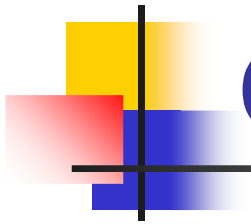
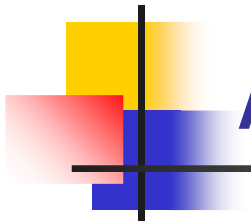
Conceitos Básicos

Universidade Federal do Amazonas
Departamento de Eletrônica e Computação



Objetivos

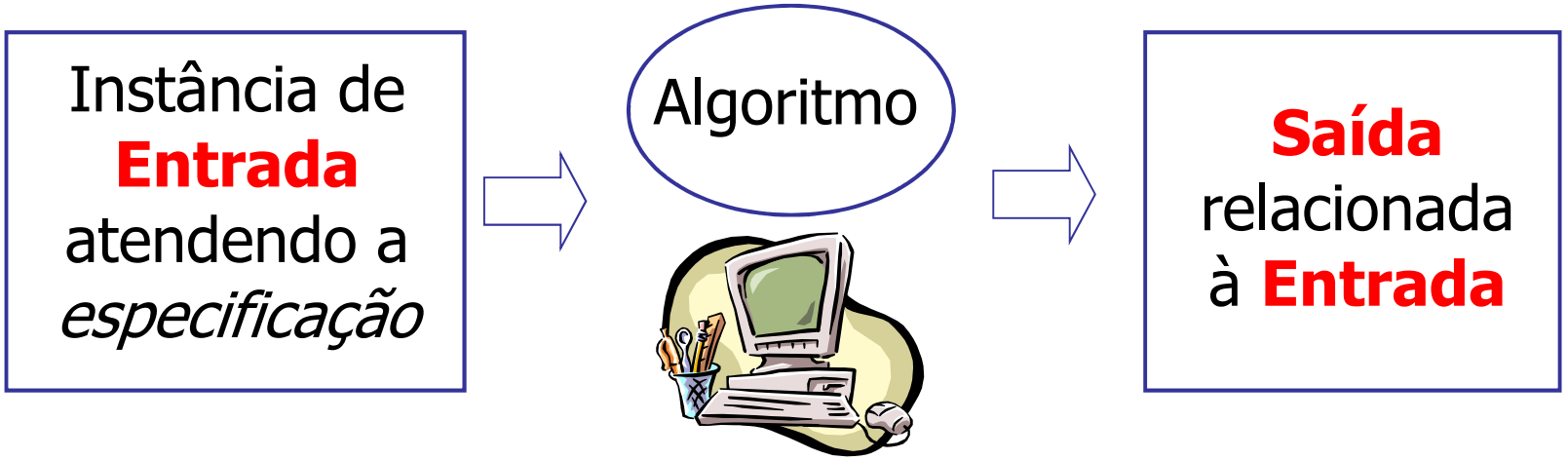
- Definir algoritmos, programas e estruturas de dados
- Analisar o tempo de execução e espaço de memória usado por um algoritmo
- Determinar loop invariante para provar propriedades em loops
- Aplicar as técnicas em algoritmos clássicos usados em ordenação e sistemas discretos
  - método da inserção, bolha e busca binária
  - filtros e controladores digitais



Algoritmos

- **Um algoritmo**: A essência de um procedimento computacional composto por instruções seqüenciais passo a passo
  - procedimento computacional bem definido (valores de entrada e saída – ordenação de números)
- **Um programa**: Implementação de um algoritmo em uma dada linguagem de programação
- **Estrutura de dados**: forma de organizar os dados necessários à solução de um problema

Solução Algorítmica



- O Algoritmo descreve ações sobre a instância de entrada
- Existem infinitos algoritmos corretos para um mesmo problema algorítmico

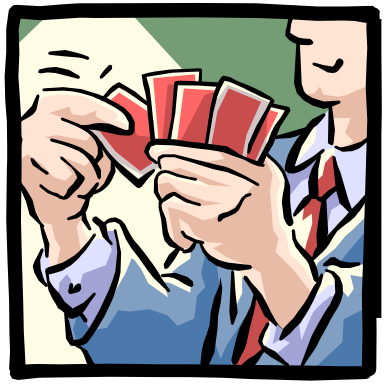
Definição Formal: Ordenação

ENTRADA
sequência de números

$a_1, a_2, a_3, \ldots, a_n$

2 5 4 10 7

(instância)



SAÍDA
uma permutação (reordenação)
da sequência de entradas

$b_1, b_2, b_3, \ldots, b_n$

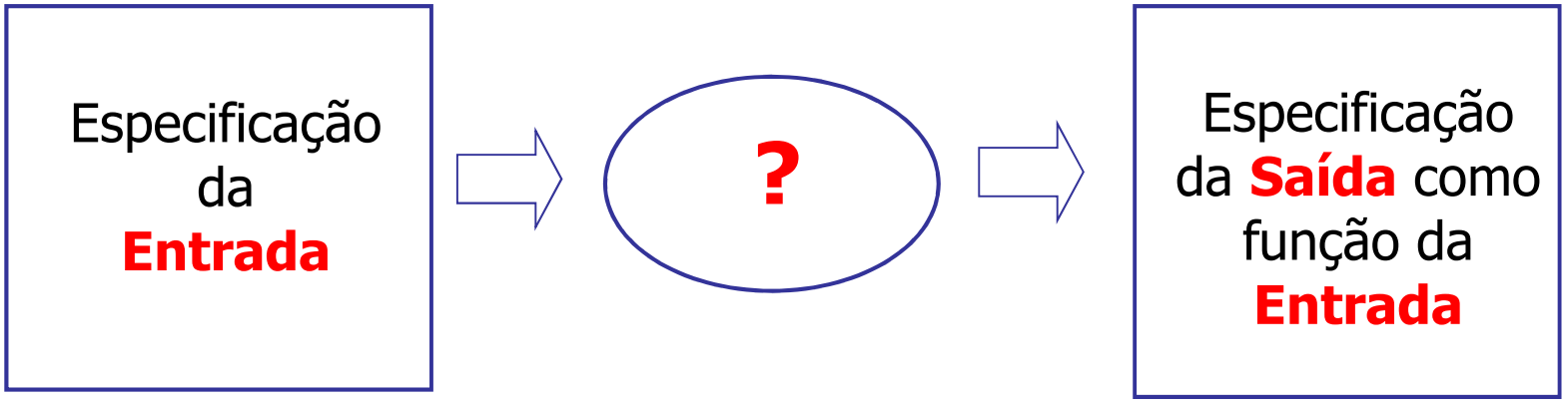
2 4 5 7 10

Corretude
Para qualquer entrada dada, o algortimo termina com saída

- $b_1, b_2, b_3, \ldots, b_n$, onde
- $b_1 < b_2 < b_3 < \ldots < b_n$

Tempo de Execução
Depende de:
- número de elementos (n)
- o quão (parcialmente) ordenada está a lista
- solução algoritmica
- disp. de armazenamento

Problema Algorítmico

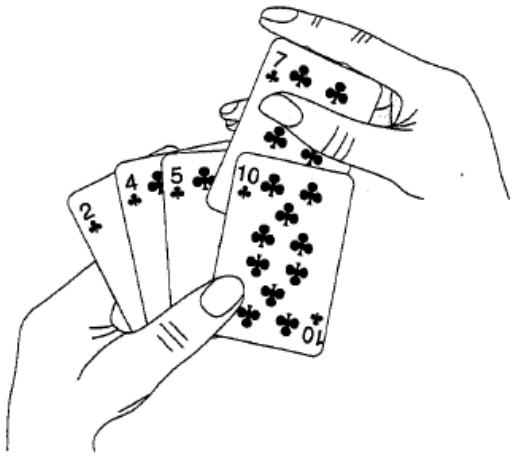


- Existe um número infinito de *instâncias* da entrada que satisfazem a especificação. Por exemplo:
  - Uma seqüência finita, ordenada, não decrescente de números naturais:
    - 1, 20, 908, 909, 100000, 1000000000
    - 3, 15, 105, 876, 1000, 100000

