

# COMPLEXIDADE DE ALGORITMOS



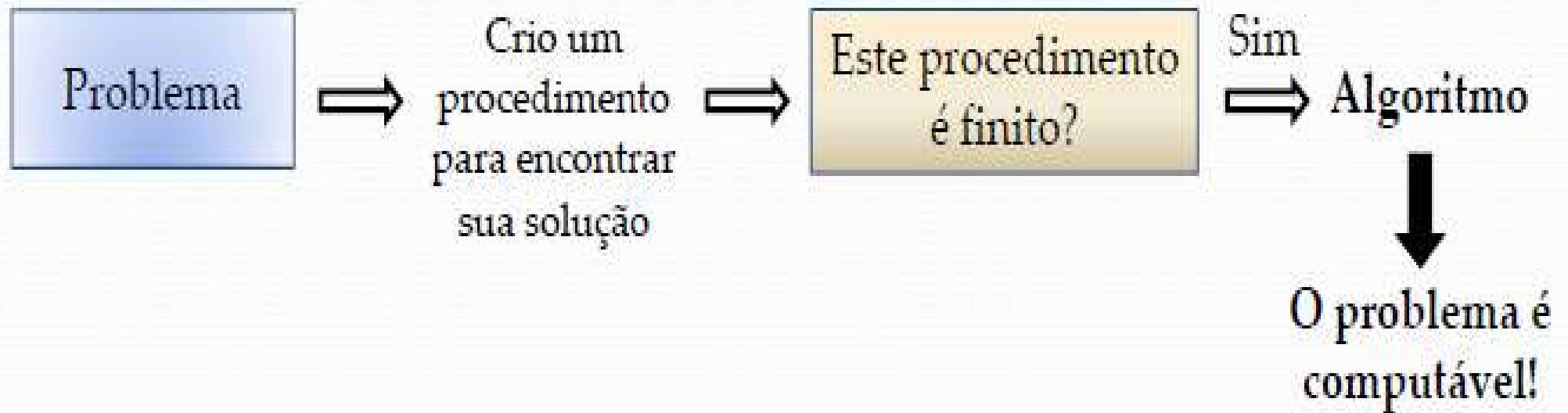
## Introdução

Prof. Dr. Luiz Camolesi Jr.

## Definição

A **Complexidade de um Algoritmo** consiste na quantidade de “trabalho” necessária para a sua execução, expressa em função das operações fundamentais, as quais variam de acordo com o algoritmo e em função do volume de dados

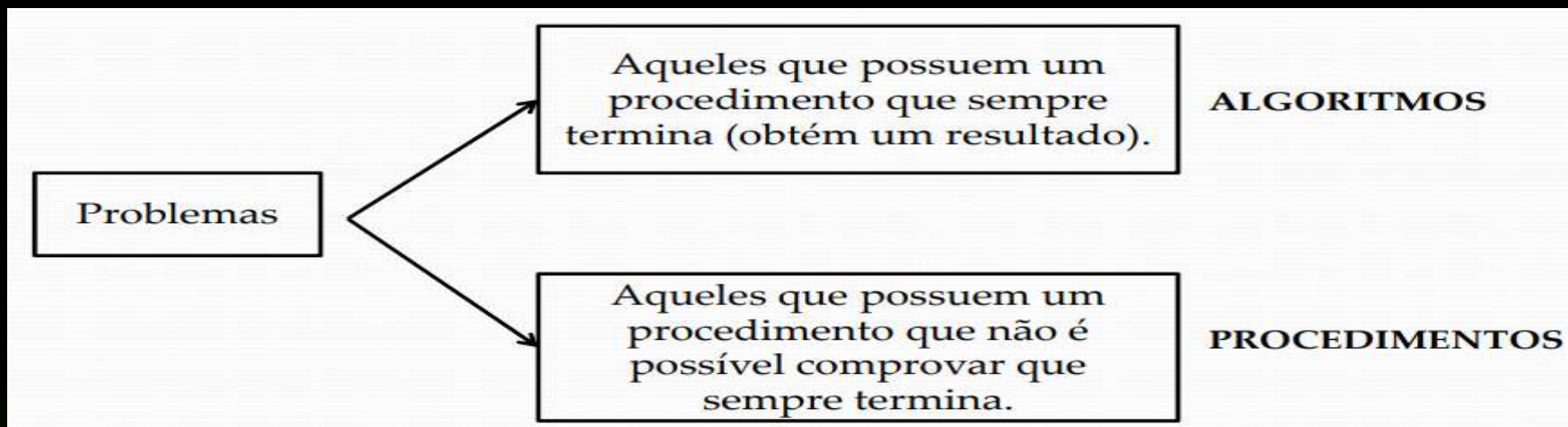
# INTRODUÇÃO



**Computabilidade:** refere-se à existência ou não de um procedimento que resolve um determinado problema em um número finito de passos.

# Tipos de Problemas

- Tipo 1 : problemas que podem ser resolvidos com um conjunto de operações e que sempre terminam e obtém um resultado
- Tipo 2 : problemas que podem ser resolvidos com um conjunto de operações mas que não é possível comprovar que irá obter um resultado (que possui termino)

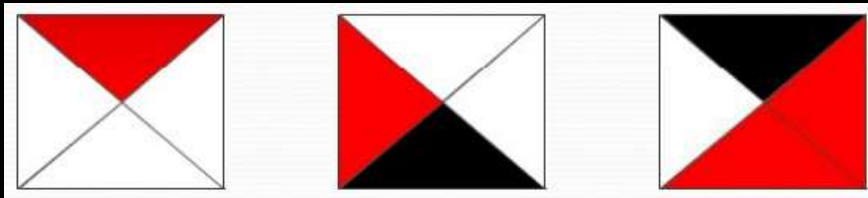


quais problemas podem ser resolvidos com o auxílio desta máquina?

Exemplo de problema não-computável:

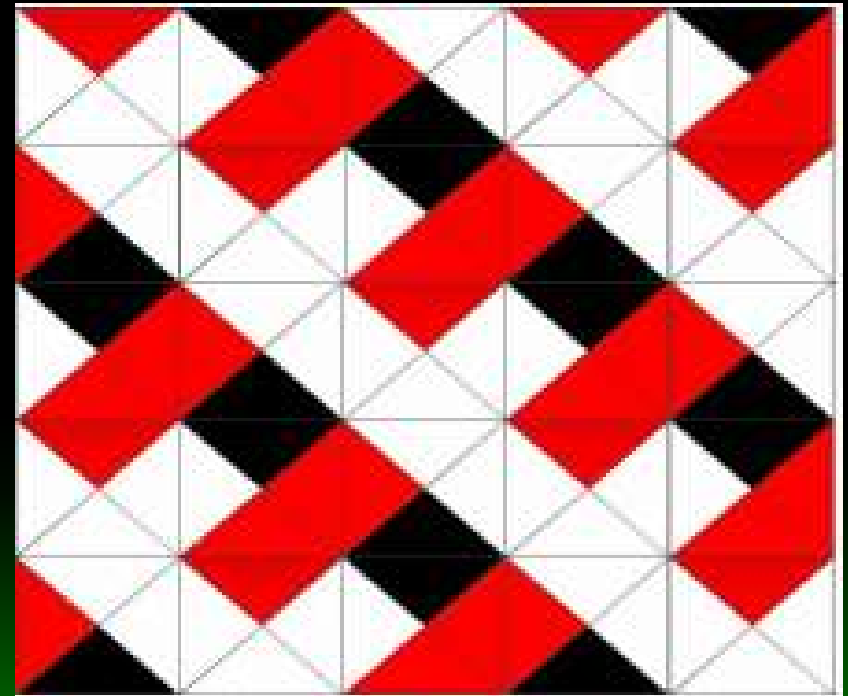
## Problema do azulejo (tiling problem):

Um azulejo é um quadrado com orientação fixa. Temos um conjunto de tamanho  $T$  contendo azulejos em 4 cores (C1 a C4), não necessariamente distintas, conforme figura. O desafio é cobrir uma área quadrada finita com azulejos respeitando a restrição de que os azulejos devem se encostar no lado de mesma cor.



Desejamos preencher uma área de  $5 \times 5$  azulejos.

