Capítulo1- O papel dos Algoritmos na Computação

Prof. Manoel Ribeiro

1.1 Introdução

O nome da disciplina é Complexidade de Algoritmos.

O que é então um algoritmo?

Informalmente, um algoritmo é um procedimento computacional bem definido que:
recebe um conjunto de valores como entrada e
produz um conjunto de valores como saída.

Um algoritmo é uma ferramenta para resolver um determinado problema computacional.

A descrição do problema define a relação que deve existir entre a entrada e a saída do algoritmo.

Exemplo: Problema da Ordenação

Problema: rearranjar um vetor $A[1 \ldots n]$ em ordem crescente.

Entrada:

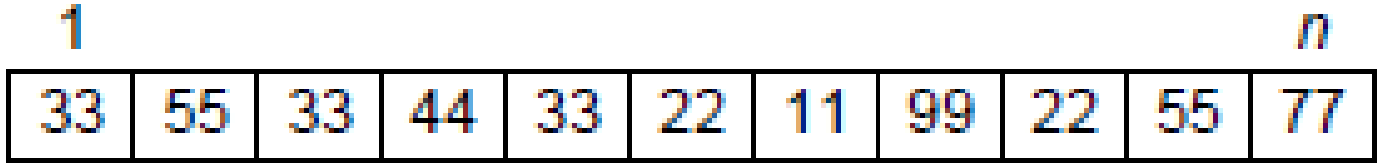
1										n
33	55	33	44	33	22	11	99	22	55	77

Saída:

1										n
11	22	22	33	33	33	44	55	55	77	99

Instância de um problema

Dizemos que o vetor



é uma instância do problema de ordenação.

Em geral, uma instância de um problema é um conjunto de valores que serve de entrada para o problema (respeitando as restrições impostas na descrição deste).

Problemas de Ordenação

Como muitos programas a utilizam como etapa intermediária, a ordenação é uma operação fundamental em ciência da computação.

Existem um grande número de bons algoritmos de ordenação à nossa disposição

O melhor algoritmo para determinada aplicação depende – entre outros fatores
Do **número de itens** a ordenar
Do **grau de ordenação** já apresentado por esses itens
Das possíveis restrições aos valores dos itens
Da arquitetura do computador
Tipo de dispositivo de armazenamento: memória principal ou memória secundária.

Iremos analisar alguns dos principais algoritmos de ordenação existentes.

ENCO01070 – Complexidade de Algoritmos

O que veremos nesta disciplina?

Como estimar a quantidade de **recursos** (tempo, memória) que um algoritmo consome/gasta = análise de complexidade

Como provar a "corretude" de um algoritmo

Como projetar algoritmos eficientes (= rápidos) para vários problemas computacionais

“Corretude” de um algoritmo

Diz-se que um algoritmo é **correto** se, para toda instância de entrada, ele parar com a saída correta.

Dizemos que um algoritmo correto **resolve** o problema computacional dado.

Um algoritmo incorreto poderia não parar em algumas instâncias de entrada ou poderia parar com uma resposta incorreta. (Bugs, como a condição de parada ser zero, após uma operação com números reais).

Ao contrário do que se poderia esperar, às vezes os algoritmos incorretos podem ser úteis, se pudermos controlar sua taxa de erros. Porém, de modo geral, nos concentraremos apenas em algoritmos corretos.

Usaremos **Invariantes de Laço** para nos ajudar a entender por que um algoritmo é correto.

A importância dos algoritmos para a computação

- Exemplos de aplicações para o uso/desenvolvimento de algoritmos “eficientes”?
  - projetos de genoma de seres vivos
  - rede mundial de computadores
  - comércio eletrônico
  - planejamento da produção de indústrias
  - logística de distribuição
  - computação científica, imagens médicas
  - *games* e filmes, ...