Primeiro Algoritmo

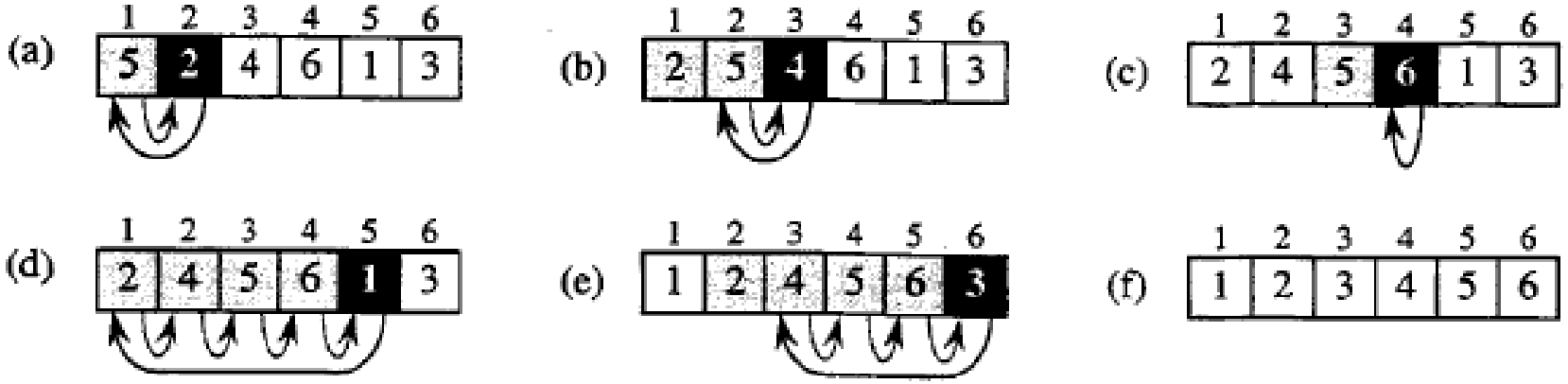
- O nosso primeiro algoritmo, o de ordenação por inserção, resolve o problema de ordenação
  - Entrada: $\langle a_1, a_2, \ldots a_n \rangle$
  - Saída: $\langle a'_1, a'_2, \ldots a'_n \rangle$ tal que $a'_1 \leq a'_2 \leq \ldots \leq a'_n$
- Procdimento:

1. Mão esquerda vazia e cartas p/ baixo
2. Remove uma carta inser. na pos. correta
3. Compara as cartas da direita p/ esquerda

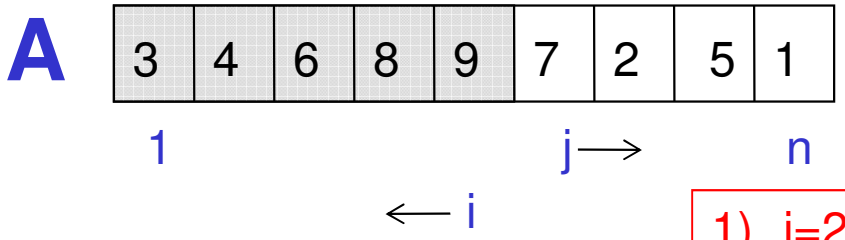


Ordenação por Inserção (1)

- Entrada: *<5,2,4,6,1,3>*
- Saída: *<1,2,3,4,5,6>*



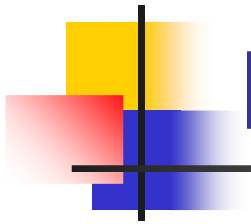
Ordenação por Inserção (2)



"insira A[j] na sequência ordenada A[1..j-1]"

1) j=2; pivot=4; i=1; A[2]=4
2) j=3; pivot=6; i=2; A[3]=6
3) j=4; pivot=8; i=3; A[4]=8
4) j=5; pivot=9; i=4; A[5]=9

```
1 para j=2 até n
2   faça pivot:=A[j]
3      i ← j-1
4      enquanto i>0 e A[i]>pivot faça
5         A[i+1] ← A[i]
6         i--;
7      fim-enquanto
8      A[i+1] ← pivot;
```

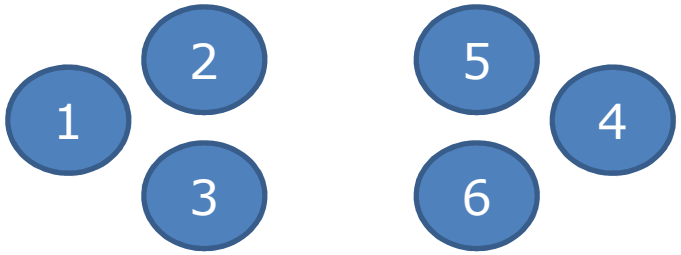


Exercício (1)

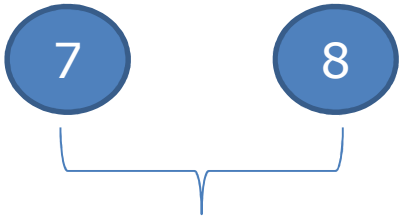
1) Você tem 8 moedas com o mesmo peso, exceto uma que é levemente mais pesada do que as outras. Você tem uma balança antiga que permite pesar pilhas de moedas para checar qual é mais pesada (ou se tem o mesmo peso). Qual é o **número mínimo de pesagens** que você pode fazer para saber qual moeda é mais pesada?

2) Dado um vetor ordenado, como você encontra a localização de inteiro **x**?

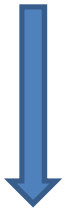
Solução do Exercício 1



Se os dois grupos tiverem o mesmo peso

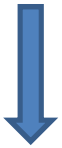


Uma das duas é a mais pesada



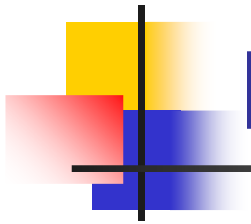
Se um grupo for mais pesado do que o outro

Se uma das moedas for mais pesada, encontramos a solução



Se as duas moedas tiverem o mesmo peso

.....



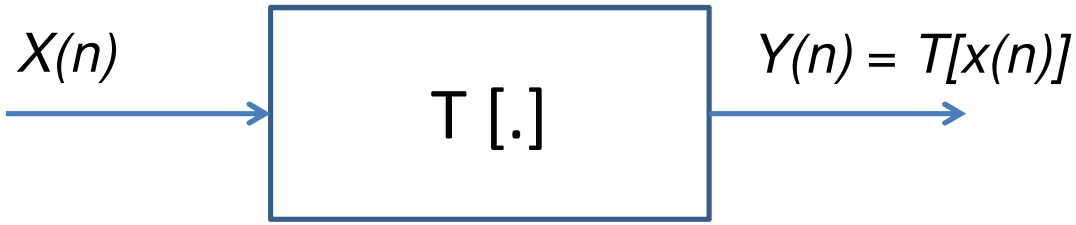
Exercício (2)

3) Implemente o algoritmo da ordenação por inserção usando a linguagem C ou C++. Analise o algoritmo em termos dos loops para os seguintes casos:
   a) Os elementos do vetor já estão ordenados
   b) Os elementos do vetor estão em ordem inversa
   c) Alguns elementos do vetor não estão ordenados

4) Agora considere um vetor com 1K, 10K, 100K e 1M posições e compute o tempo de execução usando **wall** e **cpu time** para cada um dos casos a), b) e c)

Solução Algorítmica em Sistemas Discretos

- Um sistema discreto (SD) é um operador matemático
  - transforma um sinal em um outro sinal
  - usa um conjunto fixo de operações



- Um sinal de entrada *x(n)* é transformado em um sinal de saída *y(n)* através da transformação *T[.]*
- A relação de entrada e saída pode ser expressada em termos de uma função ou regra matemática

Exemplo de Solução Algorítmica em Sistema Discreto

- Considere um sistema discreto modelado pela seguinte eq. de diferença

$$y(n) = -ay(n-1) + x(n) \qquad x(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

substituido a=-1/2 e aplicando a função de impulso unitário

$$y(0) = -(-0.5) \times 0 + 1 = 1$$
$$y(1) = -(-0.5) \times 1 + 0 = 0.5$$
$$y(2) = -(-0.5) \times (-0.5) + 0 = 0.25$$
$$\ldots$$

$$|y(0)| + |y(1)| + \ldots |y(n)| = \frac{1}{1-0.5} = 2$$

```
...
for(i=0; i<n; i++) {
  if (i==0)
    y[0] = x[0];
  else
    y[i]=-a*y[i-1]+x[i];
}
...
```

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tempo linear
  - N é o número de saída no passado
  - M é o número das entradas atuais e do passado