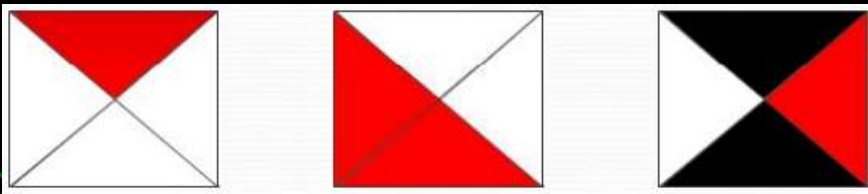
Solução →



Exemplo de problema não-computável:

## Problema do azulejo (tiling problem):

Um azulejo é um quadrado com orientação fixa. Temos um conjunto de tamanho  $T$  contendo azulejos em 4 cores (C1 a C4), não necessariamente distintas, conforme figura. O desafio é cobrir uma área quadrada finita com azulejos respeitando a restrição de que os azulejos devem se encostar no lado de mesma cor.

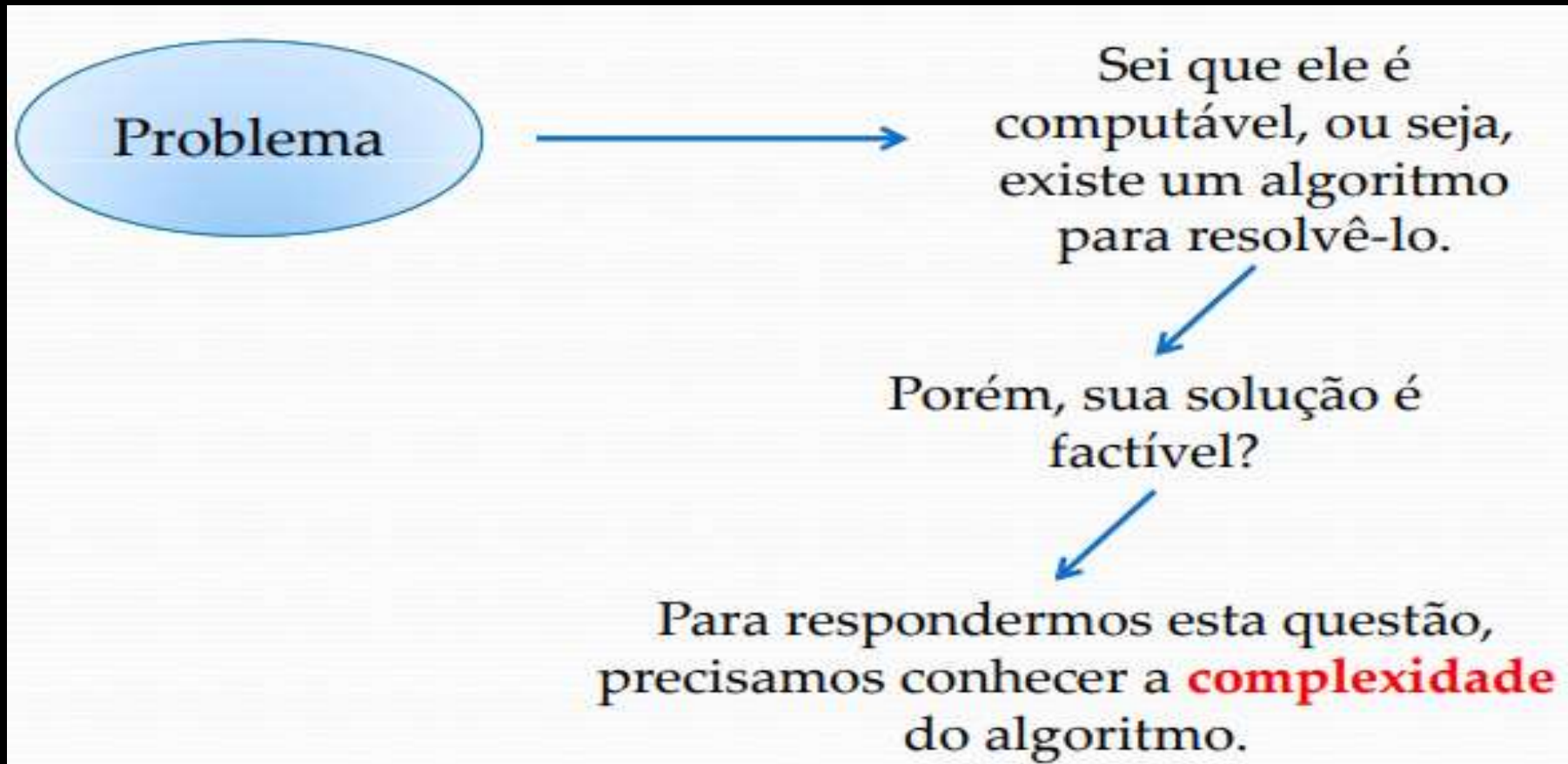


Desejamos preencher uma área de  $3 \times 3$  azulejos.

# Não tem Solução

Problema inicialmente estudado por  
Hao Wang em 1961





Vamos então à **Complexidade de Algoritmos** ....

Um algoritmo serve para resolver um determinado problema, e todos os problemas têm sempre uma entrada de dados (N)

- O tamanho desse N afeta sempre diretamente no tempo de resposta de um algoritmo
- Dependendo do problema, já existem alguns algoritmos prontos, ou que podem ser Adaptados
- O problema é: qual algoritmo escolher?



Exemplo:

Solução de um sistema lineares com n ( $x_1, x_2, x_3, \dots, x_n$ ) incógnitas

$$\begin{pmatrix} 10 & 5 & -3 & 2 \\ 0 & 6 & -4 & 2 \\ 0 & 0 & 5 & -3 \\ 0 & 0 & 0 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 19 \\ 0 \\ 5 \\ 10 \end{pmatrix}$$

Métodos:

Cramer



Gauss



Estimativa empírica de tempo, para um processador rodando em 3GHz

n	Cramer	Gauss
2	4 ns	2 ns
3	12 ns	8 ns
4	48 ns	20 ns
5	240ns	40 ns
10	7.3ms	330 ns
20	152 anos	2.7 ms

A complexidade de um algoritmo pode ser dividida em:

- **Complexidade Espacial:** Quantidade de recursos utilizados para resolver o problema;
- **Complexidade Temporal:** Quantidade de Tempo utilizado. Pode ser visto também como o número de instruções necessárias para resolver determinado problema;

Em ambos os casos, a complexidade é medida de acordo com o tamanho dos dados de entrada ( $N$ )

Existem três escalas de complexidade:

- Melhor Caso
- Caso Médio
- Pior Caso

Nas três escalas, a função de complexidade retorna a complexidade de um algoritmo com entrada de  $N$  elementos

A Complexidade independe de **linguagem** ou **Computador** usado para a solução.



Existem três escalas de complexidade:

- Melhor Caso
- Caso Médio
- Pior Caso

Nas três escalas, a função de complexidade retorna a complexidade de um algoritmo com entrada de  $N$  elementos

## Caso Melhor

Definido pela letra grega  $\Omega$  (Ômega)

É o menor tempo de execução em uma entrada de tamanho  $N$

É pouco usado, por ter aplicação em poucos casos.

Ex.:

Se realizada uma busca por um número em uma lista de  $N$  números a complexidade no melhor caso é  $f(N) = \Omega(1)$ , pois assume-se que o número estaria logo na cabeça da lista.

## Caso Médio

Definido pela letra grega  $\theta$  (Theta)

Difícil de se determinar na maioria dos casos

Deve-se obter a média dos tempos de execução de todas as entradas de tamanho N e baseado em probabilidade de determinada condição ocorrer.

Ex.:

Se realizada uma busca por um número em uma lista de N números a complexidade no caso médio é  $f(N) = \theta((N+1)/2)$ , pois assume-se que o número estaria em qualquer posição.

## Caso Pior

Mais utilizado

Representado pela letra grega  $O$  (*ômicron*)

Baseia-se no maior tempo de execução sobre todas as entradas de tamanho  $N$ .

Ex.:

Se realizada uma busca por um número em uma lista de  $N$  números a complexidade no caso pior é  $f(N) = O(N)$ , pois assume-se que o número estaria na última posição.

# Cálculo de Complexidade

Para determinar a função que representa a complexidade de um algoritmo (ou seu **Comportamento Assintótico**) deve-se considerar as operações de maior impacto do algoritmo, como segue:

Operação primitivas	Custo na Complexidade
Execução de um operador	1
Execução de uma Comparação	1
Execução de acesso a índice	1
Execução de chamada de função	Mais de 1 (depende)
Retorno de função	1



# Cálculo de Complexidade

Abordagem consiste simplesmente na contagem de operações realizadas.

