^{\pi}}$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$f_3(n) = n \log n$

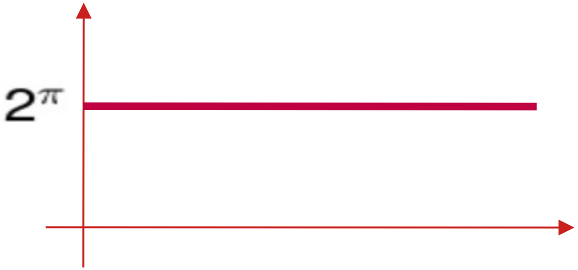
$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$f_7(n) = n^2$

$f_8(n) = n$



	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	$\boxed{\Theta}$							
f_2		Θ						
f_3			Θ					
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

$\boxed{\text{Prove que } 2^{\pi} = \Theta(2^{\pi})}$

Existem c1, c2 e n0 constantes positivas tal que

$0 <= c_1 2^{\pi} <= 2^{\pi} <= c_2 2^{\pi}$ para $n >= n_0$

Ex: $c_1 = c_2 = n_0 = 1$

$c_1 = 0.5$, $c_2 = 2$, $n_0 = 0.1$ (não podem ser 0…)

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^{\pi}$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				$\boxed{?}$	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

n logn = ? (n^2) – Prove

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^\pi$ = $\Theta(1)$ → constante!!!
$f_2(n) = 2^n$
$\boxed{f_3(n) = n \log n}$
$f_4(n) = \log n$
$f_5(n) = 100n^2 + 150000n$
$f_6(n) = n + \log n$
$\boxed{f_7(n) = n^2}$
$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

n logn = ? (n^2) – Prove

n logn = $O(n^2)$, pois existem constantes positivas c e n_0 tais que:

$0 \le n \log n \le cn^2$, para todo $n \ge n_0$

Isso vale para c = ?, n_0 = ?

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^{\pi}$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				$\boxed{O}$	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

n logn = ? (n^2) – Prove

n logn = O(n^2), pois existem constantes positivas c e n_0 tais que:

0 <= n logn <= cn^2, para todo n >= n_0

Isso vale para c = 1, n_0 = 2

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^\pi$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			?				Θ	
f_8								Θ

n^2 = ? (n logn) – Prove

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^\pi$ = $\Theta(1)$ → constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			?				Θ	
f_8								Θ

n^2 = ? (n logn) – Prove

$n^2 = \Omega$(n logn), pois

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^{\pi}$ = $\Theta(1) \rightarrow$ constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7							Θ	
f_8								Θ

n^2 = ? (n logn) – Prove

$n^2 = \Omega$(n logn), pois existem constantes positivas c e n_0 tais que:

0 <= c n logn <= n^2, para todo n >= n_0

Isso vale para c = ?, n_0 = ?

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^\pi$ $= \Theta(1) \rightarrow$ constante!!!

$f_2(n) = 2^n$

$\boxed{f_3(n) = n \log n}$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$\boxed{f_7(n) = n^2}$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			$\boxed{\Omega}$				Θ	
f_8								Θ

n^2 = ? (n logn) – Prove

$n^2 = \Omega(n \log n)$, pois existem constantes positivas c e n_0 tais que:

$0 <= c\, n \log n <= n^2$, para todo $n >= n_0$

Isso vale para $c = 1$, $n_0 = 2$

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$f_1(n) = 2^{\pi}$$ = $\Theta(1) \rightarrow$ constante!!!

$$f_2(n) = 2^n$$
$$f_3(n) = n \log n$$
$$f_4(n) = \log n$$
$$f_5(n) = 100n^2 + 150000n$$
$$f_6(n) = n + \log n$$
$$f_7(n) = n^2$$
$$f_8(n) = n$$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			Ω				Θ	
f_8								Θ

Obs: Prove que n logn NÃO é $\Omega(n^2)$

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$f_1(n) = 2^{\pi} \quad = \Theta(1) \rightarrow \text{constante!!!}$$
$$f_2(n) = 2^n$$
$$f_3(n) = n \log n$$
$$f_4(n) = \log n$$
$$f_5(n) = 100n^2 + 150000n$$
$$f_6(n) = n + \log n$$
$$f_7(n) = n^2$$
$$f_8(n) = n$$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			Ω				Θ	
f_8								Θ

Obs: Prove que n logn NÃO é $\Omega(n^2)$

Se fosse, então existiriam constantes positivas c e n_0 tais que:

$0 <= cn^2 <= n \log n$, para todo $n >= n_0$

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^{\pi}$ = $\Theta(1) \rightarrow$ constante!!!

$f_2(n) = 2^n$

$f_3(n) = n \log n$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$f_7(n) = n^2$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			Ω				Θ	
f_8								Θ

Obs: Prove que n logn NÃO é $\Omega(n^2)$

Se fosse, então existiriam constantes positivas c e n_0 tais que:

$0 <= cn^2 <= n \log n$, para todo $n >= n_0$

Se fosse verdade, quem poderia ser esse c e n_0?

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$f_1(n) = 2^{\pi}$ $= \Theta(1) \rightarrow$ constante!!!

$f_2(n) = 2^n$

$f_3(n) = n \log n$

$f_4(n) = \log n$

$f_5(n) = 100n^2 + 150000n$

$f_6(n) = n + \log n$

$f_7(n) = n^2$

$f_8(n) = n$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			Ω				Θ	
f_8								Θ

Obs: Prove que n logn NÃO é $\Omega(n^2)$

Se fosse, então existiriam constantes positivas c e n_0 tais que:

$0 <= cn^2 <= n \log n$, para todo $n >= n_0$

Se fosse verdade, $0 <= cn <= \log n \Rightarrow c <= \log n / n$ para todo $n >= n_0$

Mas $\lim_{n \rightarrow \infty} \log n / n = 0$, MAS c deve ser positiva! CONTRADIÇÃO

Exercício

Quais as relações de comparação assintótica (O, Ω, Θ) das funções:

$$f_1(n) = 2^\pi \quad = \Theta(1) \rightarrow \text{constante!!!}$$
$$f_2(n) = 2^n$$
$$f_3(n) = n \log n$$
$$f_4(n) = \log n$$
$$f_5(n) = 100n^2 + 150000n$$
$$f_6(n) = n + \log n$$
$$f_7(n) = n^2$$
$$f_8(n) = n$$

	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
f_1	Θ							
f_2		Θ						
f_3			Θ				O	
f_4				Θ				
f_5					Θ			
f_6						Θ		
f_7			Ω				Θ	
f_8								Θ

FAÇAM AS PROVAS FORMAIS PARA OS DEMAIS!!!!

(Inclusive para o que NÃO É)

Material extra sobre O, Ω, Θ

https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/Oh.html

Além de explicações, há 3 listas de exercícios. FAÇAM! (Cai na prova...)

Referências

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest & Clifford Stein. Algoritmos - Tradução da 2a. Edição Americana. Editora Campus, 2002 (Capítulo 3).
- Michael T. Goodrich & Roberto Tamassia. Estruturas de Dados e Algoritmos em Java. Editora Bookman, 4a. Ed. 2007 (Capítulo 4).
- Nívio Ziviani. Projeto de Algoritmos com implementações em C e Pascal. Editora Thomson, 2a. Edição, 2004 (Seção 1.3).
- Notas de aula dos professores Marcos Chaim, Cid de Souza, Cândida da Silva e Delano M. Beder.
- Notas de aula dos professores Fátima L. S. Nunes e Norton T. Roman