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tempo linear
  - N é o número de saída no passado
  - M é o número das entradas atuais e do passado

y(n) é a saída no instante *n*

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tem
  - N é o número de saíd
  - mero das entra atuais e do passado

y(n-k) é a saída *k* passos no passado

y(n) é a saída no instante *n*

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tem
  - N é o número de saíd
  - mero das entra atuais e do pass

y(n-k) é a saída *k* passos no passado

x(n-k) é a entrada *k* passos no passado

y(n) é a saída no instante *n*

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tem
  - N é o número de saíd
  - mero das entra atuais e do pas

y(n-k) é a saída *k* passos no passado

x(n-k) é a entrada *k* passos no passado

y(n) é a saída no instante *n*

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

a_k são os coeficientes para as saídas

Filtros Digitais

- Filtros digitais podem ser definidos como sistema de tempo discreto no tem
  - N é o número de saíd
  - mero das entra atuais e do pass

y(n-k) é a saída *k* passos no passado

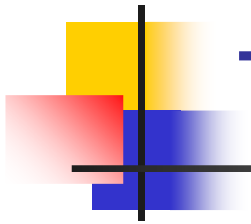
x(n-k) é a entrada *k* passos no passado

y(n) é a saída no instante *n*

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

a_k são os coeficientes para as saídas

b_k são os coeficientes para as entradas

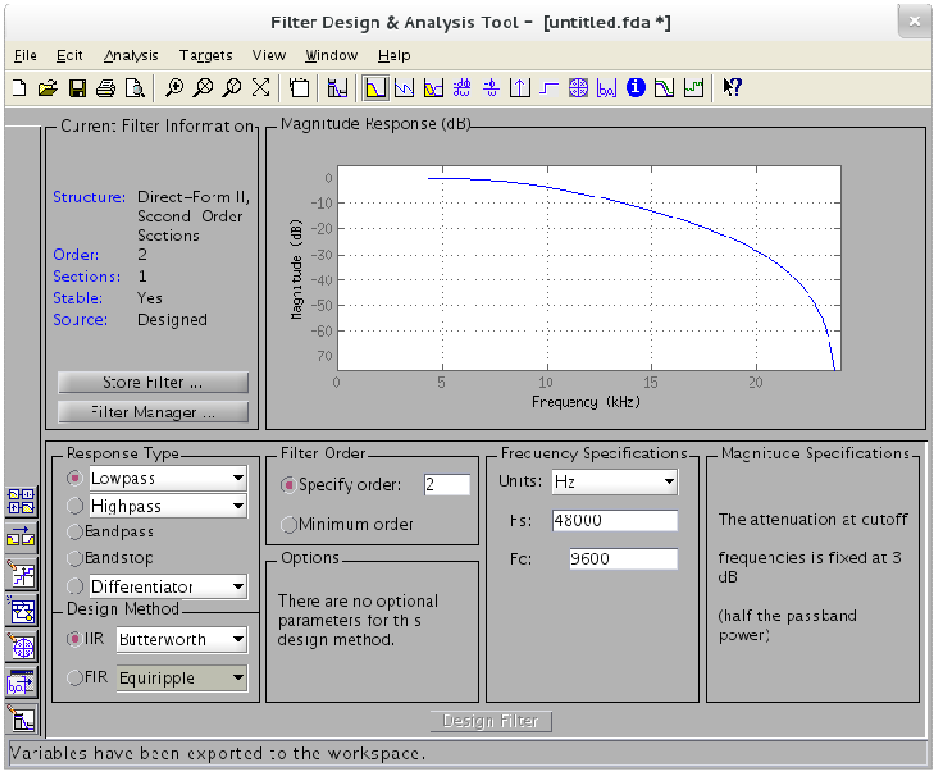


Tipos de Filtros Digitais

- Filtros digitais são geralmente classificados de acordo com as características do domínio da frequência
  - **Passa-baixa**: atenua frequências mais elevadas do que a frequência de corte
  - **Passa-alta**: atenua freqüências mais baixas do que a frequência de corte
  - **Passa-banda**: atenua frequências fora da faixa entre as frequências de corte
  - **Bandstop**: atenua frequências dentro da faixa entre as frequências de corte
  - **Passa-tudo**: não atenua nenhuma frequência, geralmente muda apenas a fase do sinal

Análise e Projeto de Filtros (1)

- **fdatool**: permite projetar filtros digitais no Matlab



- Exportar os coeficientes: file->export, workspace

Análise e Projeto de Filtro (2)

- **[B, A] = sos2tf(SOS,G)**: obtém a função de transferência com os respectivos coeficientes

B =	0.2066	0.4131	0.2066
A =	1.0000	-0.3695	0.1958

- **format long**: aumenta o tamanho do formato dos números
- **fi(B,1,7,5):** 1: bit do sinal; 7: tamanho da palavra; 5: parte fracionária
- **fi(A,1,7,5):** 1: bit do sinal; 7: tamanho da palavra; 5: parte fracionária

Análise e Projeto de Filtro (3)

- Matlab retorna os seguintes coeficientes:

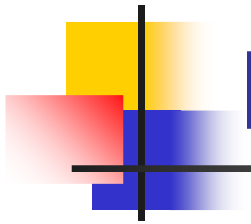
B =	0.21875	0.40625	0.21875
A =	1.0000	-0.375	0.1875

aplicando os coeficientes em:

$$y(n) = -\sum_{k=1}^{N} a_k\, y(n-k) + \sum_{k=0}^{M} b_k\, x(n-k)$$

temos:

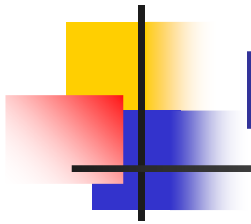
$$y(n) = 0.375\, y(n-1) - 0.1875\, y(n-2) + 0.21875\, x(n)$$
$$+ 0.40625\, x(n-1) + 0.21875\, x(n-2)$$



Exercício (3)

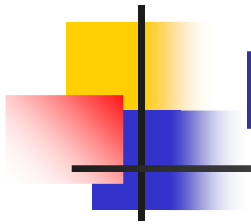
1) Forneça o algoritmo para implementar o seguinte sistema discreto (de um passa baixa IIR segunda ordem) considerando *a=0.4375* e *b=-0.125*

$$y(n) = ay(n-1) – by(n-2) + x(n)$$



Loop Invariante

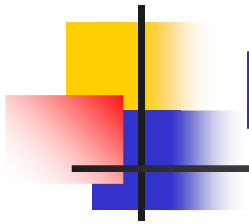
- Loop invariante é uma invariante usada para provar propriedades de loops:
  - nos ajudam a entender porque um algoritmo é correto
- Devemos mostrar três detalhes:
  - **Inicialização:** verdadeiro antes da inicialização do loop
  - **Manutenção:** Se for verdadeiro antes de uma iteração do loop, permanecerá verdadeiro antes da próxima iteração
  - **Término:** Quando o loop termina, a invariante fornece uma propriedade útil para mostrar que o algoritmo é correto



Padrão do Loop Invariante

- O padrão geral para inserir loop invariante no código é dado por:

```
...
  // the Loop Invariant must be true here
  while ( TEST CONDITION ) {
     // top of the loop
     ...
     // bottom of the loop
     // the Loop Invariant must be true here
  }
  // Termination + Loop Invariant = Goal
  ...
```



Exemplo: Loop Invariante

- Considere o seguinte código:

```
#define SIZE 10
int x;
void main() {
  while (x<SIZE)
   x=x+1;
}
```

Qual seria o loop invariante? X<=10

Qual seria a condição de parada? X==10

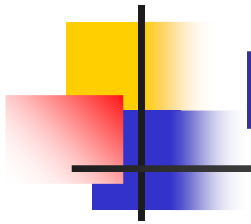
Exemplo: Loop Invariante

- Considere o seguinte código:

```
#define SIZE 10
int x;
void main() {
  assert(x<=10);
  while (x<SIZE) {
   x=x+1;
   assert(x<=10);
  }
  assert(x<=10);
  assert(x==10);
}
```

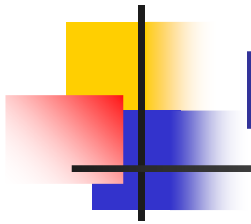
Qual seria o loop invariante? X<=10

Qual seria a condição de parada? X==10



Loop Invariante (1)

- *"No começo de cada iteração do loop **para** (linhas de 1 a 8), o subarranjo A[1..j-1] consiste dos elementos contidos originalmente em A[1..j-1] , mas em sequência ordenada"*
- Prova das propriedades:
  - **Inicialização:** mostrar que o loop invariante é válido antes da primeira iteração do loop
    - quando j=2 então o subarranjo A[1..j-1] consiste apenas do único elemento A[1], que é de fato o elemento original A[1]. Além disso, esse subarranjo é ordenado e isso mostra que o loop invariante é válido antes da primeira iteração