Linha	Código
1	function x();
2	{ int a, b;
3	a = 10,
4	b = 5;
5	If ( a < b)
6	a++;
7	else
8	b++;
9	}

Complexidade
-
-
1
1
1
2
-
2
-

Total: 5

Complexidade  
Constante.

$$\Omega(n) = 5$$

$$\Theta(n) = 5$$

$$O(n) = 5$$

# Cálculo de Complexidade

Exemplo:

Linha	Código
1	function x();
2	{ int a, b;
3	scanf("%d",&a);
4	scanf("%d",&b);
5	If ( a < b)
6	a=0;
7	else
8	b++;
9	}

Complexidade
-
-
2 (?)
2 (?)
1
1
-
2
-

Complexidade  
Constante.

Total: ?

$$\Omega(n) = 6$$

$$\Theta(n) = ?$$

$$O(n) = 7$$

# Cálculo de Complexidade

## Exemplo:

Linha	Código	Complexidade
1	function x();	-
2	{ int a, b;	-
3	scanf("%d",&a);	2 (?)
4	for (b=1; b<a; b++)	$1 + (1 * a) + (2 * (a - 1))$
5	if ( b < 10 )	$1 * (a - 1)$
6	b = (b*2)-1;	$3 * (1..9)$
7	else	-
8	b = (b+1)/(a*2);	$4 * (1... \infty)$
9	}	-

Complexidade  
Constante.

Complexidade	Total
$\Omega(a) =$ Caso melhor  $2 \leq a < 10$	$2 + 1 + (1 * a) + (2 * (a - 1))$ $+ 1 * (a - 1) + 3 * (9)$ <b>Para <math>a = 9</math>, temos:</b> <b>63</b>
$\Theta(a) =$ Caso médio	?????
$O(a) =$ Caso pior  Sendo $a \geq 10$	$2 + 1 + (1 * a) + (2 * (a - 1))$ $+ 1 * (a - 1) + 3 * (9) +$ $4 * (1)$ <b>Para <math>a = 10</math>, temos:</b> <b>67</b>

# Cálculo de Complexidade

## Complexidade Constante

O algoritmo reage de modo bastante previsível.

O tempo de execução não varia com a quantidade de dados processados



# Cálculo de Complexidade

Exemplo:

Complexidade  
Constante.

Linha	Código	Complexidade
1	function x();	-
2	{ int c[11]={0,1,1,1,1,1,1,1,1,1,1};	11 * 2
3	int b;	-
4	for (b=1; b<11; b++)	1+(1*11) +(2*10)
5	if ( c[b]>0 )	2 * 10
6	b= (b*2)-1;	3 * ?
7	else	-
8	b = b/2;	2 * ?
9	}	-

Complexidade	Total
$\Omega(a) =$ Caso melhor	22 + 32 + 20 + 20 <b>94</b>
$\Theta(a) =$ Caso médio	22 + 32 + 20 + 25 <b>99</b>
$O(a) =$ Caso pior	22 + 32 + 20 + 30 <b>104</b>

# Cálculo de Complexidade

Exemplo:

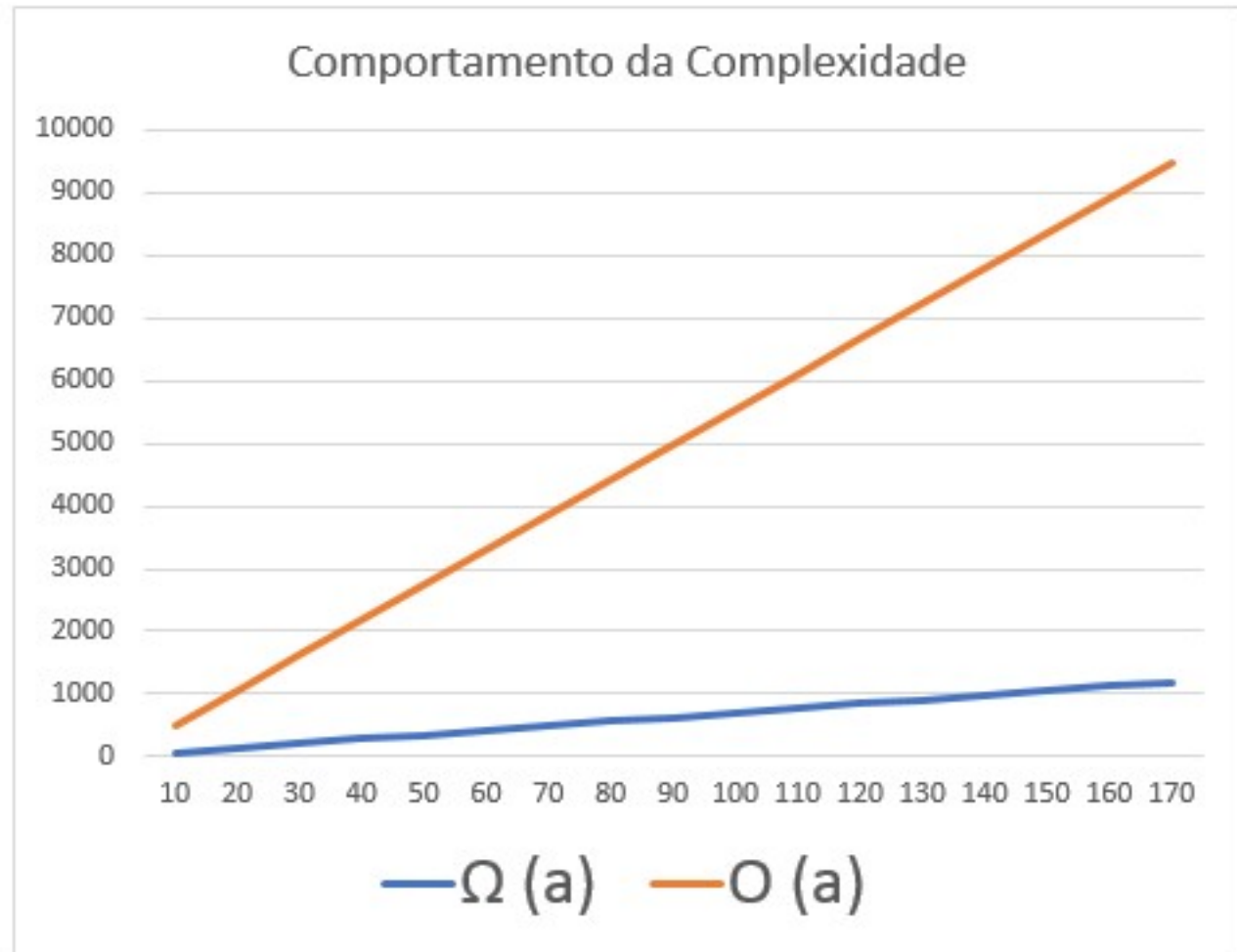
Complexidade  
Não Constante.

Linha	Código
1	function x(int a, int c[]);
2	{
3	int b;
4	for (b=1; b<a; b++)
5	if ( c[b]>0 )
6	b= (b*2)-1;
7	else
8	b = b/2;
9	}

Complexidade
-
-
-
$1 + (1 * a) + 2 * (a - 1)$
$2 * (a - 1)$
$3 * ?$
-
$2 * ?$
-

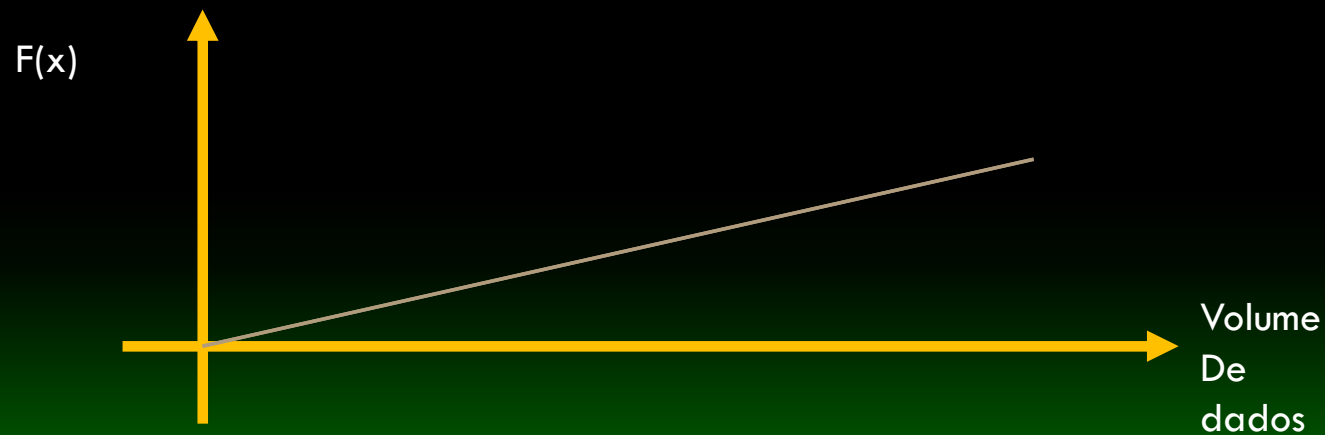
Complexidade	Total
$\Omega(a) =$ Caso melhor	$1 + (1 * a) + 2 * (a - 1) + 2 * (a - 1) + 2 * a$ $= 7a - 3$
$\Theta(a) =$ Caso médio	???
$O(a) =$ Caso pior	$1 + (1 * a) + 2 * (a - 1) + 2 * (a - 1) + 3 * a$ $= 8a - 3$