Duas características comuns aos exemplos de aplicação apresentados

1. Tem muitas soluções candidatas, a grande maioria não resolve o problema que temos em mãos. Encontrar uma solução que o resolva, ou uma solução que seja a “melhor”, pode representar um desafio significativo
2. Eles tem aplicações práticas. Dentre os problemas apresentados, o da determinação do caminho mais curto é o que fornece os exemplos mais fáceis. Empresa de transporte rodoviário tem interesse financeiro em determinar os caminhos mais curtos, porque caminhos menores resultam em menores custos de mão de obra e combustível; ou um nó de roteamento na Internet pode precisar encontrar o caminho mais curto da rede, para rotear uma mensagem com rapidez. Nesses casos a estrutura de dados é o Grafo.

Dificuldade intrínseca de problemas

- Infelizmente, existem certos problemas para os quais **não se conhece** algoritmos eficientes capazes de resolvê-los. Exemplos são os problemas $\mathcal{NP}$-completos.

  Curiosamente, **não foi provado** que tais algoritmos não existem! Interprete isso como um desafio para inteligência humana.

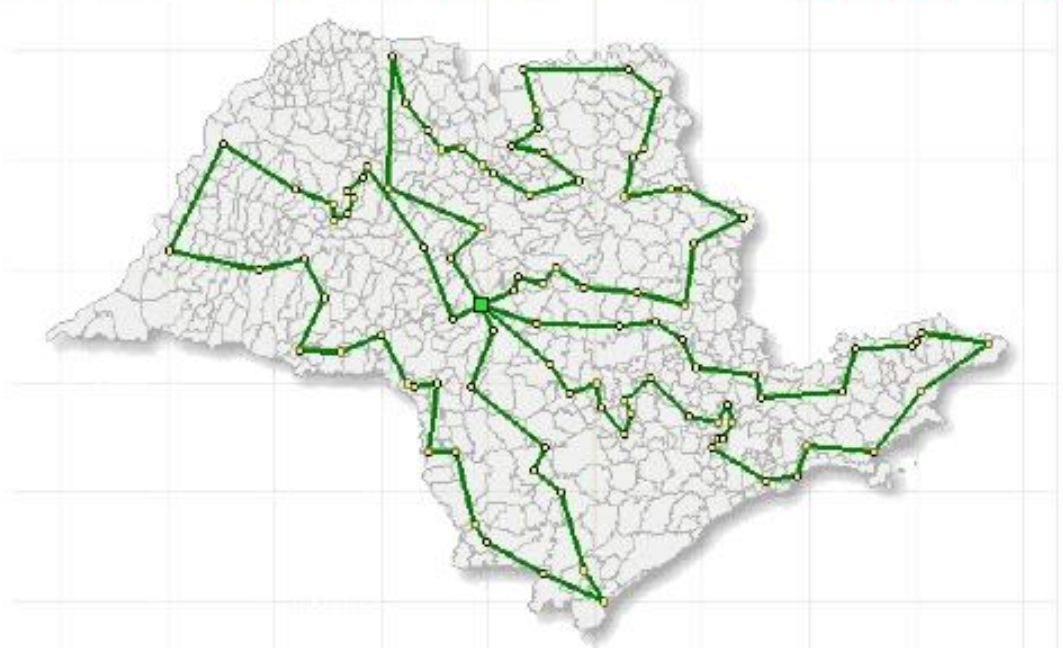
- Esses problemas tem a característica notável de que se um deles admitir um algoritmo "eficiente" então todos admitem algoritmos "eficientes".

- Por que devo me preocupar com problemas $\mathcal{NP}$-sei-lá-o-quê?

  Problemas dessa classe surgem em inúmeras situações práticas, como problemas $\mathcal{NP}$-difíceis.