Loop Invariante (2)

- **Manutenção:** demonstrar que cada iteração mantém o loop invariante
  - O corpo do loop **para** funciona deslocando-se A[j-1], A[j-2], A[j-3] e daí por diante uma posição a esquerda, até ser encontrada a posição adequada para A[j] (linhas 3 a 7), e nesse ponto o valor de A[j] é inserido
- **Término:** examinar o que ocorre quando o loop termina
  - O loop **para** externo termina quando *j* excede *n*, isto é, quando *j=n+1*. Substituindo *j* por *n+1* no enunciado do loop invariante, temos que o subarranjo *A[1..n]* consiste nos elementos originalmente contidos em *A[1..n]*, mas em sequência ordenada

Loop Invariante (Ordenação)

- O algoritmo é dito **correto** se a condição de parada é alcançada e o loop invariante é verdadeiro

```
1 para j=2 até n
2 // Invariante:  A[1..j-1] é uma seq. ordenada do
original A[1..j-1]
3   faça pivot:=A[j]
4      i ← j-1
5      enquanto i>0 e A[i]>pivot faça
6        A[i+1] ← A[i]
7        i--;
8      fim-enquanto
9      A[i+1] ← pivot;
10 // Invariante:  A[1..j-1] é uma seq. ordenada do
original A[1..j-1]
11 Invariante é verdadeira A[1..n] e j==n+1
```

Exercício: Loop Invariante

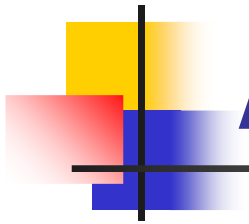
- Enuncie um loop invariante para o loop do seguinte filtro digital:

$$y(n) = 0.5\,y(n-1) + x(n)$$

$$x(n) = \begin{cases} 1, & n = 0 \\ 0, & n \neq 0 \end{cases}$$

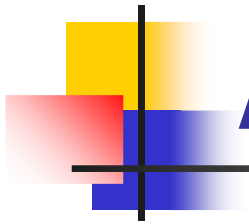
```
for(i=0; i<n; i++) {
  if (i==0)
    y[0]=x[0];
  else
    y[i] = 0.5*y[i-1]+x[i];
}
```

```
for(i=0; i<n; i++) {
  if (i==0)
    x[0]=1;
  else
    x[i]=0;
}
```



Análise de Algoritmos (1)

- Prever os recursos que o algoritmo necessitará
  - Memória, largura de banda ou hardware
- Estamos interessado na eficiência:
  - Tempo de execução
  - Espaço (memória) usado
- Eficiência como função do tamanho da entrada
- Pela análise de vários algoritmos, podemos identificar um algoritmo mais eficiente



Análise de Algoritmos (2)

- O que deve ser contabilizado?
  - Modelo RAM (instruções executadas de formal seq.)
    - Não abusar do modelo de RAM (instrução de ordenação)
  - Instruções (considerando tempo constante):
    - Aritméticas (+, -, *, etc.)
    - Movimentação de dados (carregar, armazenar, copiar)
    - Controle (desvios, chamadas de procedimento, etc.)
  - Tamanho da entrada
    - Número de itens na entrada
    - Número total de bits

Análise: Ordenação por Inserção

- Determinar o **tempo de execução** como função do tamanho da entrada

```
para j=2 até n
    faça pivot:=A[j]
       i ← j-1

       enquanto i>0 e A[i]>pivot faça

          A[i+1] ← A[i]
          i--;

       fim-enquanto

       A[i+1] ← pivot;
```

n vezes

n-1 vezes

Para cada uma das n-1 vezes, t_j vezes

Análise: Ordenação por Inserção

- Determinar o **tempo de execução** como função do tamanho da entrada

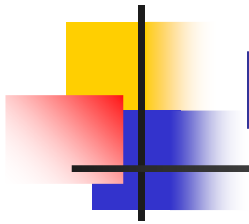
	vezes	custo
para j=2 **até** n	n	c_1
faça pivot:=A[j]	(n-1)	c_2
i ← j-1	(n-1)	c_3
enquanto i>0 **e** A[i]>pivot **faça**	$\sum_{j=2}^{n} t_j$	c_4
A[i+1] ← A[i]	$\sum_{j=2}^{n} (t_j - 1)$	c_5
i--;	$\sum_{j=2}^{n} (t_j - 1)$	c_6
fim-enquanto		
A[i+1] ← pivot;	n-1	c_7

Análise: Ordenação por Inserção

- Para calcular T(n), somamos os produtos das colunas custos e vezes, obtendo:

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4 \sum_{j=2}^{n} t_j + c_5 \sum_{j=2}^{n} (t_j - 1) + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7(n-1)$$

- Mesmo para entradas de um **mesmo tamanho**, o tempo de execução do algoritmo pode depender de qual entrada é dada
  - Análise do melhor caso
  - Análise do pior caso
  - Análise do caso médio



Melhor Caso

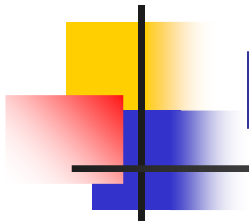
- **Melhor Caso**
  - Elementos já ordenados
  - Todos os testes falham, o loop interno nunca é executado
  - Neste caso, $t_j=1$ e a operação é executada $n-1$ vezes

$$\sum_{j=2}^{n} t_j = t_2 + t_3 + \ldots + t_n = 1 + 1 + \ldots + 1 = n - 1$$

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4(n-1) + c_7(n-1)$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_7) \cdot n - (c_2 + c_3 + c_4 + c_7)$$

  - Portanto, a complexidade de tempo é dada por $T(n)=an-b$ para constantes **a** e **b** que dependem de $\mathbf{c_i}$
    - Função linear de **n**



Pior Caso

- **Pior Caso**
  - Elementos em ordem inversa
  - Observando que:

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

  - Descobrimos que, no pior caso, T(n) é

$$T(n) = c_1 n + c_2(n-1) + c_3(n-1) + c_4\left(\frac{n(n+1)}{2} - 1\right) + c_5\left(\frac{n(n-1)}{2}\right) + c_6\left(\frac{n(n-1)}{2}\right)$$

$$+ c_7(n-1) = \left(\frac{c_4}{2} + \frac{c_5}{2} + \frac{c_6}{2}\right)n^2 + \left(c_1 + c_2 + c_3 + \frac{c_4}{2} - \frac{c_5}{2} - \frac{c_6}{2} + c_7\right)n - (c_2 + c_3 + c_4 + c_7)$$

Demonstração do Somatório

Série aritmética é definida como:

$$\sum_{j=1}^{n} j = 1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Deste modo, temos:

$$\sum_{j=2}^{n} j = 2 + 3 + \ldots + n = \frac{n(n+1)}{2} - 1$$

De forma similar, obtemos:

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

Somatórios Úteis

$$\sum_{i=1}^{n} i = 1 + 2 + \ldots + n = \frac{n(n+1)}{2}$$

$$\sum_{i=0}^{5} (i+1) = 1 + 2 + 3 + 4 + 5 + 6$$

$$\sum_{i=1}^{n} i^3 = 1^3 + 2^3 + \ldots + n^3 = \left(\frac{n(n+1)}{2}\right)^2$$

$$\sum_{i=1}^{n} a = \underbrace{(a + a + a + \ldots + a)}_{n \text{ vezes}} = na$$

$$\sum_{j=0}^{n} a^j = 1 + a + a^2 + a^3 + \ldots + a^n = \frac{1 - a^{n+1}}{1 - a}$$

$$\sum_{k=1}^{n} a^k = a + a^2 + a^3 + a^4 \ldots + a^n = \frac{a^{n+1} - a}{a - 1}$$

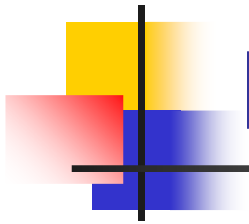
$$\sum_{i=1}^{n} \frac{1}{a^i} = \frac{a^n - 1}{a^{n+1} - a^n}$$

$$\sum_{k=1}^{n} ak = a + 2a + 3a + \ldots + na = a \sum_{k=1}^{n} k$$

$$\sum_{i=0}^{n} i = 0 + \sum_{i=1}^{n} i = \sum_{i=1}^{n} i$$

$$\sum_{i=1}^{n} x_i^2 = x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2 \neq \left(\sum_{i=1}^{n} x_i\right)^2$$

$$\left(\sum_{i=1}^{n} \sum_{j=1}^{m} x_i y_j\right) = \left(\sum_{i=1}^{n} x_i \sum_{j=1}^{m} y_j\right)$$