a	$\Omega(a)$	$O(a)$
10	67	503
20	137	1063
30	207	1623
40	277	2183
50	347	2743
60	417	3303
70	487	3863
80	557	4423
90	627	4983
100	697	5543
110	767	6103
120	837	6663
130	907	7223
140	977	7783
150	1047	8343
160	1117	8903
170	1187	9463



# Complexidade Polinomial Linear

O algoritmo reage de modo bastante previsível mas com tempo de execução variando conforme a quantidade de dados processados



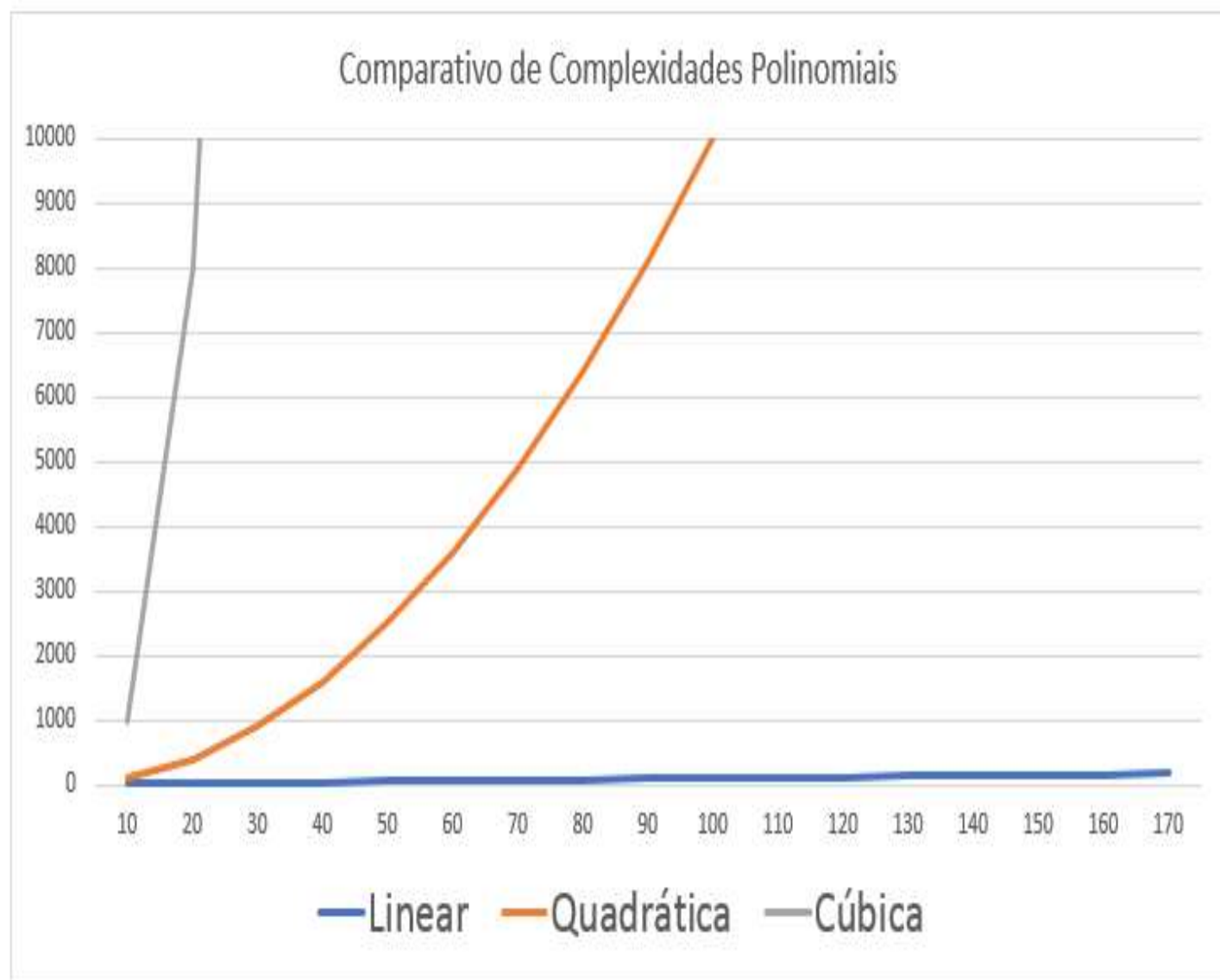
$F(x)$  – complexidade do algoritmo



# Complexidade Polinomial

O algoritmo reage de modo bastante previsível mas com tempo de execução variando conforme a quantidade de dados processados em um ritmo de degradação elevado.

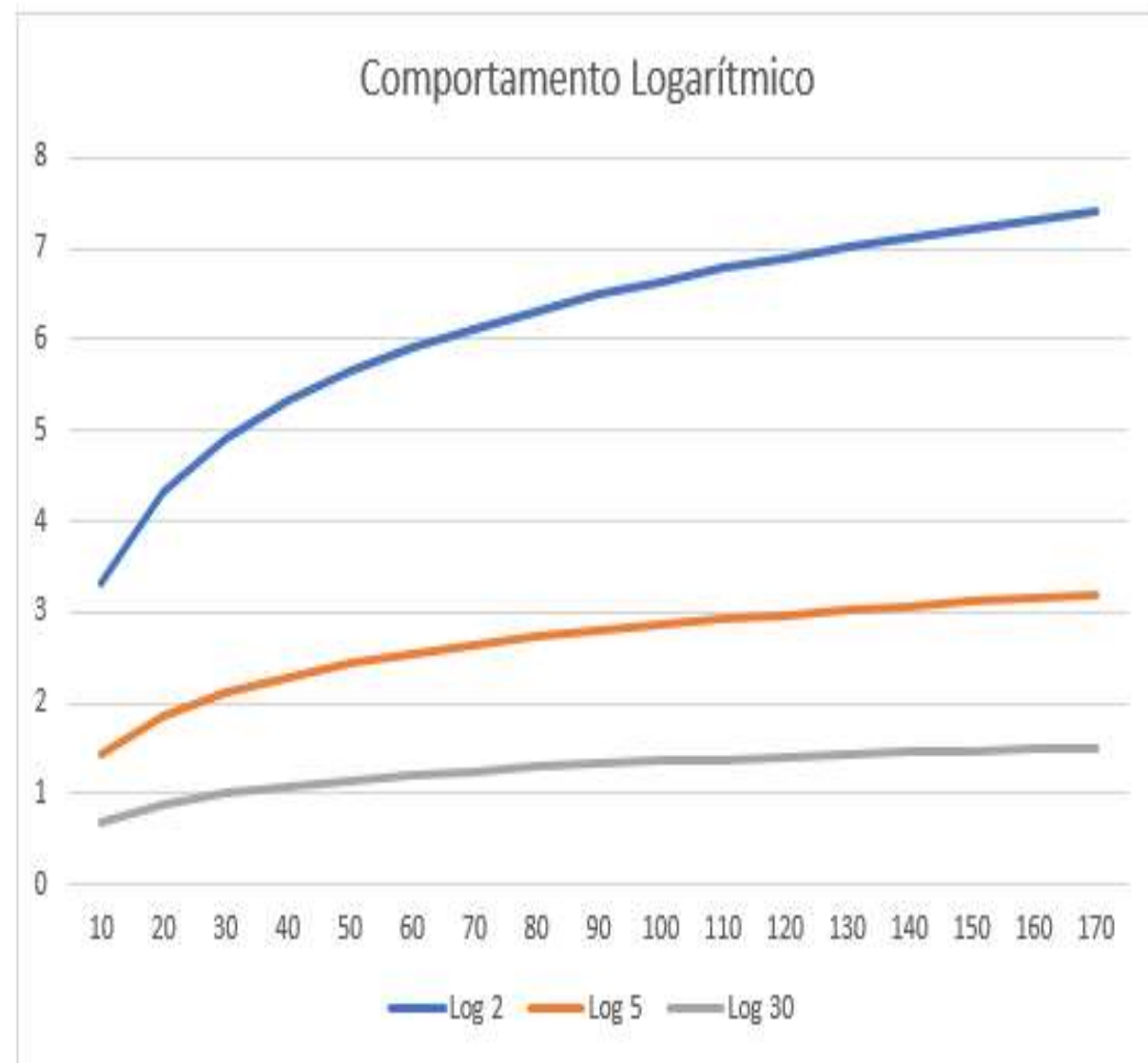
a	Linear	Quadrática	Cúbica
10	10	100	1000
20	20	400	8000
30	30	900	27000
40	40	1600	64000
50	50	2500	125000
60	60	3600	216000
70	70	4900	343000
80	80	6400	512000
90	90	8100	729000
100	100	10000	1000000
110	110	12100	1331000
120	120	14400	1728000
130	130	16900	2197000
140	140	19600	2744000
150	150	22500	3375000
160	160	25600	4096000
170	170	28900	4913000