Dificuldade intrínseca de problemas

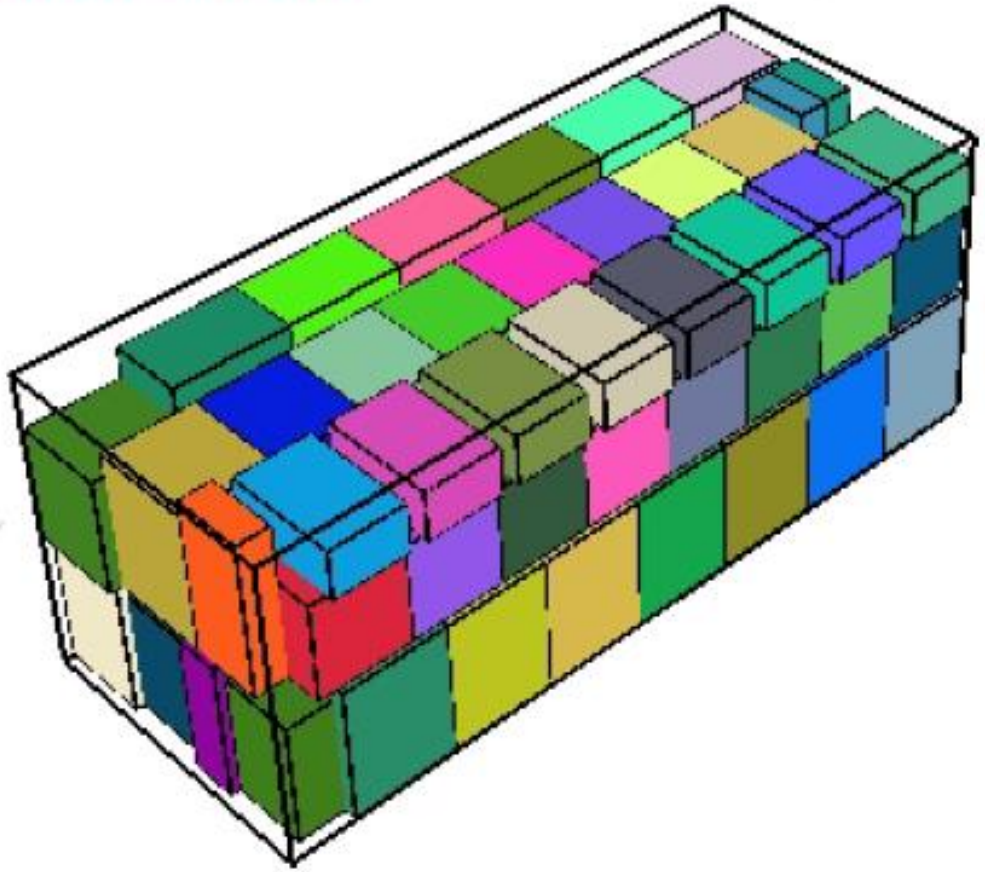
Exemplo de problema $\mathcal{NP}$-difícil: calcular as rotas dos caminhões de entrega de uma distribuidora de bebidas em São Paulo, minimizando a distância percorrida. (vehicle routing)



Qual a diferença desse problema para o da determinação do caminho mais curto, já apresentado, como um problema de fácil solução ?

Dificuldade intrínseca de problemas

Exemplo de problema $\mathcal{NP}$-difícil: calcular o número mínimo de *containers* para transportar um conjunto de caixas com produtos. (bin packing 3D)



Dificuldade intrínseca de problemas

Exemplo de problema $\mathcal{NP}$-difícil: calcular a localização e o número mínimo de antenas de celulares para garantir a cobertura de uma certa região geográfica. (facility location)

e muito mais...

É importante saber indentificar quando estamos lidando com um problema $\mathcal{NP}$-difícil!

1.2 Algoritmos e tecnologia

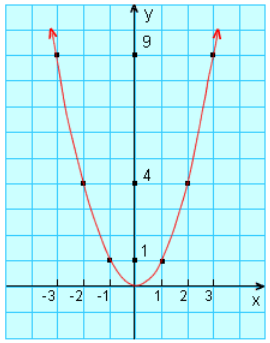
- O avanço da tecnologia permite a construção de máquinas cada vez mais rápidas. Isto possibilita que um algoritmo para determinado problema possa ser executado mais rapidamente.
- Paralelamente a isto, há o projeto/desenvolvimento de algoritmos "intrinsecamente mais eficientes" para determinado problema. Isto leva em conta apenas as características inerentes ao problema, desconsiderando detalhes de software/hardware.
- Vamos "comparar" estes dois aspectos através de um exemplo.