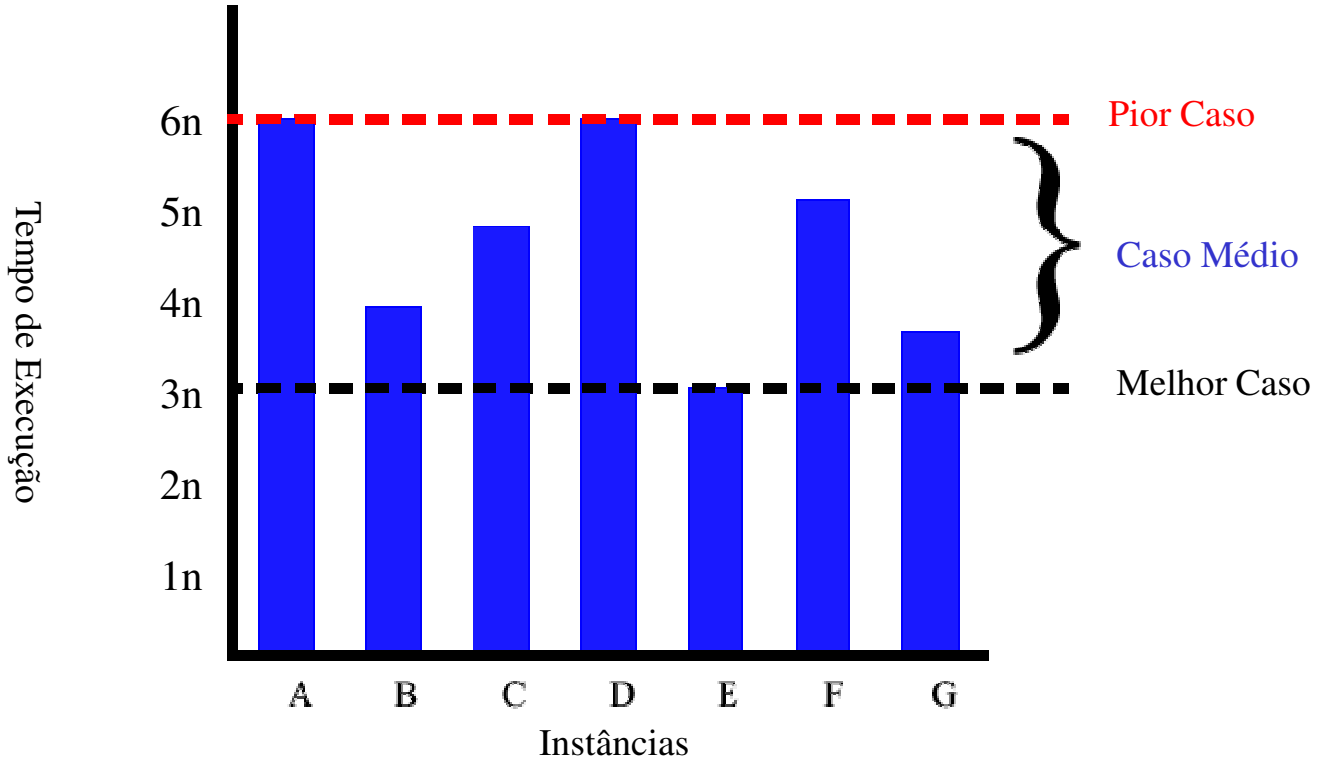


Melhor/Pior/Médio (1)

- **Melhor Caso:** elementos já ordenados
  - $t_j=1$, $T(n) = k_1 . n$, ou seja, tempo linear.
- **Pior caso**: elementos em ordem inversa
  - $t_j=j$, $T(n) = k_2 . n^2$, ou seja, tempo quadrático
- **Tempo médio**:
  - $t_j=j/2$, $T(n) = k_2 . n^2$, ou seja, tempo quadrático

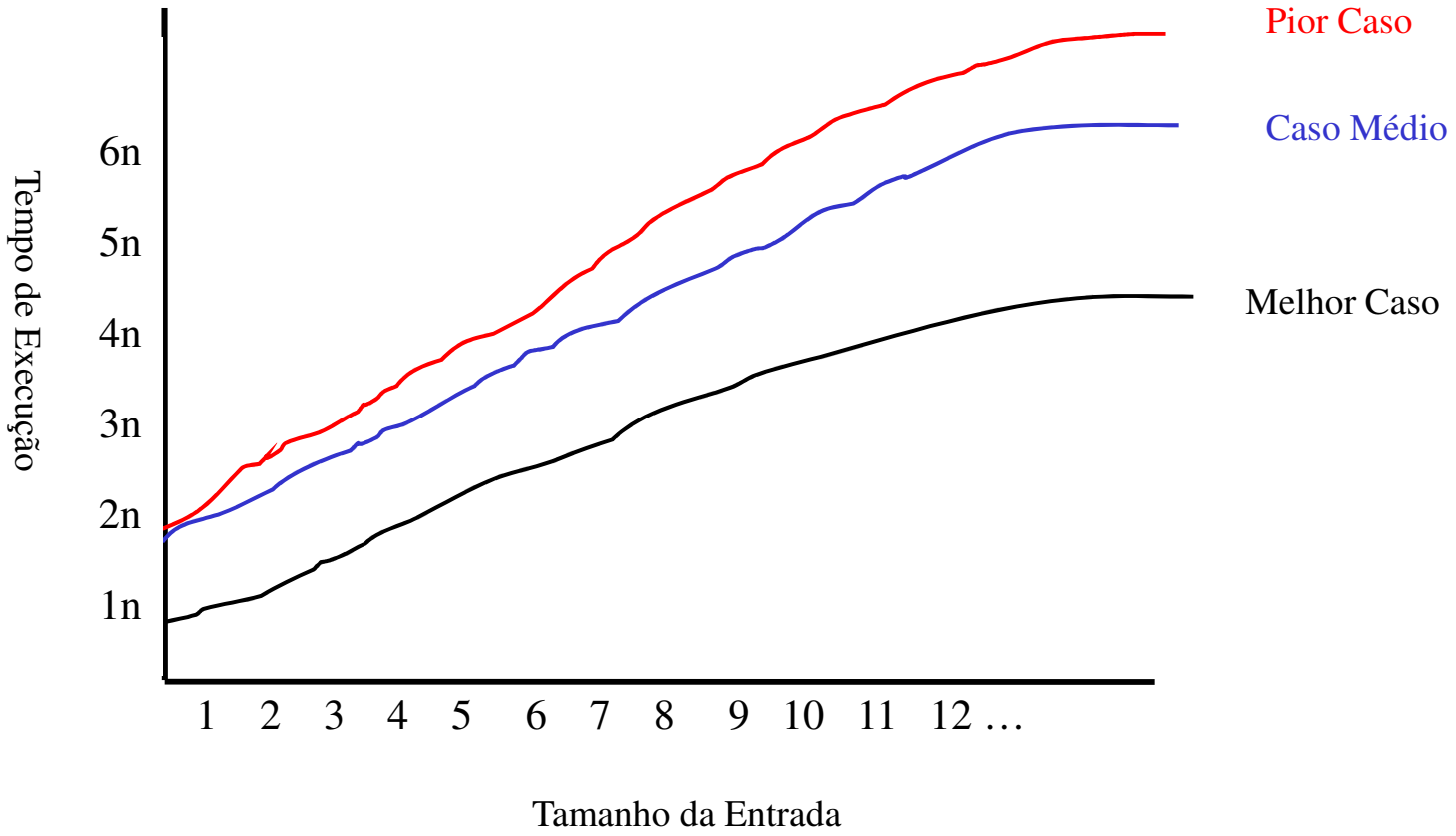
Melhor/Pior/Médio (2)

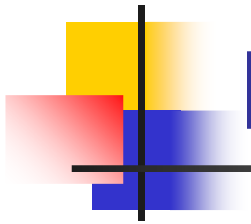
- Determinar o tempo de execução para diferentes instancias de tamanho n:



Melhor/Pior/Médio (3)

- Para instâncias de todos os tamanhos:

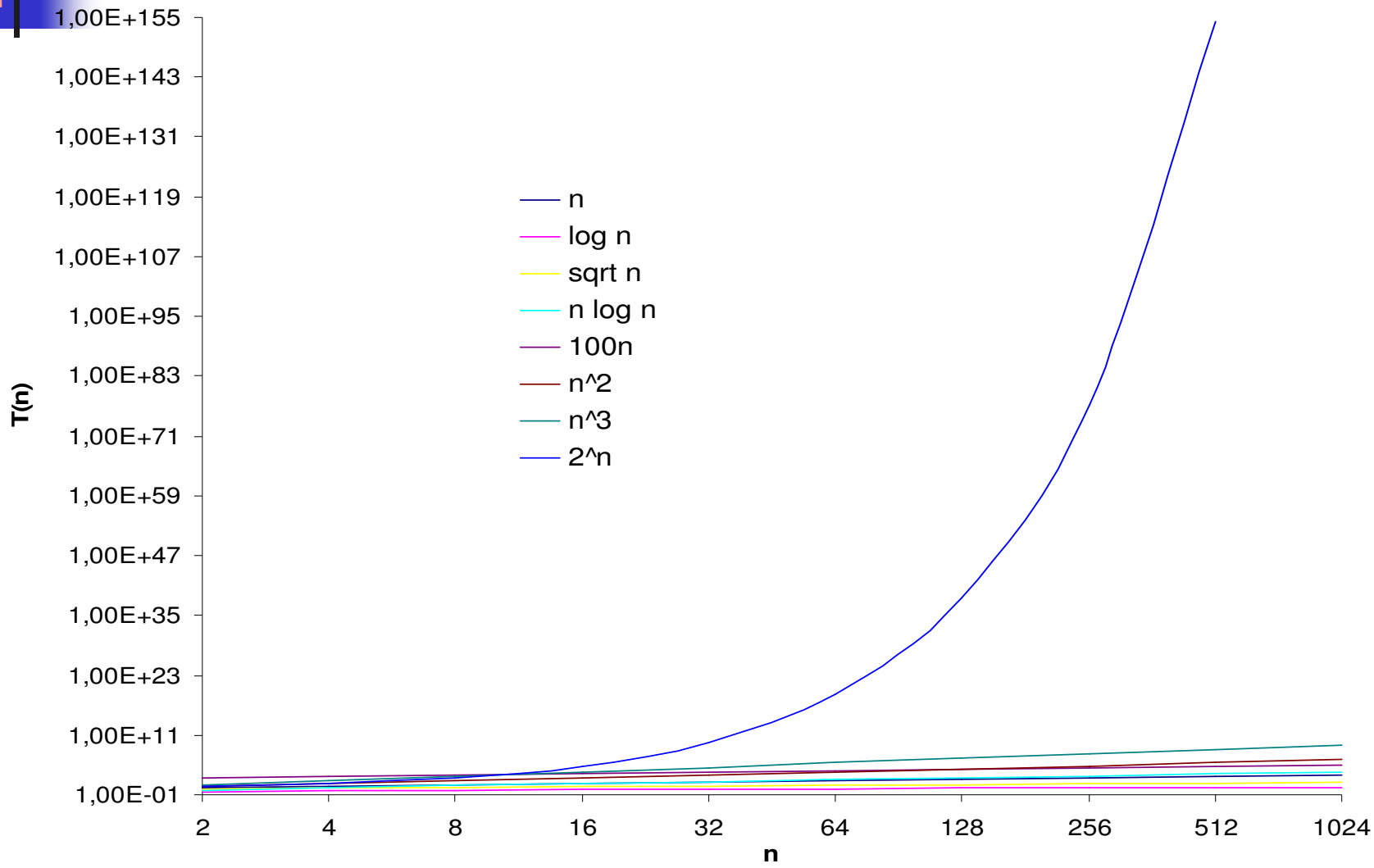




Melhor/Pior/Médio (4)

- **O pior caso** é geralmente usado
  - Estabelece um limite superior na complexidade de tempo do algoritmo
  - Para alguns algoritmos o pior caso é bastante freqüente
  - Frequentemente o caso médio é tão ruim quando o pior caso
  - Encontrar o caso médio pode ser muito difícil

Funções de Crescimento



O que é razoável em computação?

função/ n	10	20	50	100	300
n^2	1/10,000 segundos	1/2,500 segundos	1/400 segundos	1/100 segundos	9/100 segundos
n^5	1/10 segundos	3.2 segundos	5.2 minutos	2.8 horas	28.1 dias
2^n	1/1000 segundos	1 segundos	35.7 anos	400 trilhões de séculos	Séculos de 75 dígitos
n^n	2.8 horas	3.3 trilhões de anos	Seculos de 70 dígitos	Séculos de 185 dígitos	Séculos de 728 dígitos

Polinomial

Exponencial

Exemplo 2: Busca

ENTRADA

- sequência de números
- um número (consulta)

$a_1, a_2, a_3, \ldots, a_n;\ q$

2 5 4 10 7; 5

2 5 4 10 7; 9

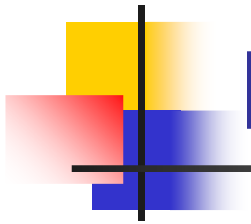
SAÍDA

- índice do número or *NIL*

j

2

NIL



Busca

```
j=1
while j<=length(A) and A[j]!=q
   do j++
if j<=length(A) then return j
else return NIL
```

- Pior caso: *f(n)=n*, caso-médio: *n/2*
- Não se pode fazer melhor! Este é um limite inferior para o problema de busca em uma seqüência arbritrária

Exemplo 2: Busca

ENTRADA

- sequência de números ordenada não-descendente
- um número (consulta)

$a_1, a_2, a_3, \ldots, a_n;\ q$

2 4 5 7 10; 5

2 4 5 7 10; 9

SAÍDA

- o índice do número encontrado ou *NIL*

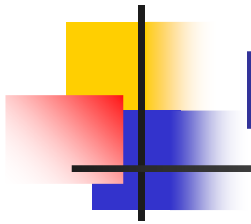
j

3

NIL

A sequência ordenada ajudou na busca?

Não, pois realizamos mais operações para encontrar o número



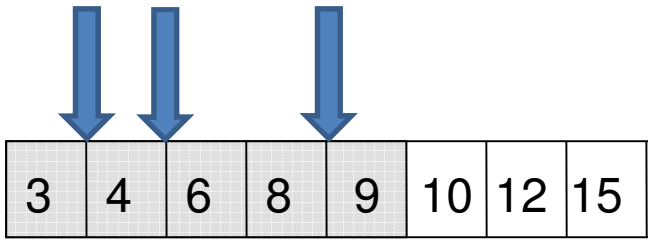
Busca Binária

- Parte do pressuposto que o vetor está ordenado e realiza sucessivas divisões do espaço de busca

```
left=1
right=length(A)
do
 j=(left+right)/2
 if A[j]==q then return j
 else if A[j]>q then right=j-1
 else left=j+1
while left<=right
return NIL
```

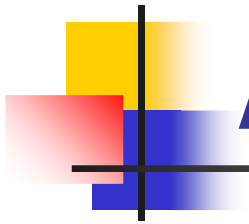
Busca Binária – Análise

- Quantas vezes o laço é executado?
  - A cada interação o número de posições n é cortado ao meio
  - Quantas vezes se corta ao meio n para se chegar a 1?
  - $lg_2\ n$



$$\lg_2 n = x \Leftrightarrow n = 2^x$$

$$\lg_2 8 = 3$$

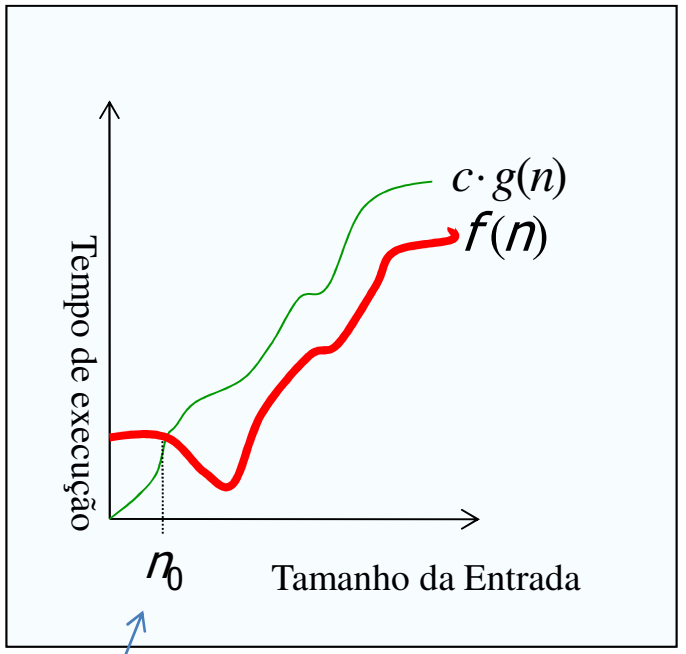


Análise Assintótica

- **Objetivo**: simplificar a análise de complexidade de tempo eliminando "detalhes" que podem ser afetados por fatores de implementação ou hardware
  - $3n^2 \rightarrow n^2$
- **Comportamento assintótico**: como o tempo de execução do algoritmo cresce com o tamanho da entrada