# Complexidade Logarítmica

O algoritmo reage com uma moderada degradação do desempenho, refletindo um comportamento logarítmico

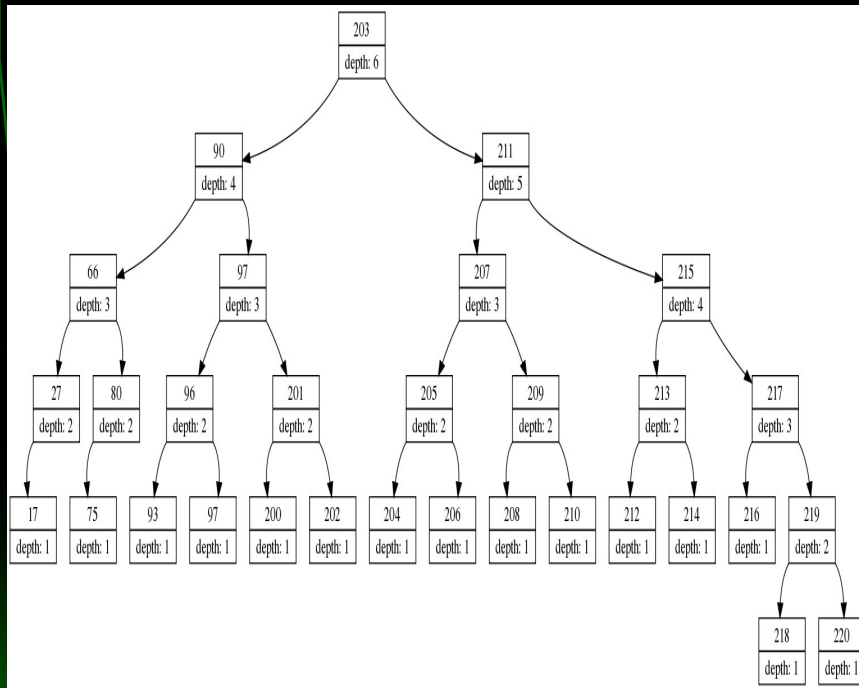
N	Log <sub>2</sub>	Log <sub>5</sub>	Log <sub>30</sub>
10	3,3219281	1,430676558	0,676992493
20	4,3219281	1,861353116	0,88078754
30	4,9068906	2,113282753	1
40	5,3219281	2,292029674	1,084582587
50	5,6438562	2,430676558	1,150189938
60	5,9068906	2,543959311	1,203795047
70	6,129283	2,639738513	1,249117521
80	6,3219281	2,722706232	1,288377634
90	6,4918531	2,795888947	1,323007507
100	6,6438562	2,861353116	1,353984985
110	6,7813597	2,92057266	1,382007522
120	6,9068906	2,974635869	1,407590094
130	7,0223678	3,024369199	1,431123779
140	7,129283	3,070415071	1,452912568
150	7,2288187	3,113282753	1,473197445
160	7,3219281	3,15338279	1,492172681
170	7,4093909	3,191050986	1,509997175



Exemplos :

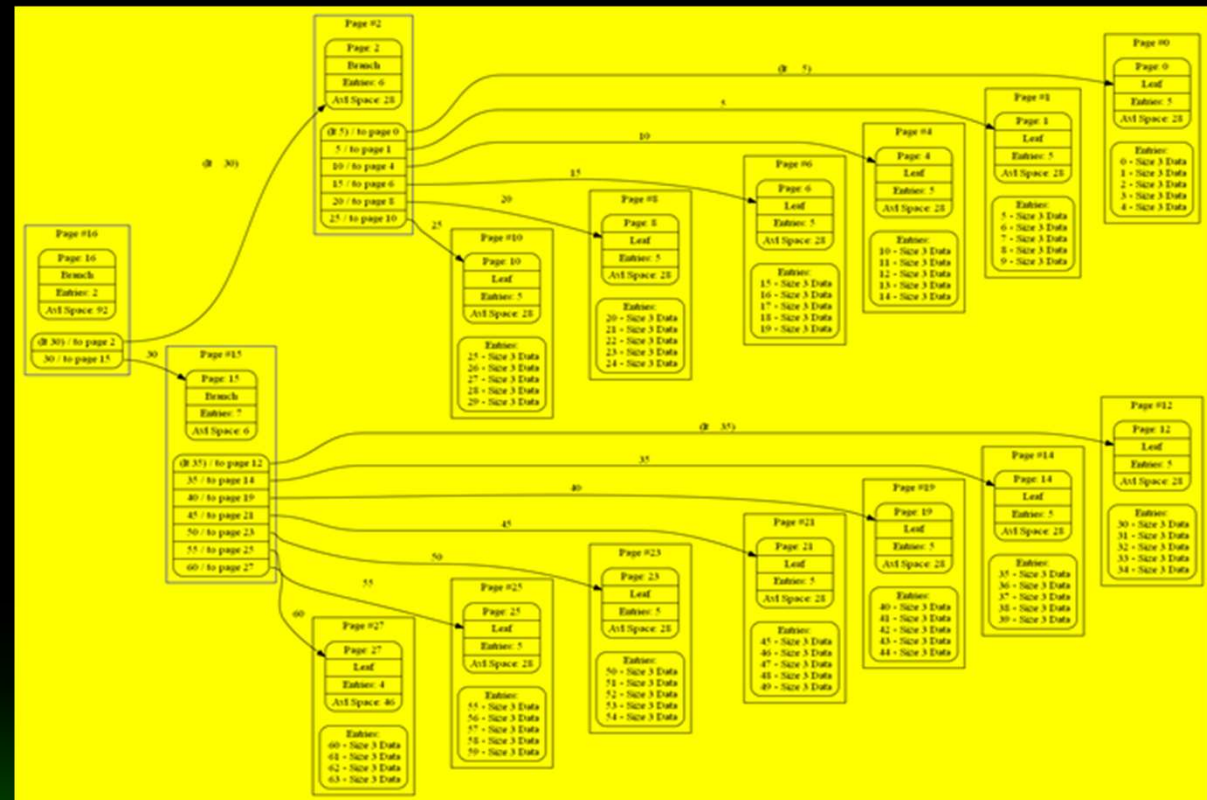
Pesquisa em **Árvore AVL**

**Base Logarítmica 2**



Pesquisa em **Árvore B**

**Base Logarítmica ?**



# Complexidade Exponencial & Complexidade Fatorial

O algoritmo reage fortemente a degradação do desempenho, refletindo rapidamente devido a elevação da quantidade de dados em processamento.

## A Invenção do Xadrez (“o poder da progressão exponencial”)

Existem diversas mitologias associadas à criação do xadrez, sendo uma das mais famosas aquela que atribui ao jovem sacerdote indiano Lahur Sessa.

No livro *O Homem que Calculava* do escritor e matemático brasileiro Malba Tahan, conta que numa província indiana de Taligana havia um poderoso rajá que estava em constante depressão e passou a descuidar-se de si e do reino.

Certo dia o rajá foi visitado por Sessa, que apresentou-lhe um tabuleiro com 64 casas brancas e pretas intercaladas e com diversas peças que representavam tropas do exército: infantaria, cavalaria, carros de combate, condutores de elefantes, o vizir e o próprio rajá. O sacerdote explicou que a prática do jogo daria conforto espiritual e cura para a depressão.