Algoritmos e tecnologia

Eficiência

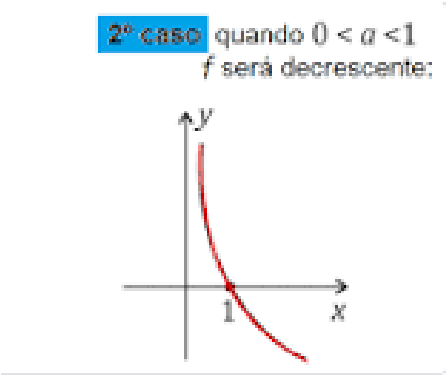
Algoritmos diferentes criados para resolver o mesmo problema muitas vezes são muito diferentes em termos de eficiência. No capítulo 2, estudaremos dois algoritmos de ordenação

1. **Ordenação por inserção**
   Leva aproximadamente $C_1 n^2$ para ordenar n itens.

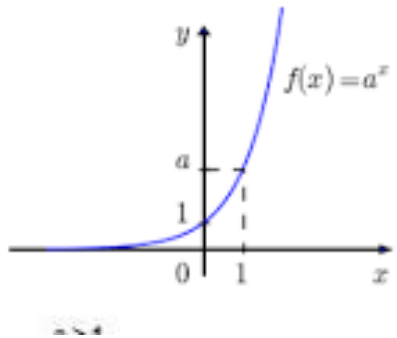


Função quadrática

2. **Ordenação por intercalação**
   Leva aproximadamente $C_2 n \lg n$ para ordenar n itens.



Função Logarítmica



Função exponencial

A ordenação por inserção normalmente tem um fator constante menor que a ordenação por intercalação, assim geralmente $c1 < c2$.

Veremos que os fatores constantes podem causar um impacto muito menor sobre o tempo de execução que a dependência do tamanho da entrada n. Se representarmos o tempo de execução da ordenação por inserção por $c1n \cdot n$ e o tempo de execução da ordenação por intercalação por $c2n \cdot \lg n$, veremos que, enquanto o fator do tempo de execução da ordenação por inserção é n, o da ordenação por intercalação é $\lg n$, que é muito menor (por exemplo, quando $n = 1.000$, $\lg n$ é aproximadamente 10 e quando n é igual a um milhão, $\lg n$ é, aproximadamente, só 20).

Embora a ordenação por inserção em geral seja mais rápida que a ordenação por intercalação para pequenos tamanhos de entradas, tão logo o tamanho da entrada *n* se torne grande o suficiente a vantagem da ordenação por intercalação de lg *n* contra *n* compensará com sobras a diferença em fatores constantes.

Independentemente do quanto *c*1 seja menor que *c*2, sempre haverá um ponto de corte além do qual a ordenação por intercalação será mais rápida.

Como exemplo concreto, vamos comparar um computador mais rápido (computador A) que executa a ordenação por inserção, algoritmo1, com um computador mais lento (computador B) que executa a ordenação por intercalação, algoritmo2.

Algoritmos e tecnologia

Exemplo: **ordenação de um vetor de n elementos**

- Suponha que os computadores A e B executam $1G$ e $10M$ instruções por segundo, respectivamente. Ou seja, **A é 100 vezes mais rápido que B.**
- Algoritmo 1: implementado em A por um excelente programador em linguagem de máquina (ultra-rápida). Executa $2n^2$ instruções.
- Algoritmo 2: implementado na máquina B por um programador mediano em linguagem de alto nível dispondo de um compilador "meia-boca". Executa $50n \log_{10} n$ instruções.

Algumas dicas....

T = Tempo de processamento do algoritmo

C= Complexidade do Algoritmo

VC= Velocidade do Computador

T = C/VC

1 M = 10^6

1 G = 10^9

1 T = 10^{12}

Algoritmos e tecnologia

- O que acontece quando ordenamos um vetor de um milhão de elementos? **Qual algoritmo é mais rápido?**
- Algoritmo 1 na máquina *A*:
  $$\frac{2.(10^6)^2 \text{ instruções}}{10^9 \text{ instruções/segundo}} \approx 2000 \text{ segundos}$$
- Algoritmo 2 na máquina *B*:
  $$\frac{50.(10^6 \log 10^6) \text{ instruções}}{10^7 \text{ instruções/segundo}} \approx 100 \text{ segundos}$$
- Ou seja, *B* foi **VINTE VEZES** mais rápido do que *A*!
- Se o vetor tiver 10 milhões de elementos, esta razão será de 2.3 dias para 20 minutos!

