MC3305
Algoritmos e Estruturas de Dados II

Aula 01 – Introdução
Custo de um algoritmo, Funções de complexidad e Recursão

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

2Q-2015

Custo de um algoritmo

Estrutura de dados

- Estrutura de dados e algoritmos estão intimamente ligados:
  - Não se pode estudar ED sem considerar os algoritmos associados a elas;
  - Asssim como a escolha dos algoritmos (em geral) depende da representação e da ED.

(1) Análise de um algoritmo particular

- **Qual é o custo de usar um dado algoritmo** para resolver um problema específico?

- Características que devem ser investigadas:
  - Tempo de execução.
  - Quantidade de memória.

(2) Análise de uma **classe** de algoritmos

- Qual é o algoritmo de menos custo possível para resolver um problema particular?
- **Toda uma familia de algoritmos é investigada.**
- Procura-se identificar um que seja o **melhor possível.**
- Colocam-se **limites** para a complexidade computacional dos algoritmos pertencentes à classe.

Custo de um algoritmo

- Se conseguirmos **determinar o menor custo possível** para resolver problemas de uma dada classe, então teremos a **medida da dificuldade inerente para resolver o problema**.

- Quando um algoritmo é igual ao menor custo possível, o **algoritmo é ótimo** para a medida de custo considerada.

- Podem existir **vários algoritmos** para resolver um mesmo problema.

  → Se a mesma medida de custo é aplicada a diferentes algoritmos então é possível **compará-los** e escolher o mais adequado.

Medida de custo pela execução de um programa em uma plataforma real

(1) Medida de custo pela execução de um programa em uma plataforma real

- Tais medidas são bastante inadequadas e os resultados jamais devem ser generalizados:
  - Os resultados são **dependentes do compilador** que pode favorecer algumas construções em detrimento de outras;
  - Os resultados **dependem de hardware**;
  - Quanto grandes quantidades de memória são utilizadas, as medidas de tempo podem depender deste aspecto.

- Apesar disso, há argumentos a favor de se obterem medidas reais de tempo:
  - Exemplo: Quando há vários algoritmos distintos para resolver o problema;
  - Assim, são considerados tanto os custos reais das operações como os custos não aparentes, tais como alocação de memória, indexação, carga, dentre outros.

Medida de custo por meio de um modelo matemático

(2) Medida de custo por meio de um modelo matemático

```c
int F1(int a, int b) {
    int i, t1, t2;

    t1 = a;
    t2 = b;

    a = t2;
    b = t1;

    for (i=a; i<b; i++)
        // ...
```

```c
int F2(int a, int b) {
    int i, t;

    t = a;

    a = b;
    b = t;

    for (i=a; i<b; i++)
        // ...
```

(2) Medida de custo por meio de um modelo matemático

- Usa um **modelo matemático** baseado em um **computador idealizado**.

- Deve ser especificado o **conjunto de operações e seus custos de execuções**.

- É mais usual ignorar o custo de algumas das operações e **considerar apenas as mais significantes.**
  - Em algoritmos de ordenação:

    Consideramos o conjunto de comparações entre os elementos do conjunto a ser ordenado e ignoramos as operações aritméticas, de atribuição e manipulação de índices, caso existam.

Função de complexidade

Função de complexidade

- Para medir o custo de execução de um algoritmo, é comum definir uma função de custo **ou função de complexidade *f*.**

- Função de **complexidade de tempo**:

  $f(n)$ mede o tempo necessário para executar um algoritmo para um problema de tamanho ***n***.

- Função de **complexidade de espaço**:

  $f(n)$ mede a memória necessária para executar um algoritmo para um problema de tamanho ***n***.