O rajá, agradecido, ofereceu uma recompensa a Lahur Sessa por sua invenção e o brâmane pediu simplesmente **um grão de trigo para a primeira casa do tabuleiro, dois para a segunda, quatro para a terceira, oito para a quarta e assim sucessivamente até a última casa**. Espantado com a modéstia do pedido, o rajá ordenou que fosse pago imediatamente a quantia em grãos que fora pedida. Após os cálculos, os sábios do rajá ficaram atônitos com o resultado que a quantidade de grãos atingiu



Casa	N. grãos
1	1
2	2
3	4
4	8
5	16
6	32
7	64
8	128
9	256
10	512
11	1.024
12	2.048
13	4.096
14	8.192
15	16.384
16	32.768
17	65.536
18	131.072
19	262.144
20	524.288
21	1.048.576

Casa	N. grãos
20	524.288
21	1.048.576
22	2.097.152
23	4.194.304
24	8.388.608
25	16.777.216
26	33.554.432
27	67.108.864
28	134.217.728
29	268.435.456
30	536.870.912
31	1.073.741.824
32	2.147.483.648
33	4.294.967.296
34	8.589.934.592
35	17.179.869.184
36	34.359.738.368
37	68.719.476.736
38	137.438.953.472
39	274.877.906.944
40	549.755.813.888
41	1.099.511.627.776

Casa	N. grãos
37	68.719.476.736
38	137.438.953.472
39	274.877.906.944
40	549.755.813.888
41	1.099.511.627.776
42	2.199.023.255.552
43	4.398.046.511.104
44	8.796.093.022.208
45	17.592.186.044.416
46	35.184.372.088.832
47	70.368.744.177.664
48	140.737.488.355.328
49	281.474.976.710.656
50	562.949.953.421.312
51	1.125.899.906.842.620
52	2.251.799.813.685.250
53	4.503.599.627.370.500
54	9.007.199.254.740.990
55	18.014.398.509.482.000
56	36.028.797.018.964.000
57	72.057.594.037.927.900
58	144.115.188.075.856.000
59	288.230.376.151.712.000
60	576.460.752.303.423.000
61	1.152.921.504.606.850.000
62	2.305.843.009.213.690.000
63	4.611.686.018.427.390.000
64	9.223.372.036.854.780.000

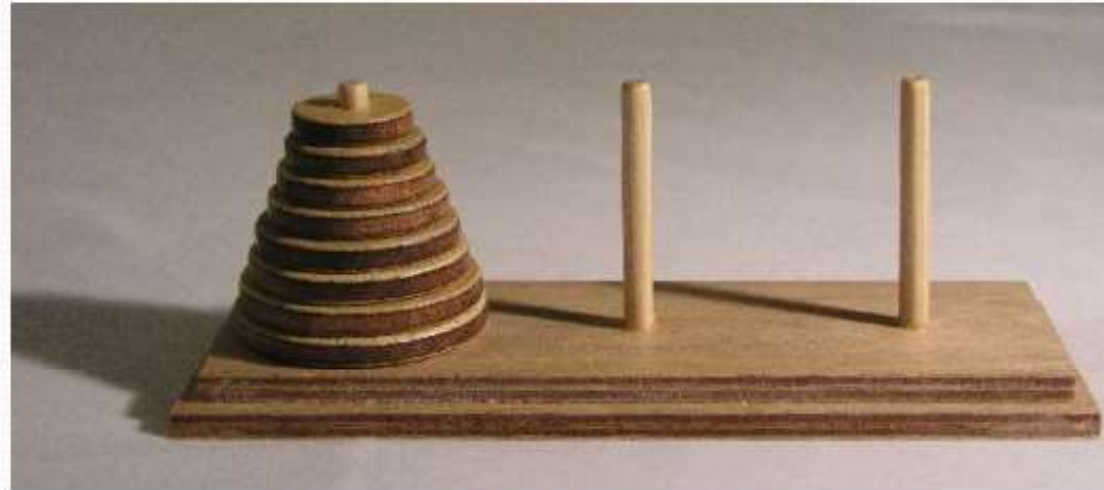
Se um grão de arroz pesa 1 g, o total equivale a 18.446.744.073.709 toneladas

Colocados em caminhões pesados (capacidade de 14 toneladas) seriam necessários cerca de 1.3 trilhões de caminhões ou 92 bilhões de aviões com capacidade 200 ton.



Exemplo :

## Problema da Torre de Hanoi:



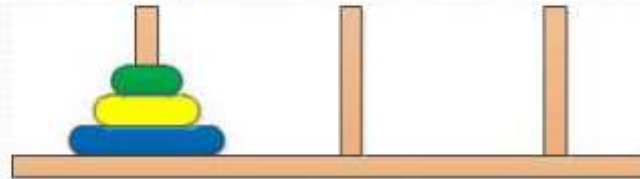
- Dado um conjunto de  $n$  discos dispostos em ordem decrescente de tamanho, o objetivo é mover a pilha para outro pino usando um terceiro como auxiliar.
- **Restrições:**
  - Somente um disco pode ser movido a cada instante.
  - Cada movimento consiste em retirar o disco do topo de uma pilha e movê-lo ao topo de outra pilha.
  - Nenhum disco pode ser colocado sobre outro que seja menor do que ele.

Exemplo :