Algoritmos e tecnologia

- O uso de um algoritmo em lugar de outro pode levar a ganhos extraordinários de desempenho.
- Isso pode ser tão importante quanto o projeto de *hardware*.
- A melhoria obtida pode ser tão significativa que não poderia ser obtida simplesmente com o avanço da tecnologia.

Algoritmos e outras tecnologias

O exemplo anterior mostra que os algoritmos, como o hardware de computadores, devem ser considerados como uma ***tecnologia***.
O desempenho total do sistema depende da escolha de algoritmos eficientes tanto quanto da escolha de hardware rápido. Os rápidos avanços que estão ocorrendo em outras tecnologias computacionais também estão sendo observados em algoritmos.

Você poderia questionar se os algoritmos são verdadeiramente tão importantes para os computadores contemporâneos levando em consideração outras tecnologias avançadas, como:

Arquiteturas computacionais e tecnologias de fabricação avançadas;
Interfaces gráficas de usuário (GUIs) intuitivas e fáceis de usar;
Sistemas orientados a objetos;
Tecnologias integradas da Web e
Redes de alta velocidade, com fio e sem fio.

A resposta é: **sim.**

Além disso, até mesmo uma aplicação que não exija conteúdo algorítmico no nível da aplicação depende muito de algoritmos.

A aplicação depende de hardware rápido? O projeto de hardware utilizou algoritmos.
A aplicação depende de interfaces gráficas de usuário? O projeto de qualquer GUI depende de algoritmos.
A aplicação depende de rede? O roteamento em redes depende muito de algoritmos.
A aplicação foi escrita em uma linguagem diferente do código de máquina? Então, ela foi processada por um compilador, um interpretador ou um montador, e todos fazem uso extensivo de algoritmos.

Os algoritmos estão no núcleo da maioria das tecnologias usadas em computadores contemporâneos.

Uma sólida base de conhecimento e técnica de algoritmos é uma das características que separam os **programadores verdadeiramente qualificados** dos novatos.

Com a moderna tecnologia de computação, você pode executar algumas tarefas sem saber muito sobre algoritmos; porém, **com uma boa base em algoritmos, é possível fazer muito, muito mais.**

Algoritmos e tecnologia

E se tivermos os tais problemas $\mathcal{NP}$-difíceis ?

$f(n)$	$n = 20$	$n = 40$	$n = 60$	$n = 80$	$n = 100$
n	$2{,}0\times10^{-11}$seg	$4{,}0\times10^{-11}$seg	$6{,}0\times10^{-11}$seg	$8{,}0\times10^{-11}$seg	$1{,}0\times10^{-10}$seg
n^2	$4{,}0\times10^{-10}$seg	$1{,}6\times10^{-9}$seg	$3{,}6\times10^{-9}$seg	$6{,}4\times10^{-9}$seg	$1{,}0\times10^{-8}$seg
n^3	$8{,}0\times10^{-9}$seg	$6{,}4\times10^{-8}$seg	$2{,}2\times10^{-7}$seg	$5{,}1\times10^{-7}$seg	$1{,}0\times10^{-6}$seg
n^5	$2{,}2\times10^{-6}$seg	$1{,}0\times10^{-4}$seg	$7{,}8\times10^{-4}$seg	$3{,}3\times10^{-3}$seg	$1{,}0\times10^{-2}$seg
2^n	$1{,}0\times10^{-6}$seg	1,0seg	13,3dias	$1{,}3\times10^{5}$séc	$1{,}4\times10^{11}$séc
3^n	$3{,}4\times10^{-3}$seg	140,7dias	$1{,}3\times10^{7}$séc	$1{,}7\times10^{19}$séc	$5{,}9\times10^{28}$séc

Supondo um computador com velocidade de 1 Terahertz (mil vezes mais rápido que um computador de 1 Gigahertz).

Ou seja, no divisor, a velocidade é sempre 10 elevado a 12. Então na primeira coluna, primeira linha é $20/10^{12}$.