Notação Assintótica (1)

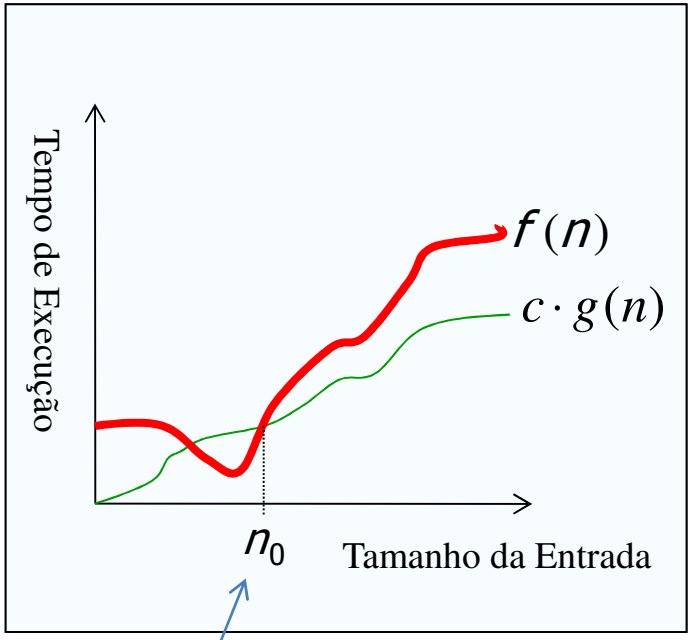
- *Notação O (O* - Notation)
  - Limite assintótico superior
  - *f(n) = O(g(n))*, se existem constantes positivas *c* e n_0 tais que **$0 \leq f(n) \leq c\ g(n)$** para $n \geq n_0$
  - *f(n)* e *g(n)* são funções sobre inteiros não negativos
- Usada para análise do pior caso (limite assintótico superior)



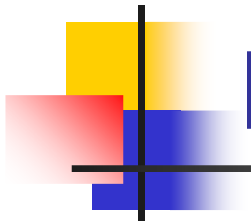
Para todos os valores *n* à direita de n_0, o valor da função *f(n)* está em ou abaixo de *g(n)*

Notação Assintótica (2)

- Notação Ω
  - limite assintótico inferior
  - $f(n) = \Omega(g(n))$ se existem constantes positivas c e n_0, *tais que* $\mathbf{0 \leq c\ g(n) \leq f(n)}$ para $n \geq n_0$
- Usada para descrever o melhor caso de tempos de execução ou limites inferiores de problemas algorítmicos
  - P.ex., o limite inferior para busca em lista não ordenada é $\Omega(n)$.



Para todos os valores n à direita de n_0, o valor da função $f(n)$ está em ou acima de $g(n)$

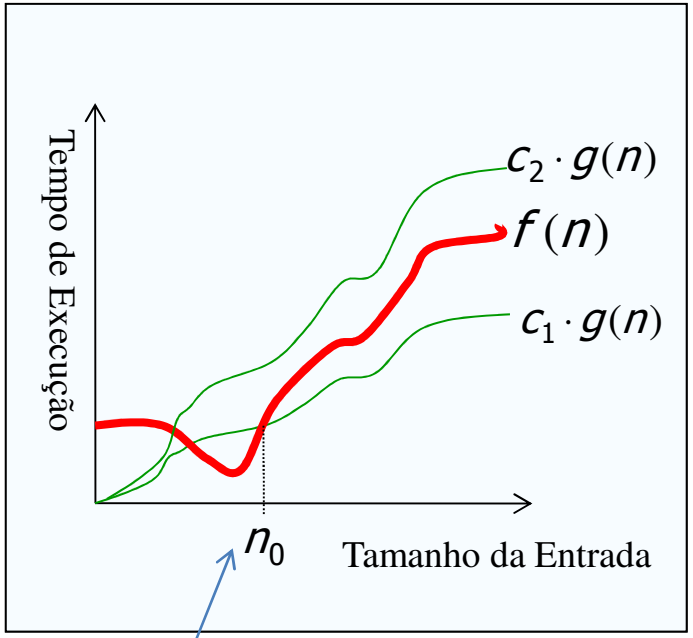


Notação Assintótica (3)

- Regra prática: Remover termos de menor ordem e fatores constantes.
  - $50\ n \log n$ é $O(n \log n)$
  - $7n - 3$ é $O(n)$
  - $8n^2 \log n + 5n^2 + n$ é $O(n^2 \log n)$
- Costuma-se utilizar aproximações de menor ordem possível
- Nota: Algoritmo assintoticamente mais eficiente será a melhor escolha exceto para entradas pequenas

Notação Assintótica (4)

- Notaçao Θ
  - Limite assintótico máximo e mínimo
  - $f(n) = \Theta(g(n))$ se existem constantes c_1, c_2, e n_0, *tais que* $\mathbf{0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)}$ para $n \geq n_0$
- $f(n) = \Theta(g(n))$ *se, e somente se,* $f(n) = O(g(n))$ e $f(n) = \Omega(g(n))$
- $O(f(n))$ é frequente confundido com $\Theta(f(n))$



Para todos os valores n à direita de n_0, o valor da função $f(n)$ reside em $c_1g(n)$ ou acima dele e em $c_2g(n)$ ou abaixo desse valor

Exercício: Notação Assintótica

- Use a definição formal de Θ para mostrar que

$$\frac{1}{2}n^2 - 3n = \Theta\left(n^2\right)$$

Solução:

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

Note que c_1 e c_2 devem ser **constantes positivas**

$$para \text{ todo } n \ge n_0.$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2 \therefore c_2 = \frac{1}{2} \; para \; n \ge 1$$

Quando *n* tende ao infinito, o termo ½ prevalece pois c_2 determina o **limite superior**

$$c_1 \le \frac{1}{2} - \frac{3}{n} \therefore c_1 = 0.5 - 0.428571 \therefore c_1 = \frac{1}{14} \text{ para } n \ge 7$$

n=7 é o menor valor para que c_1 seja uma **constante positiva**

Exercício: Notação Assintótica

- Use a definição formal de Θ para mostrar que

$$6n^3 \neq \Theta\left(n^2\right)$$

Solução:

$$c_1 n^2 \leq 6n^3 \leq c_2 n^2 \ \textit{para todo n} \geq n_0.$$

$$6n \leq c_2 \therefore n \leq \frac{c_2}{6}$$

O que não pode ser válido para um valor de *n* arbitrariamente grande, pois c_2 é uma constante

Notação Assintótica (5)

- Notação *o* : *f(n)=o(g(n))*
  - **Para todo** *c positivo,* deve existir $n_0 > 0$, tal que **0 ≤ f(n) < c g(n)** para $n \geq n_0$
  - O limite assintótico superior fornecido pela notação *O* pode ser ou não assintoticamente restrito
    - O limite $2n^2 = O(n^2)$ é, mas $2n = O(n^2)$ não é
  - Usamos a notação *o* para denotar um limite superior que não é assintoticamente restrito
  - Se *f(n)=o(g(n))*, dizemos que *g(n)* domina *f(n)*
- Notação ω: *f(n)=ω(g(n))*
  - Análoga a função *o* com relação à Ω

Notação Assintótica (6)

- Analogia com número reais
  - $f(n) = O(g(n))$ $\cong$ $f \leq g$
  - $f(n) = \Omega(g(n))$ $\cong$ $f \geq g$
  - $f(n) = \Theta(g(n))$ $\cong$ $f = g$
  - $f(n) = o(g(n))$ $\cong$ $f < g$
  - $f(n) = \omega(g(n))$ $\cong$ $f > g$
- Abuso de notação:
  - $f(n) = O(g(n))$ indica que $f(n) \in O(g(n))$

Exercícios: Notação Assintótica

- Verifique se as afirmativas são corretas:
- No pior caso, o algoritmo de inserção é

$$\Theta(n^2)$$ **Verdadeiro**

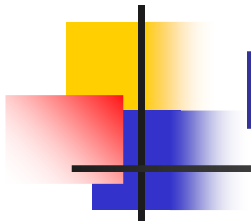
$2^{2n} = O(2^n)$ **Falso**

$2^{n+1} = O(2^n)$ **Verdadeiro**

$\Theta(n) + \Theta(1) = \Theta(n)$ **Verdadeiro**

$O(n^2) + O(n^2) = O(n^2)$ **Verdadeiro**

$O(n) \times O(n) = O(n)$ **Falso**

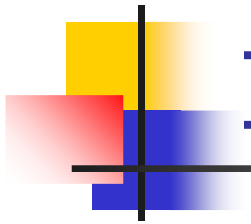


Exercício: *Bubble Sort*

- Determine o tempo de execução do algoritmo de *bubble sort*

```
1 para i=1 até n
2   faça para j=n até i+1
3     faça se A[j] < A[j-1]
4       então trocar A[j]↔ A[j-1]
```

- Qual é o tempo de execução no pior caso? Como ele se compara ao tempo de execução da ordenação por inserção?
- Enuncie um loop invariante para o loop das linhas 1 a 4



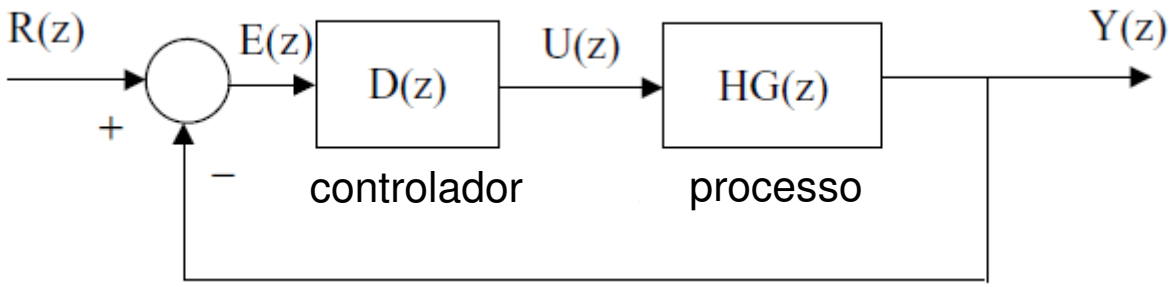
Implmentação do *Bubble Sort*

- Compare o tempo de execução da ordenação por inserção com o *bubble sort*

```
#define SIZE 140
int item[SIZE];
void bubblesort() {
int a, b, t;
  for(a = 1; a < SIZE; ++a) {
    for(b = SIZE-1; b >= a; --b) {
      /* compare adjacent elements */
      if (b-1 < SIZE && b < SIZE){
        if(item[ b - 1] > item[ b ]){
          /* exchange elements */
          t = item[ b - 1];
          item[ b - 1] = item[ b ];
          item[ b ] = t;
}  }  }  }  }
```

Controladores Digitais

- Controladores digitais podem ser implementados por computadores, microprocessadores ou DSPs
  - A função de transferência pode ser representada por uma relação de dois polinômios



$$D(z) = \frac{U(z)}{E(z)} = \frac{\sum_{j=0}^{n} b_j z^{-j}}{1 + \sum_{j=1}^{n} a_j z^{-j}}$$

Controladores Digitais: Exemplo

- Considere o seguinte controlador digital representado por uma equação de diferença *(b_1=0.5 e a_1=-0.5)*

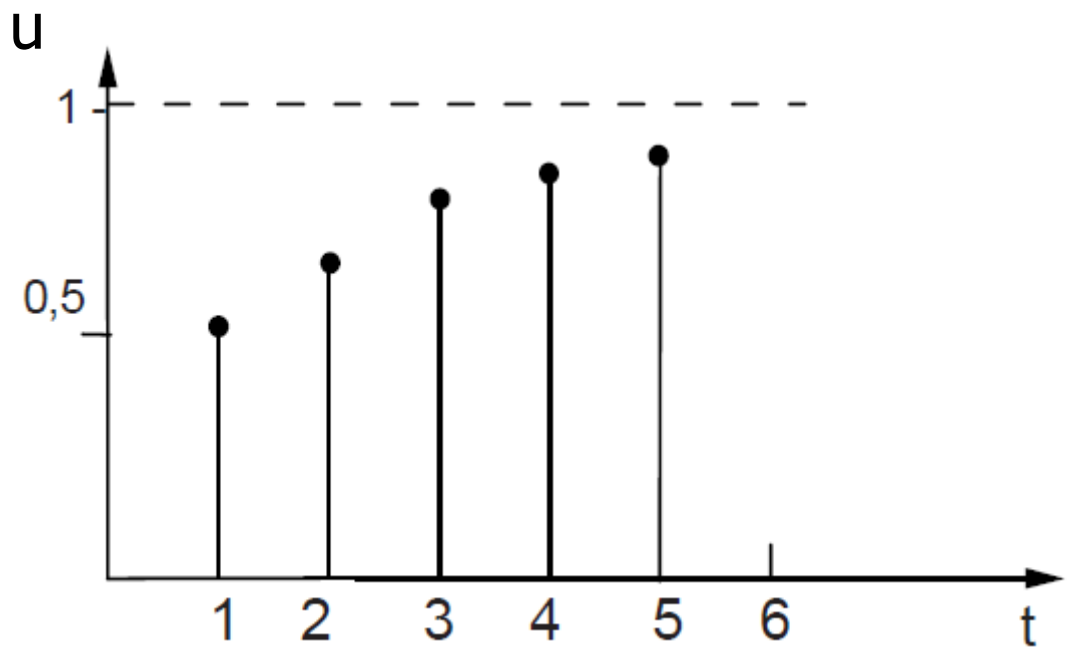
$$u(z)\times\left(1+a_1 z^{-1}\right)=e(z)\left(b_0+b_1 z^{-1}\right)$$
$$u(t)+a_1 u(t-1)=b_0 e(t)+b_1 e(t-1)$$
$$u(t)=-a_1 u(t-1)+b_1 e(t-1)$$

$$e(t)=\begin{cases}0, & t<0\\ 1, & t\geq 0\end{cases}$$

- Os valores de saída para diferentes instantes são:

t	0	1	2	3	4	5
u(t)	0	0.5	0.75	0.875	0.937	0.969

Controladores Digitais: Exemplo



- A resposta obtida assemelha-se a resposta ao degrau de um sistema de primeira ordem de tempo contínuo

Controladores Digitais: Exemplo

- Considere agora os seguintes coeficientes para a equação de diferença *(b_1=1.5 e a_1=0.5)*

$$u(t) = -a_1 u(t-1) + b_1 e(t-1)$$
$$u(t) = -0.5u(t-1) + 1.5e(t-1)$$

$$e(t) = \begin{cases} 0, & t < 0 \\ 1, & t \geq 0 \end{cases}$$

- Os valores de saída para diferentes instantes são:

T	0	1	2	3	4	5
u(t)	0	1.5	0.75	1.125	0.937	1.062

Implementação de Controladores Digitais em ANSI-C (1)

```
#include <assert.h>
#include <stdio.h>
const int xLen = 10;
const int Alen = 2;
const int Blen = 1;
int main() {
  float A[] = {1.0f, -0.5f};
  float B[] = {0.5f};
  int i,j;
  float x[xLen];
  float x_aux[xLen];
  float y[xLen];
  float y_aux[xLen];
```

Implementação de Controladores Digitais em ANSI-C (2)

```
for(i=0;i<xLen;i++) {
  x[i]=1;
  x_aux[i]=0;
  y_aux[i]=0;
}
for(i=0;i<xLen;i++) {
  y[i] = 0; //clear y
  /* Updating past x values */
  for (j=Blen-1;j>=1;j--)
    x_aux[j] = x_aux[j-1];
  x_aux[0] = x[i];
  /* Num, x values */
  for(j = 0; j < Blen; j++)
    y[i] = y[i] + B[j]*x_aux[j];
```

Implementação de Controladores Digitais em ANSI-C (3)

```
  /* Den, y values */
  for(j=0;j<Alen-1;j++)
    y[i] = y[i] - A[j+1]*y_aux[j];
  /* Updating past y values */
  for(j=Alen-2;j>=1;j--)
    y_aux[j] = y_aux[j-1];
  y_aux[0] = y[i];
}
```