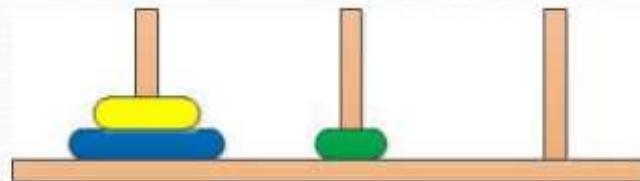
- **Problema da Torre de Hanoi:**

➤ Exemplo: 3 discos

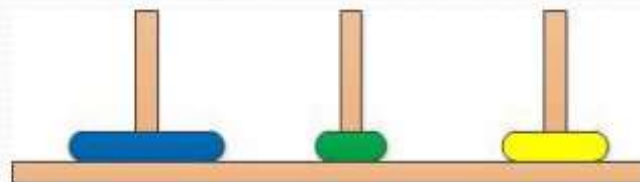
Início



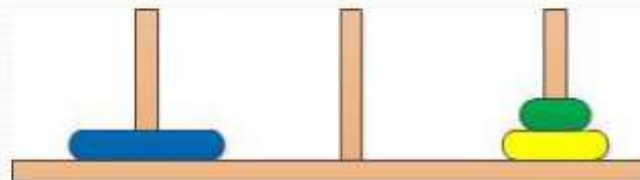
1



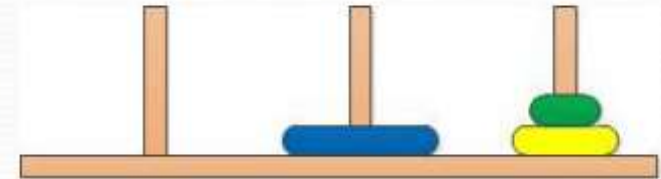
2



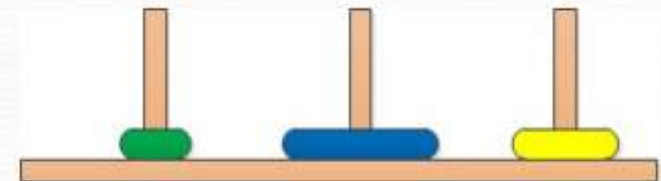
3



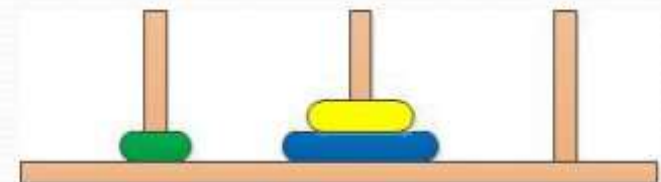
4



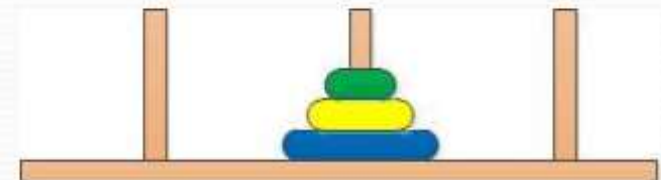
5



6



7



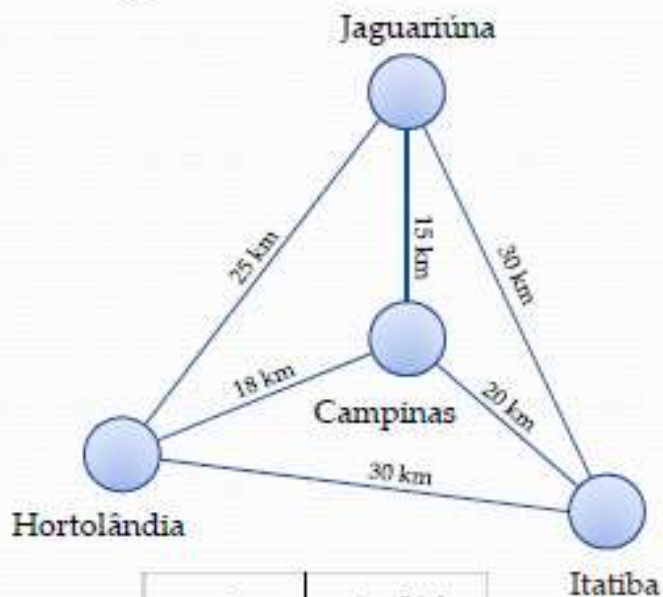
➤ **Caso geral:** o número mínimo de movimentos é igual a  $2^n - 1$ .

## Exemplo :

Problema de Grafo.

Caminho Hamiltoniano

- **Problema do Caixeiro Viajante:** Dado um mapa com  $n$  cidades e a distância entre cada par de cidades, é possível obter um percurso para um vendedor de modo que ele o complete dentro de uma dada quilometragem, visitando cada cidade uma única vez e retornando ao ponto de partida?



nós	$(n-1)!$
4	6
10	362880

Qual é a rota de menor custo?

	J	J	J	J	J	J
	C	C	H	H	I	I
Rotas	H	I	I	C	C	H
	I	H	C	I	H	C
	J	J	J	J	J	J
Custo	93	90	90	93	93	93

A solução é obtida analisando-se as  $(n-1)!$  possibilidades.

Qual é a ordem de complexidade?


Exponencial:  $O(c^n)$

É a melhor solução encontrada?

Nunca se comprovou!

**Problema Não-deterministicamente Polinomial (NP)**

## Comparação entre Classes de Complexidade Algorítmica

<p>Menor complexidade</p>  <p>Maior complexidade</p>	Notação	Nome
	$O(1)$	Ordem constante
	$O(\log n)$	Ordem logarítmica
	$O([\log n]^c)$	Ordem poli-logarítmica
	$O(n)$	Ordem linear
	$O(n \cdot \log n)$	Ordem linear-logarítmica
	$O(n^2)$	Ordem quadrática
	$O(n^3)$	Ordem cúbica
	$O(n^c)$	Ordem polinomial
	$O(c^n)$	Ordem exponencial
	$O(n!)$	Ordem fatorial



## Comparação entre Classes de Complexidade Algorítmica

Execution times of a machine that executes  $10^9$  steps by second ( $\sim 1$  GHz), as a function of the algorithm cost and the size of input  $n$ :

Size	$\log_2 n$	$n$	$n \log_2 n$	$n^2$	$n^3$	$2^n$
10	3.322 ns	10 ns	33 ns	100 ns	1 $\mu s$	1 $\mu s$
20	4.322 ns	20 ns	86 ns	400 ns	8 $\mu s$	1 ms
30	4.907 ns	30 ns	147 ns	900 ns	27 $\mu s$	1 s
40	5.322 ns	40 ns	213 ns	2 $\mu s$	64 $\mu s$	18.3 min
50	5.644 ns	50 ns	282 ns	3 $\mu s$	125 $\mu s$	13 days
100	6.644 ns	100 ns	664 ns	10 $\mu s$	1 ms	$40 \cdot 10^{12}$ years
1000	10 ns	1 $\mu s$	10 $\mu s$	1 ms	1 s	
10000	13 ns	10 $\mu s$	133 $\mu s$	100 ms	16.7 min	
100000	17 ns	100 $\mu s$	2 ms	10 s	11.6 days	
1000000	20 ns	1 ms	20 ms	16.7 min	31.7 years	

Os limites conhecidos de solução de um problema

**Cota Superior (Upper Bound):** estabelecido pelo melhor algoritmo até o Momento para determinado problema.

Exemplo: Multiplicação de duas matrizes quadradas  $n \times n$ .

- O algoritmo trivial tem complexidade  $O(n^3)$ .
- **Strassen** em 1969 reduziu a complexidade para  $O(n^{\log 7})$
- **Coppersmith** e **Winograd** melhoraram ainda para  $O(n^{2.376})$

Shmuel Winograd  
Cientista da Computação



Don Coppersmith  
Criptógrafo e Matemático



Arnold Strassen  
Matemático

## Classificação de Problemas:

**Problemas polinomiais (P):** problemas cuja melhor solução demanda algoritmos com complexidade (assintótica) de ordem até polinomial  $O(n^c)$ .

**Problemas não-deterministicamente polinomiais (NP):** problemas cuja melhor solução conhecida demanda algoritmos com complexidade assintótica de ordem exponencial  $O(c^n)$ .

**Problemas NP-Completos (NPC):** subconjunto problemas de NP que se solucionados com menor complexidade causam redução de complexidade de outros problemas.

# Todos os Problemas Computáveis

Constantes

Logarítmicos

Não Determinísticos Polinomiais (NP)

Polinomiais (P)

Não Determinísticos  
Polinomiais Completo (NPC)



## Problemas Redutíveis :

problemas que podem ser transformados/substituídos em outros, com objetivo de redução da complexidade de solução.

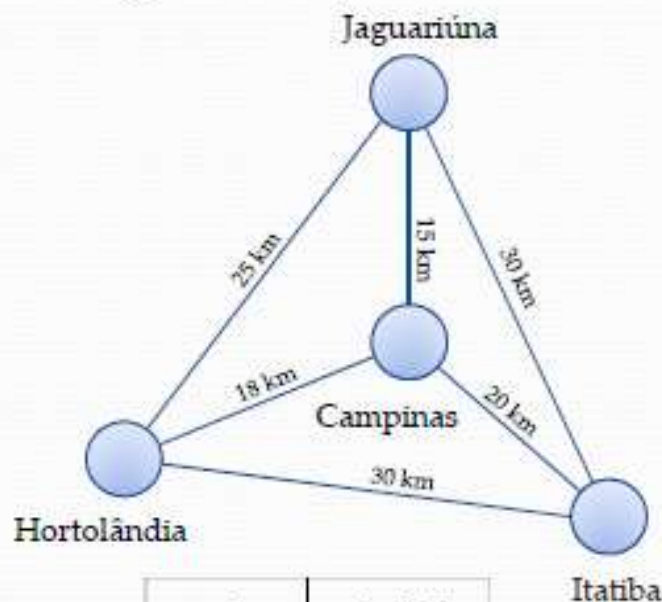
Um problema Y é redutível a um problema X se Y for um subproblema ou caso particular de X.

## Heurística:

é a utilização de uma lógica algorítmica que busca solucionar um problema considerando alguma simplificação ou redução de dados, visando a viabilidade temporal de sua execução

## Exemplo :

- **Problema do Caixeiro Viajante:** Dado um mapa com  $n$  cidades e a distância entre cada par de cidades, é possível obter um percurso para um vendedor de modo que ele o complete dentro de uma dada quilometragem, visitando cada cidade uma única vez e retornando ao ponto de partida?



nós	$(n-1)!$
4	6
10	362880

Qual é a rota de menor custo?

	J	J	J	J	J	J
	C	C	H	H	I	I
Rotas	H	I	I	C	C	H
	I	H	C	I	H	C
	J	J	J	J	J	J
Custo	93	90	90	93	93	93

A solução é obtida analisando-se as  $(n-1)!$  possibilidades.

Qual é a ordem de complexidade?

Exponencial:  $O(c^n)$

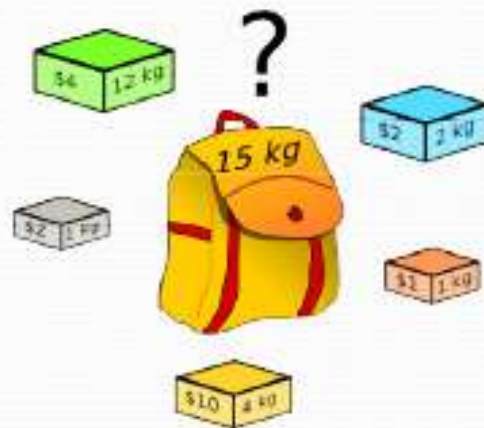
É a melhor solução encontrada?

Nunca se comprovou!

**Problema Não-deterministicamente Polinomial (NP)**

Exemplo :

- **Problema da mochila (knapsack):** é um problema de otimização combinatória. O objetivo é que se preencha uma mochila com um conjunto de itens tal que se atinja o maior valor total, não ultrapassando o peso máximo suportado.



Problema cuja melhor  
solução conhecida  
demanda um  
algoritmo de ordem  
exponencial



Problema NP

É interessante observar que  
diversos outros problemas  
se reduzem ao problema  
da mochila



- Investimento de capitais
- Corte e empacotamento
- Orçamento
- Criptografia de chave pública

Desta forma, se  
encontrarmos uma  
solução em tempo  
polinomial para o  
problema da mochila,  
teremos uma solução  
 $O(n^c)$  para todos os  
outros problemas da  
mesma classe.



O problema da Mochila é  
**NP-Completo**

**Problemas tratáveis:** aqueles cuja solução demanda um algoritmo de ordem polinomial ou menor

Também, aqueles que ainda não se provou que a melhor solução demanda algoritmos de ordem exponencial ou maior.

**Problemas não-tratáveis:** já se comprovou que a melhor solução possível demanda um algoritmo de ordem exponencial ou ainda mais custoso.

## Exemplo:

Suponha que temos cinco algoritmos A1, A2, A3, A4, A5 com as seguintes complexidades de tempo:

Algoritmo	Complexidade
A1	$n$
A2	$n \log_2 n$
A3	$n^2$
A4	$n^3$
A5	$2^n$

Considere  $\rightarrow T = \text{complexidade} * UT$

Computador hipotético.

A unidade de tempo:

$1UT = 1ms = 10^{-3}$  segundos

Alguns cálculos:

**Para  $T = 1$  (1 segundo de execução)**

Algoritmo A1 temos,  $1 = n * 10^{-3} \rightarrow n = 1.000$

Algoritmo A3 temos,  $1 = n^2 * 10^{-3} \rightarrow n = 31,6$

Algoritmo A5 temos,  $1 = 2^n * 10^{-3} \rightarrow n = 9$

**Para  $T = 60$  (1 minuto de execução)**

Algoritmo A1 temos,  $60 = n * 10^{-3} \rightarrow n = 60.000$

Algoritmo A3 temos,  $60 = n^2 * 10^{-3} \rightarrow n = 244,9$

Algoritmo A5 temos,  $60 = 2^n * 10^{-3} \rightarrow n = 15$

A tabela a seguir mostra o tamanho de problemas que podem ser resolvidos em 1 segundo, em 1 minuto e em 1 hora para cada algoritmo em um computador hipotético:

Algoritmo	Complexidade	1 seg	1 min	1 hora
A1	$n$	1 000 dados	60.000 dados	3.600.000 dados
A2	$n \log_2 n$	1 40 dados	4893 dados	200.000 dados
A3	$n^2$	31,6 dados	244,9 dados	1 897,4 dados
A4	$n^3$	10 dados	39,2 dados	1 53,3 dados
A5	$2^n$	9 dados	15 dados	21 dados



Supondo que a próxima geração de computadores seja **dez vezes mais rápida** que atual. No mesmo tempo T, quanto se espera de aumento no volume processado ?

Algoritmo A1 temos,

$$\text{Para Máquina Atual} \quad T = S * 10^{-3}$$

$$\text{Para Máquina Futura} \quad T = (n * 10^{-3})/10$$

$$S * 10^{-3} = (n * 10^{-3})/10 \quad \rightarrow n = S * 10$$

**S** : volume na máquina antiga

**n**: volume na máquina nova

Algoritmo A3 temos,

$$\text{Para Máquina Atual} \quad T = S^2 * 10^{-3}$$

$$\text{Para Máquina Futura} \quad T = (n^2 * 10^{-3})/10$$

$$S^2 * 10^{-3} = (n^2 * 10^{-3})/10 \quad \rightarrow n = S * 3$$

Algoritmo A5 temos,

$$\text{Para Máquina Atual} \quad T = 2^S * 10^{-3}$$

$$\text{Para Máquina Futura} \quad T = (2^n * 10^{-3})/10$$

$$2^S * 10^{-3} = (2^n * 10^{-3})/10$$

$$2^S = 2^n / 10$$

$$2^n = 2^S * 10$$

$$\log 2^n = \log 2^S + \log 10 \quad \rightarrow n = S + 3$$



A próxima geração de computadores seja **dez vezes mais rápida** que a atual. A tabela abaixo mostra o aumento no tamanho do problema que os algoritmos poderão resolver no mesmo tempo.

Algoritmo	Complexidade	Volume processado Maq. Atual	Volume processado Maq. Futura
A1	$n$	S1	$S1 * 10$
A2	$n \log_2 n$	S2	$S2 * 10$ (aprox.)
A3	$n^2$	S3	$S3 * 3$
A4	$n^3$	S4	$S4 * 2$
A5	$2^n$	S5	$S5 + 3$

## Exercício

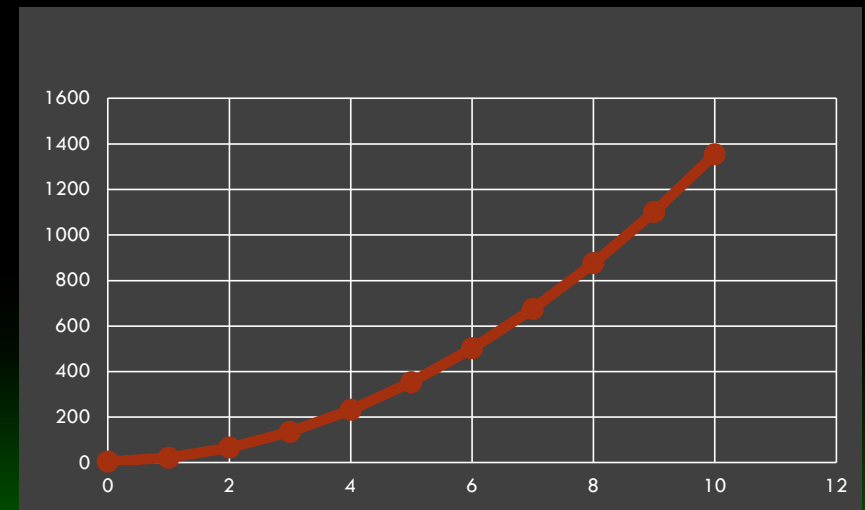
1) Qual a função de complexidade do código abaixo para o caso pior:

```
/* CALCULO SOBRE A MATRIZ (4*4) */  
soma = 0;  
menor = 999;  
for (linha = 0; linha <=3; linha++)  
{  
    for (coluna = 0; coluna <=3; coluna++)  
    {  
        soma = soma + M[linha][coluna];  
  
        if (M[linha][coluna]<menor)  
        {  
            menor = M[linha][coluna];  
        }  
    }  
}
```

Resposta: 232

Para matriz N\*N) a função de complexidade é

$$F(n) = 13*n^2 + 5*n + 4$$



## Exercício

2) Considere a necessidade um programa que obtenha o menor valor de um vetor de 100 elementos.

Faça o algoritmo e calcule a complexidade nas situações:

1. ☐ O vetor está sempre cheio
2. ☐ O vetor não está sempre cheio
3. ☐ O vetor está ordenado
4. ☐ O vetor não está ordenado
5. ☐ O algoritmo não é recursivo
6. ☐ O algoritmo é recursivo

## Exercício

3) Suponha que tenhamos três algoritmos (A1, A2 e A3) com complexidade assintótica  $O(n^2)$  para resolução do mesmo problema. As funções de complexidade destes algoritmos são:

$$A1 \rightarrow f(n) = 2*n^2 + 9*n$$

$$A2 \rightarrow g(n) = 9*n^2 + 15*n$$

$$A3 \rightarrow h(n) = n^2 + 300$$

Faça o gráfico para a análise comportamental de cada algoritmo diante do crescimento acelerado do volume  $n$  de dados.

## Exercício

4) Procure a lógica e desenvolva em C o algoritmo eficiente de multiplicação de Matrizes Quadradas de **Strassen**

Acesse:

<http://mathworld.wolfram.com/StrassenFormulas.html>

Arnold Strassen

Matemático