Na primeira coluna
Quinta linha
$2^{20}/10^{12}$

Complexidade no desempenho de algoritmos

Um **algoritmo** resolver um **problema** quando ...
Para qualquer entrada, produz uma resposta correta
Tempo e **memória** concedidos são suficientes para sua execução

Em casos práticos
Um algoritmo resolver um problema **não significa sua aceitabilidade**
Recursos de espaço e tempo requeridos têm grande importância

Complexidade no desempenho de algoritmos

Exemplo: Solução de sistemas de equações lineares

- O algoritmo mais imediato está longe de ser razoável em termos de eficiência
- Tamanho do problema $\times$ tempo de execução

n	Método de Cramer	Método de Gauss
2	22 μs	50 μs
3	102 μs	150 μs
4	456 μs	353 μs
5	2,35 ms	666 μs
10	1,19 min	4,95 ms
20	**15225 séculos**	**38,63 ms**
40	**5×10^{33} séculos**	**0,315 s**

$\rightarrow$ Os dois algoritmos calculam o determinante de uma matriz $n \times n$

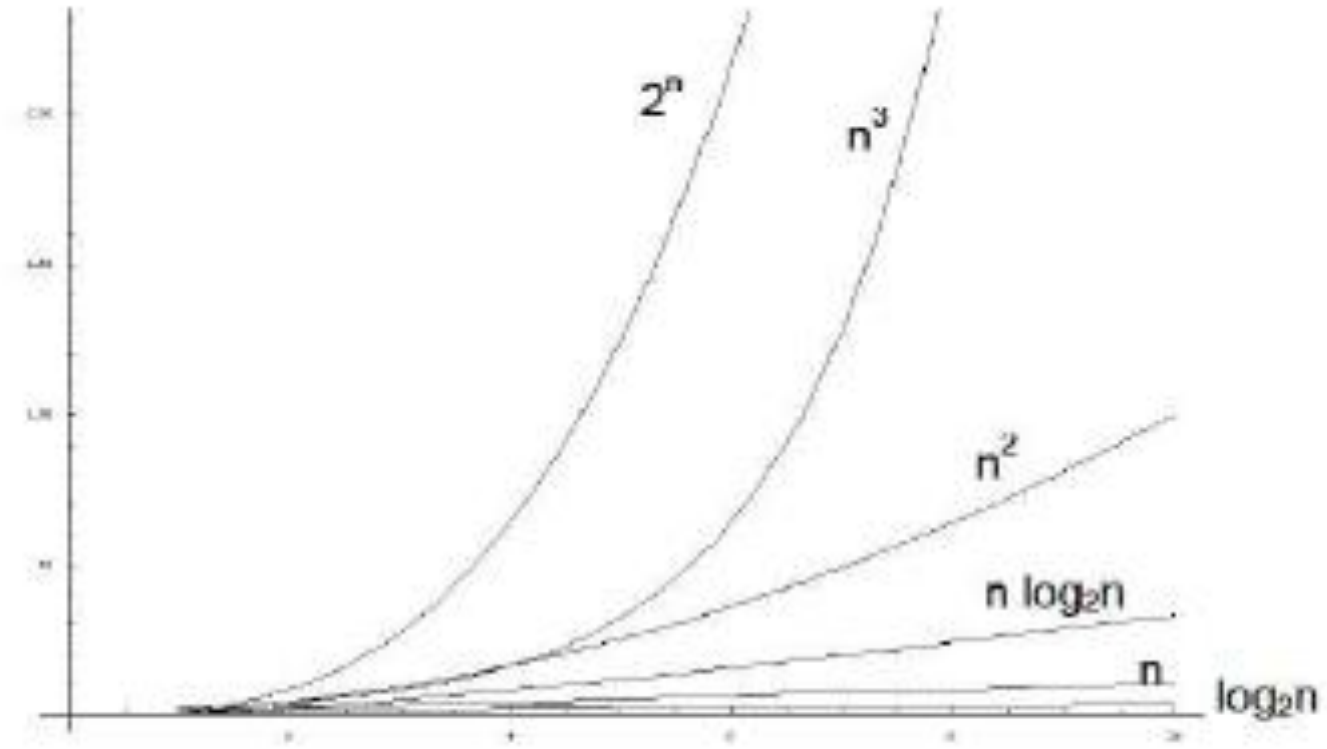
Complexidade no desempenho de algoritmos

→ **Se tivermos uma máquina mais rápida, a importância da complexidade de tempo de um algoritmo deixa de existir?**

Veremos que:
Para uma **algoritmo rápido**, qualquer melhoria na velocidade de execução das operações básicas(**máquina mais rápida**) é sentida, e o conjunto de problemas que podem ser resolvidas por ele aumenta sensivelmente.
Esse impacto é bem menor no caso de **algoritmos** menos eficientes (**mais lentos**).

Para ilustrar vamos considerar o impacto do uso de máquinas mais rápidas em algoritmos com crescimento no tempo de execução:
Linear
Quadrático
Exponencial

Complexidade no desempenho de algoritmos



Crescimento de funções

Complexidade no desempenho de algoritmos

Linear

O tempo de execução é proporcional ao tamanho da entrada

Durante o mesmo tempo t:

Computador com velocidade v resolve problemas de tamanho máximo x_1

Computador com velocidade 10v resolve problemas de tamanho máximo $10X_1$

Quadrático

Agora o tempo de execução é proporcional a n^2 para uma entrada de tamanho n.

Durante o mesmo tempo t:

Computador com velocidade v resolve problemas de tamanho máximo X_3

Computador com velocidade 10v resolve problemas de tamanho máximo $3{,}16X_3$

Exponencial

Agora o tempo de execução é proporcional a 2^n para uma entrada de tamanho n.

Durante o mesmo tempo t:

Computador com velocidade v resolve problemas de tamanho máximo X_5

Computador com velocidade 10v resolve problemas de tamanho máximo $3{,}3 + X_5$

Complexidade no desempenho de algoritmos

Resposta a pergunta anterior...

- As máquinas, tornando-se mais rápidas, passam a poder resolver problemas maiores, e é a complexidade do algoritmo que determina o novo tamanho máximo de problema que pode ser resolvido

$\rightarrow$ Complexidade do algoritmo $\times$ tamanho máximo do problema resolvível

Complex.	**Tam. máx. prob. máq. lenta**	**Tam. máx. prob. máq. rápida**
$\log_2 n$	x_0	x_0^{10}
n	x_1	$10x_1$
$n \log_2 n$	x_2	$10x_2$ (para x_2 grande)
n^2	x_3	$3,16x_3$
n^3	x_4	$2,15x_4$
2^n	x_5	$x_5 + 3,3$
3^n	x_6	$x_6 + 2,096$

Exercícios

1. Considere dois algoritmos A e B com complexidades $8n^2$ e n^3, respectivamente, qual é o mais eficiente ? Qual é o maior valor de n, para o qual o algoritmo B é mais eficiente que o algoritmo A ?

2. Um algoritmo tem complexidade $2n^2$. Num certo computador **A**, em um tempo *t*, o algoritmo resolve um problema de tamanho 25. Imagine agora que você tem disponível um computador **B** 100 vezes mais rápido. Qual é o tamanho máximo do problema que o mesmo algoritmo resolve no mesmo tempo *t*?

3. Considere o exercício anterior para um algoritmo de complexidade 2^n , no qual o computador B é 128 vezes mais rápido.

3. Do livro texto clrs, 1.2-1, 1.2-2, 1.2-3.

Resolução EX1.

n	$8n^2$	n^3
1	8	1
2	32	8
3	72	27
4	128	64
5	200	125
6	228	216
7	392	343
8	512	512
9	648	792

Os algoritmos têm igual desempenho quando **n=8**.
Até **n=7**, B é mais eficiente que A.

Resolução EX2

$C = 2\ n^2$

No computador A , em um tempo T resolve para n=25

Em B, no qual Vb = 100 Va, no mesmo tempo T, quando será n= ?

$T = C/Vc$

$Ta = Tb$

$2x25^2 / Va = 2xn^2 /100Va$

$n^2 = 100x25^2$

n = 250. Ou seja, o tamanho em B é 250.