Comportamento assintótico de funções

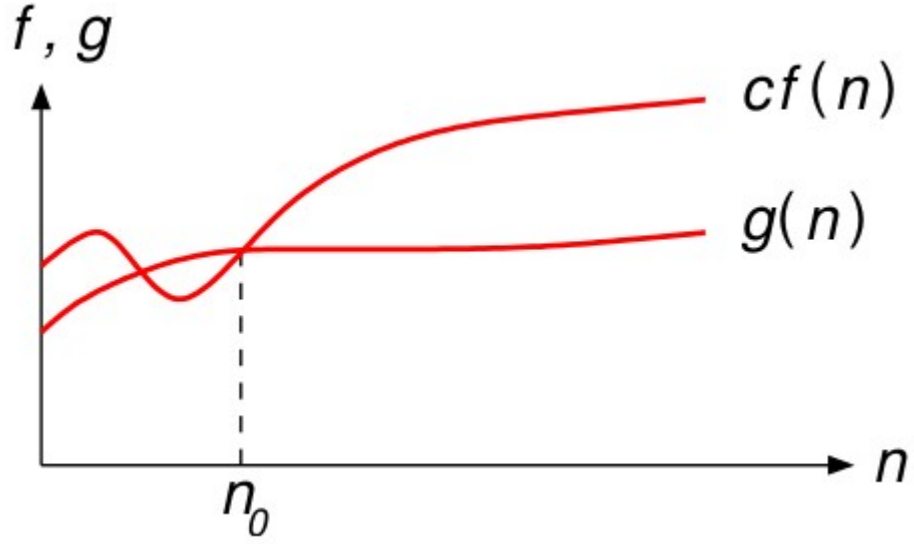
- A análise de algoritmos é realizada p**ara valores grandes de n**.
- Estudaremos o comportamento assintótico das **funções de custo**.
- O comportamento assintótico de **f(n)** representa o limite do comportamento de custo, quando **n** cresce.

Dominação assintótica

Definição:

Uma função ***f(n)*** **domina assintoticamente** uma outra função ***g(n)*** se existem duas constantes positivas **c** e n_0 tais que, para $n \geq n_0$, temos:

$$|g(n)| \leq c|f(n)|$$

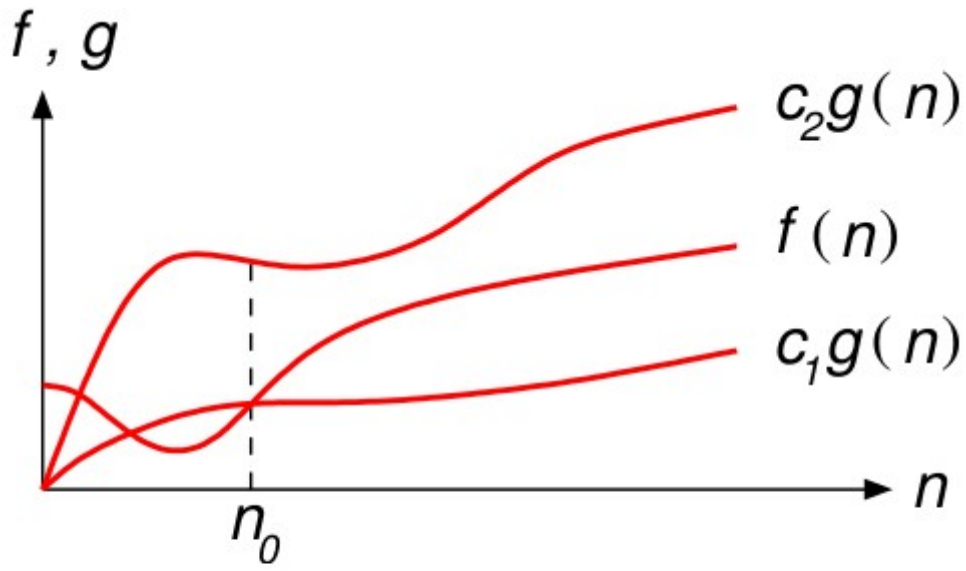


Notação assintótica de funções

Existem 3 notações assintóticas de funções:

- Notação Θ
- Notação O $(\text{`}O \textit{ grande}\text{'})$
- Notação Ω

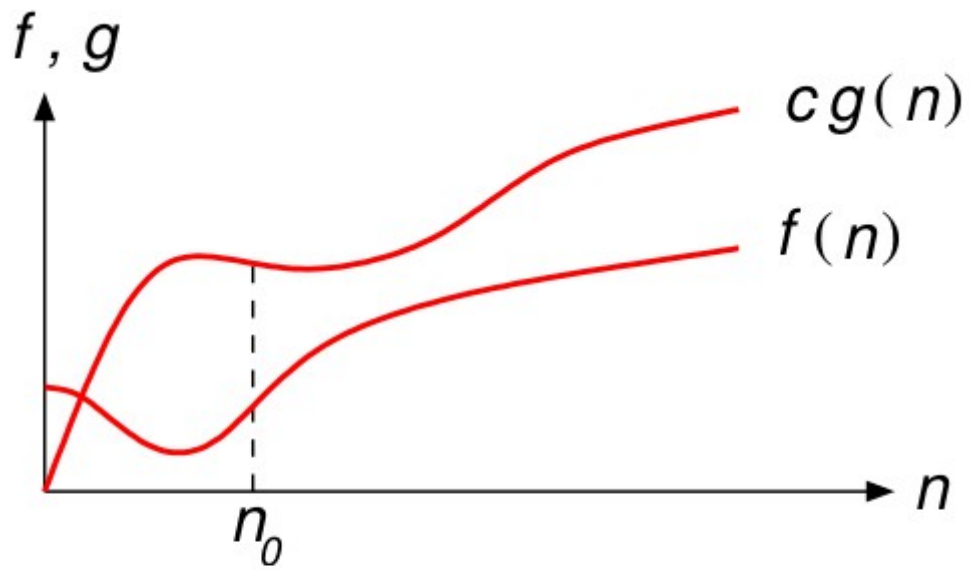
Notação Θ



$$f(n) = \Theta(g(n))$$

g(n) é um limite assintótico firme de f(n)

Notação O (‘O grande’)

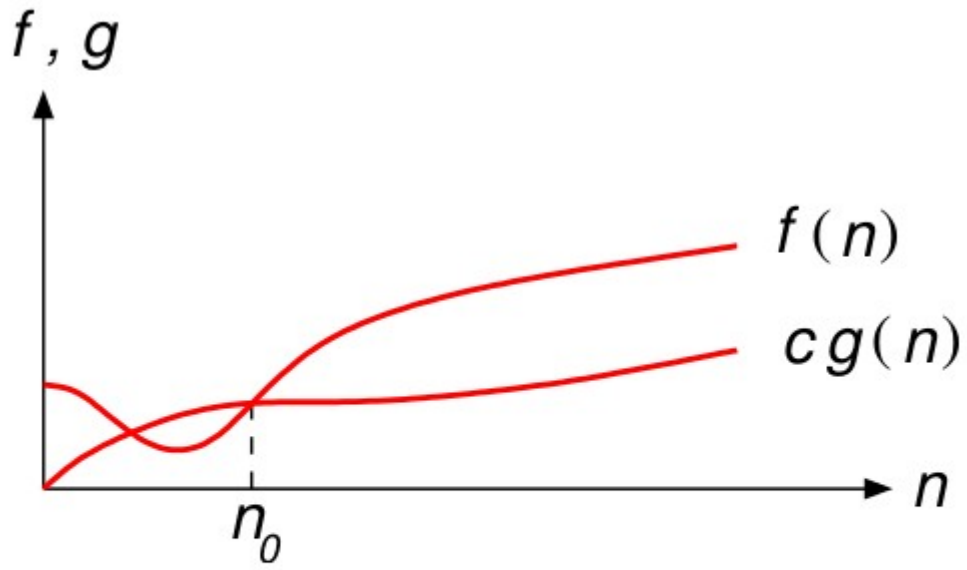


$$f(n) = O(g(n))$$

f(n) é da ordem no máximo g(n)

O é usada para expressar o tempo de execução de um algoritmo no **pior caso**, está se definindo também o limite (superior) do tempo de execução desse algoritmo **para todas as entradas**.

Notação Ω



$$f(n) = \Omega(g(n))$$

Omega: Define um limite inferior para a função, por um fator constante.

g(n) é um limite assintoticamente inferior

Teorema

Para quaisquer funções $f(n)$ e $g(n)$,

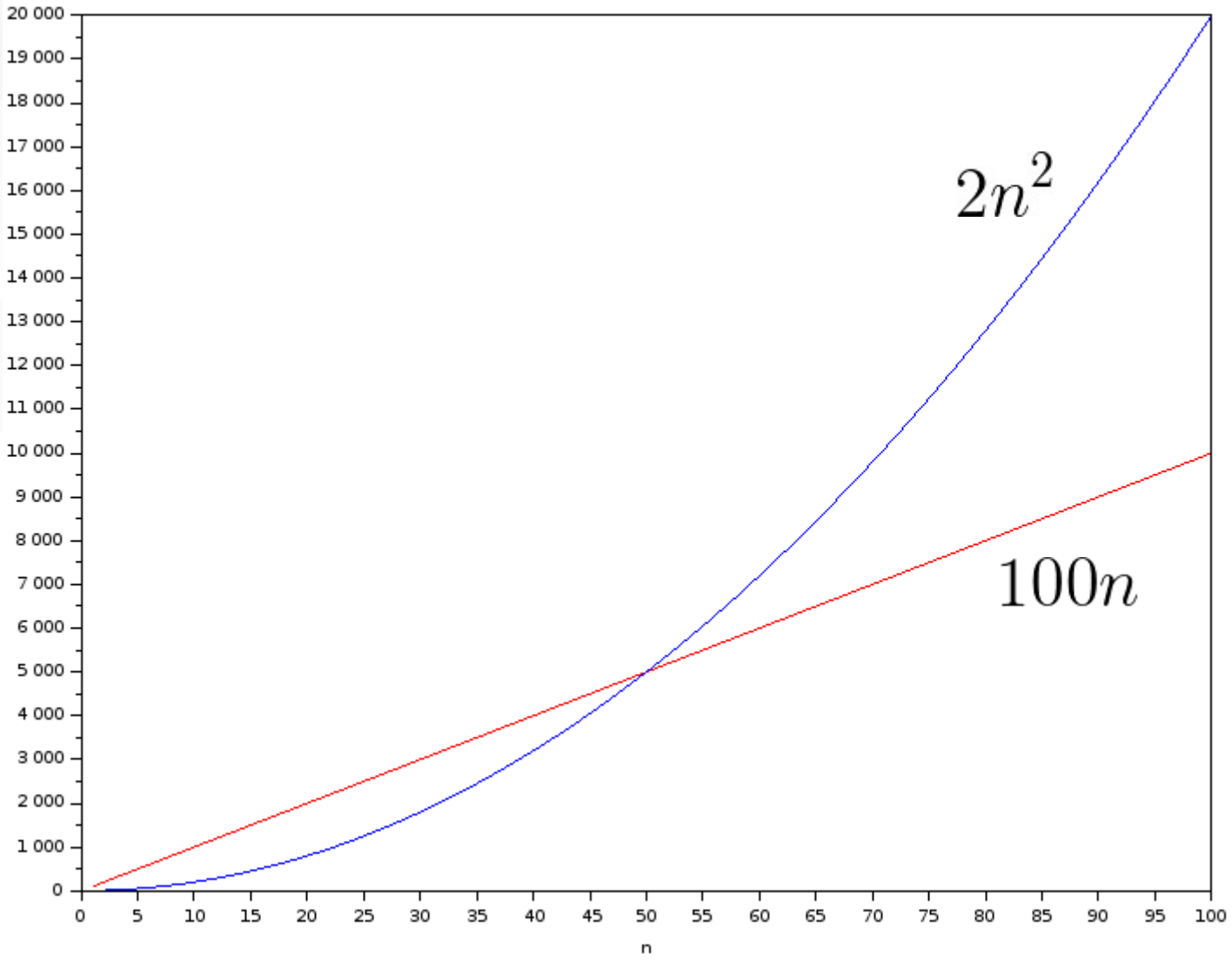
$$f(n) = \Theta(g(n))$$

se e somente se,

$$f(n) = O(g(n)), \text{ e}$$

$$f(n) = \Omega(g(n))$$

Comparação de programas



Hierarquias de funções

A seguinte herarquia de funções pode ser definida do ponto de vista assintótico:

$$1 \prec \log\log n \prec \log n \prec n^{\epsilon} \prec n^{c} \prec n^{\log n} \prec c^{n} \prec n^{n} \prec c^{c^{n}}$$

onde c e ϵ são constantes arbitrárias com $0 < \epsilon < 1 < c$

ATIVIDADE 01: Hierarquias de funções

$$\frac{N^2}{2} - \frac{N}{2}$$

$$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$$

$$\lg N + 1$$

$$\frac{N(N+1)}{2}$$

ATIVIDADE 01: Hierarquias de funções

$\frac{N^2}{2} - \frac{N}{2}$

$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$

$\lg N + 1$

$\frac{N(N+1)}{2}$

$\frac{N^3}{6} - \frac{N^2}{2} + \frac{N}{3}$

$\frac{N(N+1)}{2}$

$\frac{N^2}{2} - \frac{N}{2}$

$\lg N + 1$

Ordem de crescimento

Cúbico

Quadrático

Quadrático

Logaritmico

Maior hierarquia

Menor hierarquia

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$			

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`		

linearithmic	$N \log N$			

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`		

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`		

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`		

linearithmic	$N \log N$			

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`		

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`		

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$			

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`		

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`		

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$		*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`		

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`		

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$		*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`		

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$		*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`
			triple loop	*check all triples*

exponential	2^N			

description	order of growth	typical code framework	description	example
constant	1	`a = b + c;`	*statement*	*add two numbers*
logarithmic	$\log N$		*divide in half*	*binary search*

linear	N	`double max = a[0];`
		`for (int i = 1; i < N; i++)`
		`  if (a[i] > max) max = a[i];`
			loop	*find the maximum*

linearithmic	$N \log N$		*divide and conquer*	*mergesort*

quadratic	N^2	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  if (a[i] + a[j] == 0)`
		`  cnt++;`
			double loop	*check all pairs*

cubic	N^3	`for (int i = 0; i < N; i++)`
		`  for (int j = i+1; j < N; j++)`
		`  for (int k = j+1; k < N; k++)`
		`  if (a[i] + a[j] + a[k] == 0)`
		`  cnt++;`
			triple loop	*check all triples*

exponential	2^N		*exhasutive search*	*check all subsets*

(*) Fonte: http://algs4.cs.princeton.edu/14analysis/

ATIVIDADE 02: Ordem de crescimento

```
int G1 (int N) {
    int n, i, sum=0;
    for (n=N; n>0; n/=2)
        for (i=0; i<n; i++)
            sum++;
    return sum;
}
```

```
int G2 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for(j=0; j<i; j++)
            sum++;
    return sum;
}
```

```
int G3 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for (j=0; j<N; j++)
            sum++;
    return sum;
}
```

ATIVIDADE 02: Ordem de crescimento

```
int G1 (int N) {
    int n, i, sum=0;
    for (n=N; n>0; n/=2)
        for (i=0; i<n; i++)
            sum++;
    return sum;
}
```

Linear

G1(N) = 2N-1

```
int G2 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for(j=0; j<i; j++)
            sum++;
    return sum;
}
```

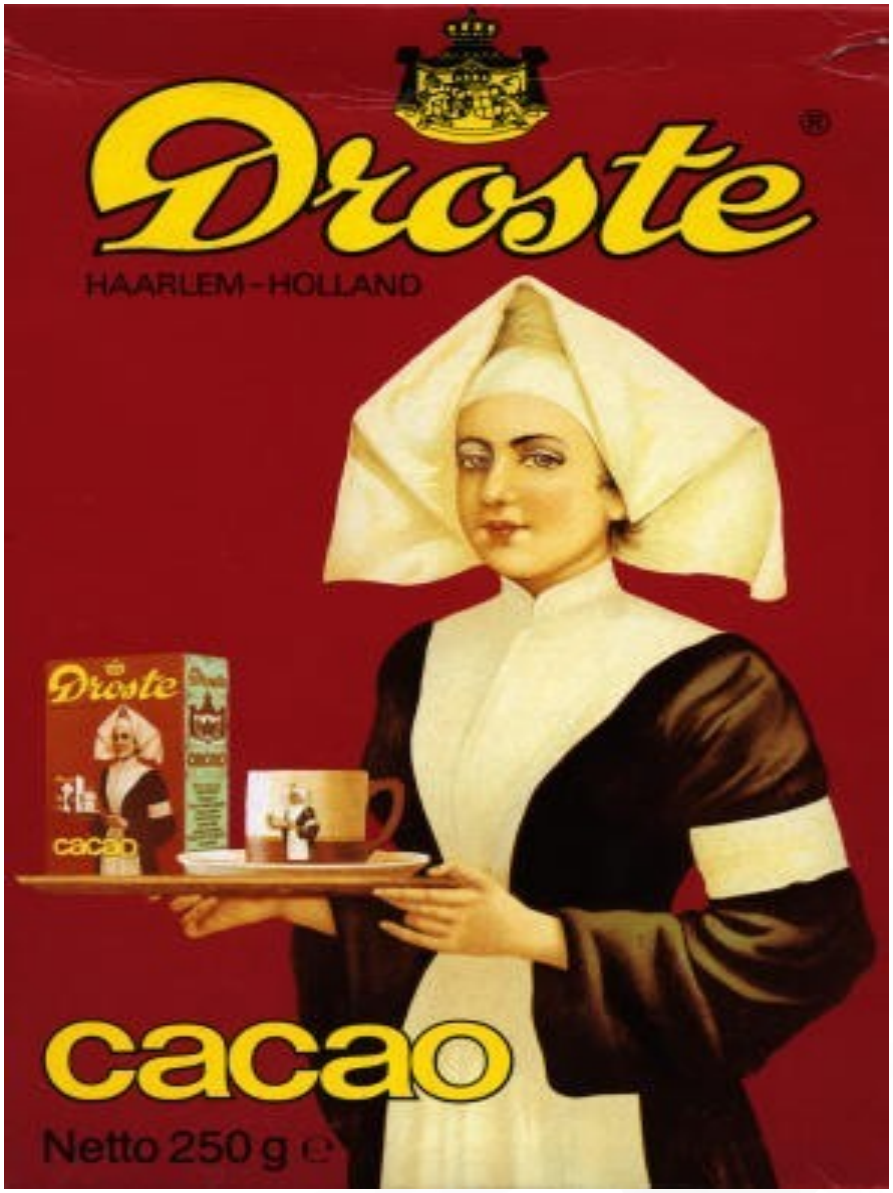
Linear

G2(N) = 2N-1

```
int G3 (int N) {
    int i, j, sum=0;
    for (i=1; i<=N; i*=2)
        for (j=0; j<N; j++)
            sum++;
    return sum;
}
```

Linearithmic

G3(N) = N(lg(N)+1)



Anuncio de cacao com uma imagem recursiva.

Recursão

Recursão

- O conceito de recursão é de fundamental importância em computação!

- Muitos problemas computacionais têm a seguinte propriedade:

  Cada instância do problema contém uma instância menor do mesmo problema.

  → Dizemos que esses problemas têm ***estrutura recursiva***.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Recursão

Para resolver um tal problema é natural aplicar o seguinte método:

- Se a instância em questão é pequena:
  → Resolva-a diretamente
  (use força bruta se necessário)
- Senão
  → