PANC: Projeto e Análise de Algoritmos

Aula 02: Introdução a Projeto e Análise de Algoritmos

Breno Lisi Romano

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista

Sumário

- Introdução
- Algoritmo
  - Definição
  - Exemplos
- Dificuldades Intrínsecas de Problemas
- Algoritmo vs. Problema vs. Modelo Computacional
- Análise de Algoritmos
  - Perspectivas
  - Estudo Prático Inicial

Introdução

- O que veremos nesta disciplina?
  - Como provar a "**Corretude**" de um algoritmo
  - Estimar a **quantidade de recursos** (tempo, memória) de um algoritmo = **Análise de Complexidade**
  - Técnicas e ideias gerais de projeto de **algoritmos**: divisão e conquista, programação dinâmica, algoritmos gulosos, etc.
  - Tema recorrente: natureza **recursiva** de vários problemas
  - A **dificuldade intrínseca** de vários **problemas**: inexistência de soluções eficientes

A importância dos Algoritmos

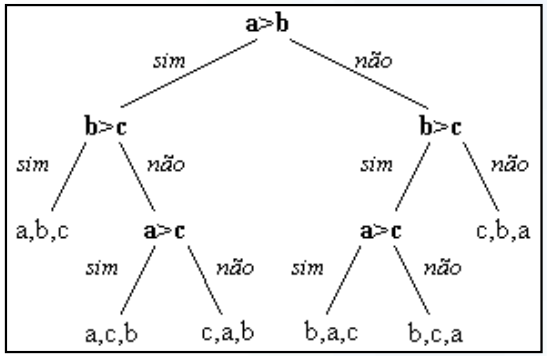
- **Exemplos de aplicações para o uso/desenvolvimento de algoritmos "eficientes"?**
  - projetos de genoma de seres vivos
  - rede mundial de computadores
  - comércio eletrônico
  - planejamento da produção de indústrias
  - logística de distribuição
  - computação científica
  - imagens médicas
  -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

Algoritmo (1)

- Definição?
- **Definição Informal 01:**
  - É qualquer **procedimento computacional** bem definido que toma algum valor ou conjunto de valores como **entrada** e **produz** algum valor ou conjunto de valores como **saída**.
  - É uma **sequência de passos computacionais que transformam uma entrada na saída**
- **Definição Informal 02:**
  - Uma **ferramenta** para **resolver** um **problema** computacional bem especificado
  - O enunciado do **problema** computacional **especifica**, em termos gerais, o relacionamento entre **entrada** e **saída**
  - O **algoritmo** descreve um **procedimento computacional** para alcançar esse relacionamento da entrada com a saída

Exemplo 01: Ordenação de 03 Valores

- **Problema de Ordenação:**
  - Ordenar 03 números inteiros de maneira decrescente
- **Entrada:**
  - a, b, c
- **Saída:**
  - a > b > c | a > c > b | b > a > c | b > c > a | c > a > b | c > b > a
- **Instância de um Problema:**
  - Uma instância de um problema consiste em uma entrada específica para qual se deseja calcular uma solução para o problema, por exemplo:
    - Entrada: 30, 11, 25
    - Saída: 30, 25, 11




```
Função Ordenar03Valores()
{
    Ler a, b, c
    if (a > b)
        if (b > c) imprimir (a > b > c)
        else
            if (a > c) imprimir (a > c > b)
            else imprimir (c > a > b)
    else
        if (b > c)
            if (a > c) imprimir(b > a > c)
            else imprimir(b > c > a)
        else imprimir(c > b > a)
}
```

Exemplo 02: Ordenação

- **Problema de Ordenação:**
  - Ordenar uma sequência de números de maneira crescente
- **Entrada:**
  - Uma sequência de n números $<a_1, a_2, \ldots, a_n>$
- **Saída:**
  - Uma permutação $<a'_1,a'_2,\ldots,a'_n>$ da sequência de entrada, tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- **Instância de um Problema:**
  - Entrada: <31, 41, 59, 25, 41>
  - Saída: <25 , 31, 41, 41, 59>

Exemplo 03: Primalidade

- **Problema da Primalidade:**
  - Determinar se um dado número inteiro é primo
- **Entrada:**
  - Número Inteiro
- **Saída:**
  - Decisão: É primo ou Não é primo!
- **Instância de um Problema:**
  - Entrada: 9411461 / Saída: É primo
  - Entrada: 8411461 / Saída: Não é primo

Algoritmo (2)

- **Definição Formal 01:**
  - Conjunto das **regras** e **procedimentos lógicos** perfeitamente **definidos** que levam à **solução de um problema** em um número de etapas (Dicionário Houaiss da Língua Portuguesa, 2001, 1a edição)
- **Definição Formal 02:**
  - *The term algorithm is used in computer science to **describe** a **problem-solving method** suitable for **implementation** as a **computer program** (Algorithms in C, Sedgewick, 1998, 3rd edition)*
- **Observações:**
  1. O número de **etapas** deve ser **finito**: se um algoritmo levar décadas, séculos ou milênios para executar, ele é impraticável
  2. Existem **diferentes modelos computacionais** nos quais os algoritmos podem ser implementados

Algoritmo (3)

- Podemos **descrever** um **algoritmo** de diferentes maneiras:
  - implementando-o em **linguagem de máquina** diretamente executável em hardware
  - em **português**
  - usando uma linguagem de **programação de alto nível**: C, Pascal, Java, Python, etc
  - em um **pseudocódigo** de alto nível → Livro do Cormen
- Nesta disciplina, usaremos essencialmente as **duas últimas alternativas**

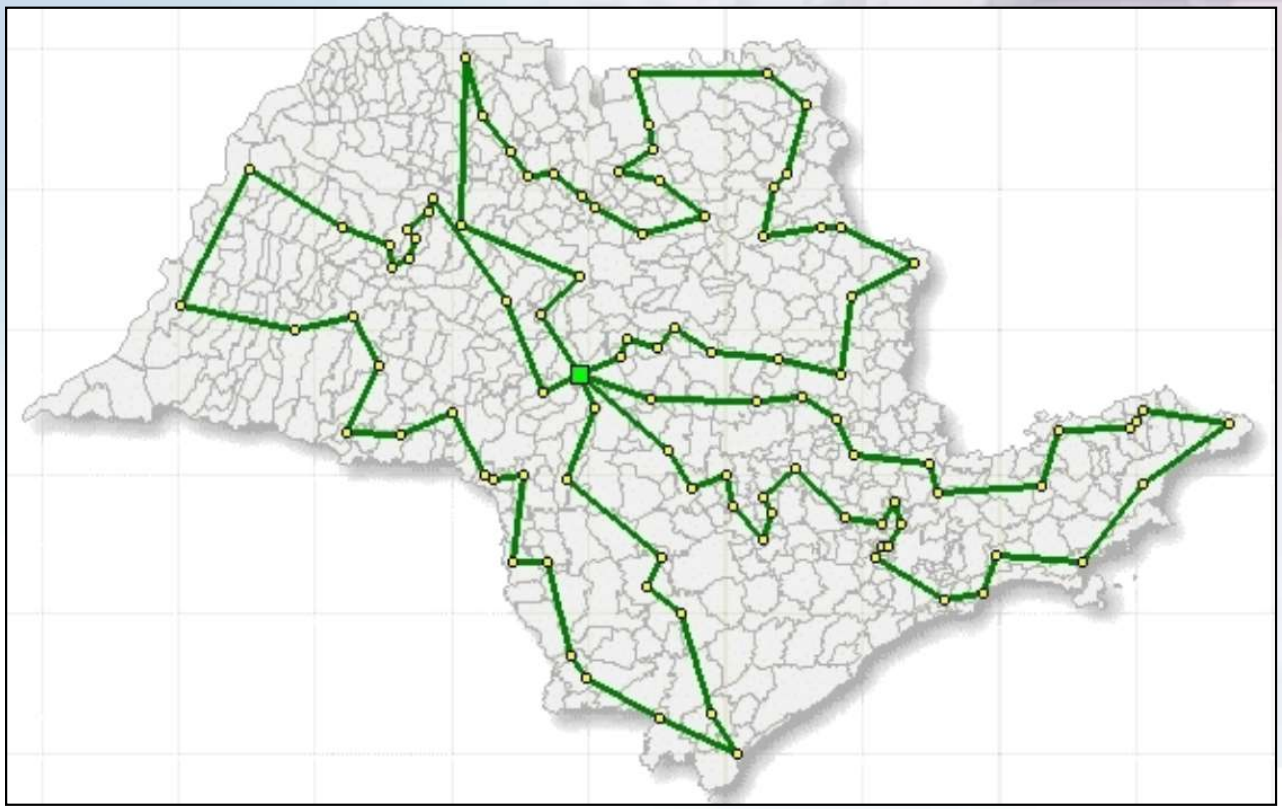
Dificuldade Intrínseca de Problemas (1)

- Infelizmente, existem certos **problemas** para os quais **não se conhece algoritmos eficientes** capazes de resolvê-los → Problemas 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 se preocupar com esses problemas?**
  - Problemas dessa classe surgem em inúmeras situações práticas

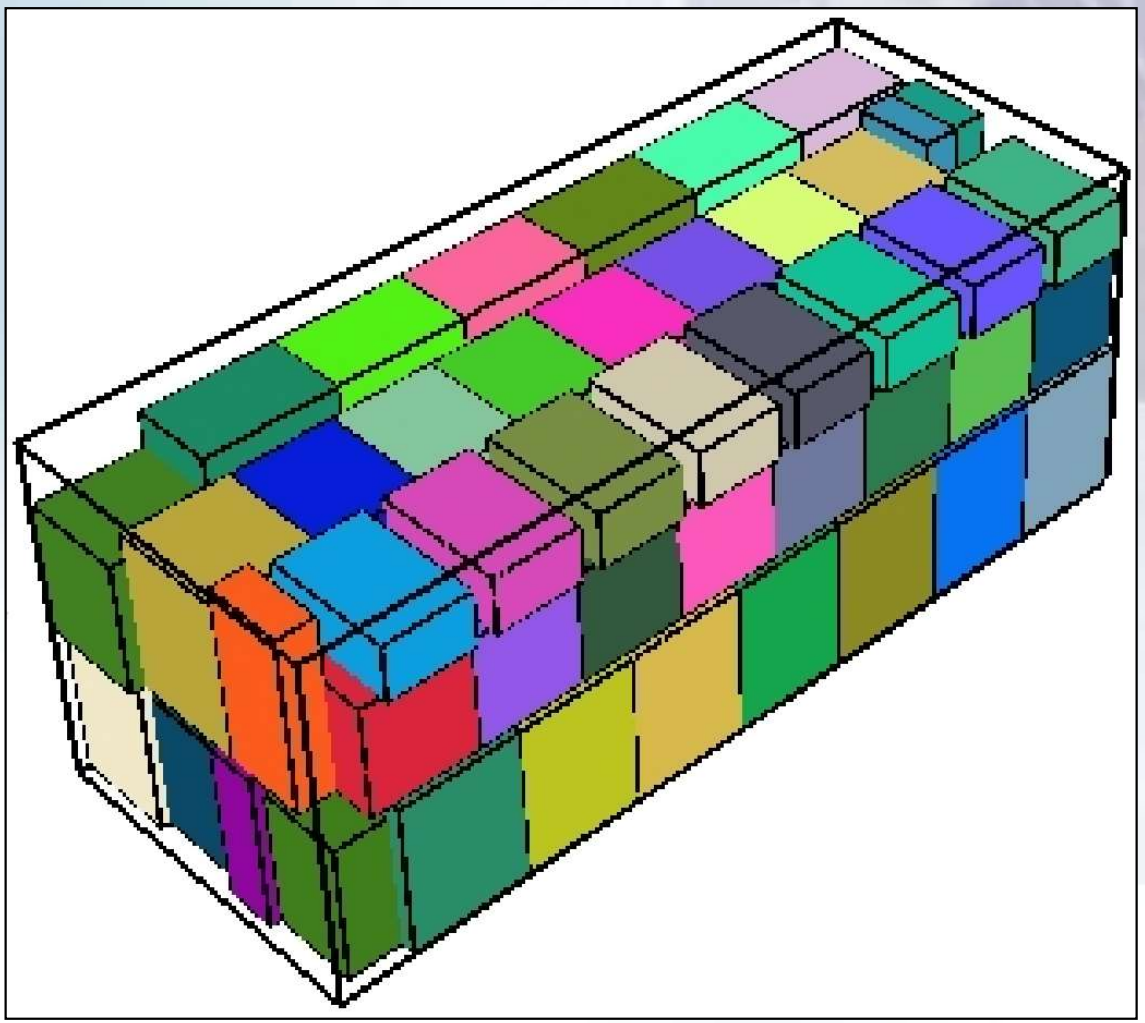
Dificuldade Intrínseca de Problemas (2)

- ***Vehicle Routing***: Calcular as rotas dos caminhões de entrega de uma distribuidora de bebidas no estado de São Paulo, minimizando a distância percorrida



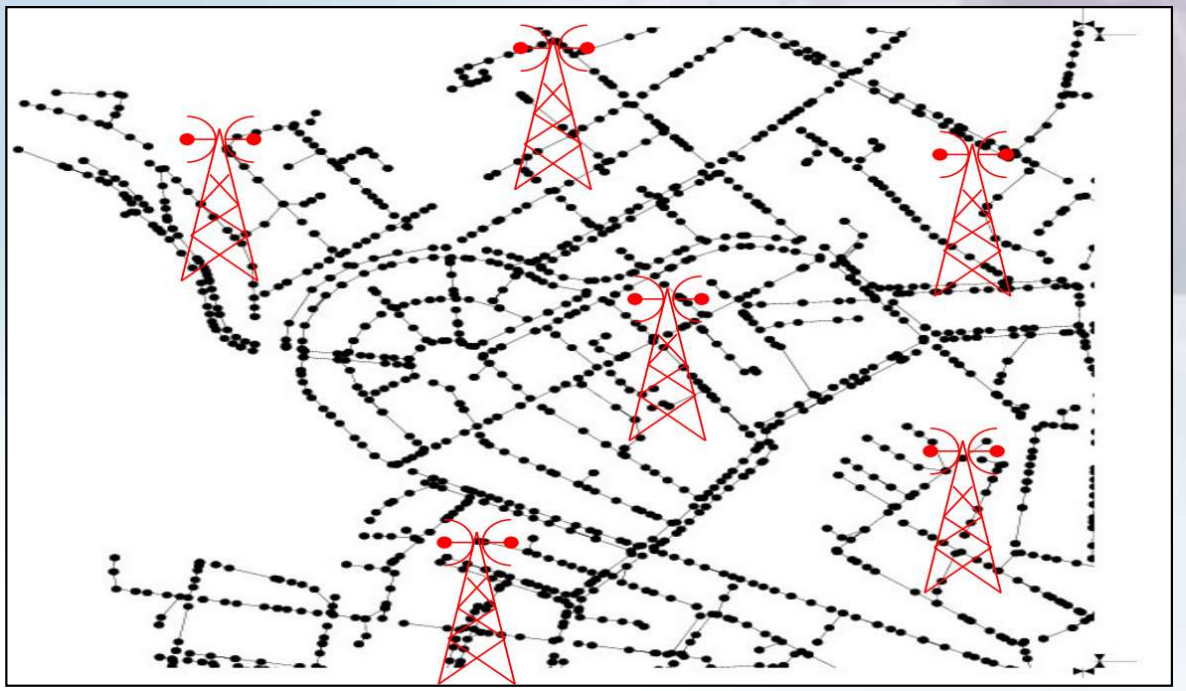
Dificuldade Intrínseca de Problemas (3)

- ***Bin Packing 3D***: calcular o número mínimo de containers para transportar um conjunto de caixas com produtos



Dificuldade Intrínseca de Problemas (4)

- ***Facility Location***: calcular a localização e o número mínimo de antenas de celulares para garantir a cobertura de uma certa região geográfica



- É importante saber identificar quando estamos lidando com um problema desta natureza!!!

Algoritmo vs. Problema vs. Modelo Computacional

- Um programa pode ser entendido como um **algoritmo implementado** em uma **determinada linguagem** para solucionar um **problema computacional** específico em um **modelo computacional** em particular.
- Observações importantes:
  - Nem todo problema computacional é solucionável
  - Diferentes problemas possuem diferentes níveis de dificuldade
  - Diferentes algoritmos possuem diferentes níveis de complexidade
  - Dependendo do modelo computacional utilizado, pode não haver algoritmo possível para determinado problema
  - Cada instrução executada em um modelo computacional possui um **custo de tempo**
  - Cada dado armazenado em um modelo computacional possui um **custo de espaço**

Modelo Computacional (1)

- **Condição ideal (irreal):** os computadores têm velocidade de processamento e memória infinita → Qualquer algoritmo é igualmente bom e esta disciplina é inútil
- **O mundo real:** há computadores com velocidade de processamento na ordem de bilhões de instruções por segundo e trilhões de bytes em memória
  - Mas ainda assim existe uma limitação na velocidade de processamento e memória dos computadores
- Neste caso, faz muita diferença ter um **bom algoritmo (eficiente*)**

*** virtude ou característica de (alguém ou algo) ser competente, produtivo, de conseguir o melhor rendimento com o mínimo de erros**

Modelo Computacional (2)

- Vamos pensar sobre o Modelo Computacional para **Ordenação** de **Arrays** com **n elementos:**
  - **Computador A**: Executa 1G (10^9) instruções/segundo
  - **Computador B**: Executa 10M (10^7) instruções/segundo
  - Ou seja, A é **100 vezes** mais rápido que B
- **Algoritmo 01:**
  - Implementado em A
  - Excelente Programador em Linguagem de Máquina
  - Executa $2.n^2$ instruções
- **Algoritmo 02:**
  - Implementado em B
  - Programador Iniciante em Linguagem de Alto Nível
  - Executa 50 n. log.n instruções

Modelo Computacional (3)

- O que acontece quando ordenamos um array de um milhão de elementos (10^6)? **Qual algoritmo é mais rápido?**

- **Algoritmo 1 na Máquina A:**

$$E_a = \frac{2.(10^6)^2\ instruções}{(10^9)\ instruções\ por\ segundo} \rightarrow \text{2000 segundos}$$

- **Algoritmo 2 na Máquina B:**

$$E_b = \frac{50.(10^6 log 10^6)\ instruções}{(10^7)\ instruções\ por\ segundo} \rightarrow \text{30 segundos}$$

- Ou seja, B foi **66,67 VEZES** mais rápido do que A
  - Se o array tiver 10 milhões de elementos, esta razão será de 2.3 dias para aproximadamente 07 minutos!

Modelo Computacional (4)

- O uso de um **algoritmo adequado** pode levar a ganhos extraordinários de desempenho
- Isso pode ser tão importante quanto a arquitetura de hardware (**modelo computacional**)
- A melhora obtida pode ser tão significativa que não poderia ser obtida simplesmente com o avanço da tecnologia
  - As melhorias nos algoritmos produzem avanços em componentes básicos das aplicações
  - Mas para analisarmos os algoritmos, precisamos pensar independente do modelo computacional
- **Conclusão:** Precisamos padronizar o Modelo Computacional

Modelo Computacional Genérico - Arquitetura de von Neumann

- Um único processador (ou unidade lógico aritmética)
- Memória RAM
- Operações sequenciais, não há paralelismo
- Instruções:
  - Aritméticas (soma, subtração, multiplicação, divisão, resto, piso e teto)
  - Movimentação de dados (carregar, armazenar e copiar)
  - Controle (desvio condicional e incondicional, chamada e retorno de rotinas)
- Cada instrução tem tempo de execução constante, embora possam ser diferentes
- Com base nas operações definidas, outras operações mais complexas podem ser derivadas
- Algumas áreas “cinzas” como exponenciação são tratadas como tempo constante também

Análise de Algoritmo (1)

- "*Algorithm analysis usually means 'give a big-O figure for the running time of an algorithm' (Of course, a big-Θ would be even better). This can be done by getting a big-O figure for parts of the algorithm and then combining these figures using the sum and product rules for big-O. Another useful technique is to pick an elementary operation, such as additions, multiplications or comparisons, and observing that the running time of the algorithm is big-O of the number of elementary operations. Then, you can analyze the exact number of operations as function of n in the worst case. This is easier to deal with because it is an exact function of n and you don't have the messy big-O symbols to carry through your analysis*"
- **- Ian Parberry, *Problems on Algorithms***

Análise de Algoritmo (2)

- **Definição:**
  - Analisar um algoritmo significa prever os recursos que ele necessitará para sua execução. Os mais importantes são tempo e espaço
- **Tempo de Execução:**  **Estudaremos esta questão!**
  - É o número de instruções primitivas executadas pelo algoritmo
- **Espaço:**
  - É de fato o espaço necessário para armazenar dados durante a execução do algoritmo
- **Utilidade:**
  - Com base na análise, pode-se identificar a **eficiência** de cada **algoritmo** para um determinado problema
  - A análise é realizada levando em consideração um determinado modelo computacional

Perspectivas

- **Definição:**
  - Além do ambiente computacional, o comportamento de um algoritmo pode variar de acordo com o **comportamento da entrada (tamanho, estrutura, etc.)**, o que gera diferentes perspectivas
- **Melhor Caso:**
  - A entrada está organizada de maneira que o algoritmo levará o tempo mínimo para resolver o problema
- **Pior Caso:**
  - A entrada está organizada de maneira que o algoritmo levará o tempo máximo para resolver o problema
- **Caso Médio:**
  - A entrada está organizada de maneira que o algoritmo levará um tempo médio para resolver o problema

Análise e Perspectivas

- As **análises** se concentram geralmente no pior caso e no caso médio:
  - O **pior caso** nos dá uma ideia de quão ruim pode ser o comportamento do algoritmo - cálculo razoavelmente simples, nos dá uma garantia de que o algoritmo não poderá ser mais lento. Pode ser crucial para aplicações críticas.
  - O **caso médio** nos dá uma ideia de como o algoritmo se comportará em boa parte dos casos: determinar qual é o comportamento médio pode ser mais complexo, envolvendo probabilidades

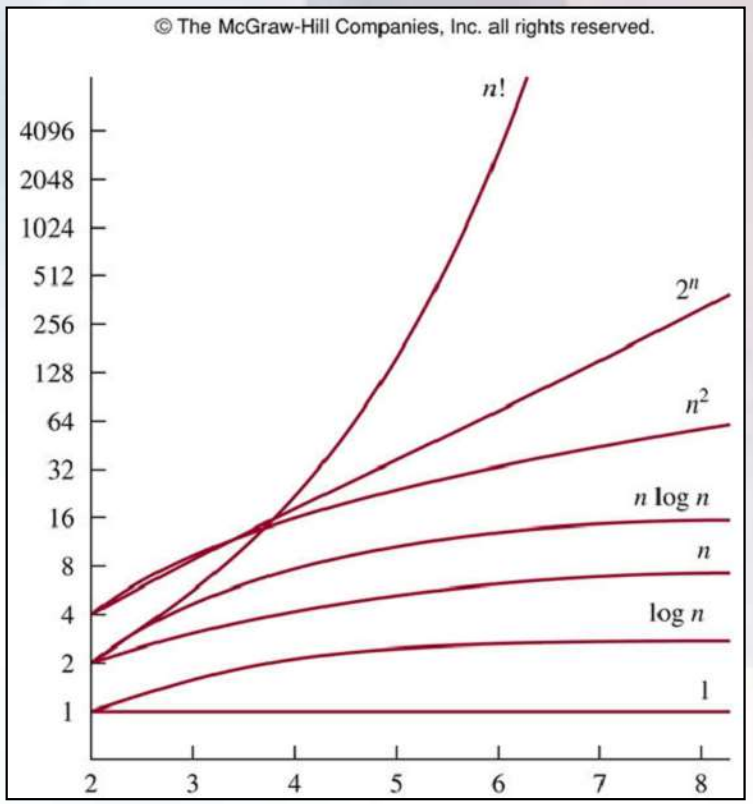
Funções Tipicamente Utilizadas

- **Representação**:
  - O **comportamento do algoritmo** é expressado como uma **função matemática** definida sobre o **tamanho** da **entrada**, denotada por T(n)
    - Por exemplo, ordenar 3 números é mais rápido que ordenar 1000 usando o mesmo algoritmo, porém, ambos seguem uma mesma função de crescimento do tempo
  - Geralmente, o **tempo de execução aumenta** com o aumento da **entrada**
- **Análise Assintótica**:
  - A análise deve ser simplificada, e para a expressão da complexidade em cada perspectiva utilizamos a análise assintótica
  - Estamos interessados mais no **comportamento**, na **taxa de crescimento do tempo de execução** do que de fato na precisão da função utilizada para expressar a complexidade

Medida de Complexidade e Eficiência de Algoritmos

- Um **algoritmo** é chamado **eficiente** se a função que mede sua **complexidade** de tempo e limitada por um **polinômio** no **tamanho** da **entrada**
  - Por exemplo: n, $3n - 7$, $4n^2$, $143n^2 - 4n + 2$, n^5

- Mas por que polinômios?
  - Resposta: **polinômios** são funções bem **comportadas**!!!



Estudo Prático: Análise Inicial da Complexidade – Discussão sobre o Tempo de Execução (1)

- **Problema**:
  - Dado um array A de n números inteiros, determine o maior valor entre eles.
- **Tópicos da Análise**:
  1. Quais operações são relevantes?
  2. Quais são os limitantes superior e inferior para este problema?
  3. Melhor Caso? Pior Caso? Caso Médio?
- Como **calcular** "grosseiramente", em função de n, o **número máximo** de **operações** realizadas por este algoritmo?

```
1 int arrayMax(int A[ ], int n)
2 {
3     int currentMax = A[0];
4     for(int i=1; i<n; i++){
5         if(A[i] > currentMax)
6             currentMax = A[i];
7     }
8     return currentMax;
9 }
```

Estudo Prático: Análise Inicial da Complexidade – Discussão sobre o Tempo de Execução (2)

- **Tópicos da Análise**:
  1. Quais operações são relevantes?
  2. Quais são os limitantes superior e inferior para este problema?
  3. Melhor Caso? Pior Caso? Caso Médio?
- Como **calcular** "grosseiramente", em função de n, o **número máximo** de **operações** realizadas por este algoritmo?

```
1 int arrayMax(int A[ ], int n)
2 {
3     int currentMax = A[0];
4     for(int i=1; i<n; i++){
5         if(A[i] > currentMax)
6             currentMax = A[i];
7     }
8     return currentMax;
9 }
```

Conclusões

- Na **análise de complexidade** de um **algoritmo** estamos mais interessados no seu **comportamento geral** do que em outros detalhes que podem depender da máquina, do sistema operacional, da linguagem, dos compiladores, etc...

- Procura-se **medir a complexidade** de um algoritmo em função de um **parâmetro do problema**, geralmente, o **tamanho da entrada**

- Alguma **operação** (ou conjunto de operações) devem **balizar a análise de complexidade** de um algoritmo

PANC: Projeto e Análise de Algoritmos

Aula 02: Introdução a Projeto e Análise de Algoritmos

Breno Lisi Romano

Dúvidas???

http://sites.google.com/site/blromano

Instituto Federal de São Paulo – IFSP São João da Boa Vista
Bacharelado em Ciência da Computação – 3º Semestre

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus São João da Boa Vista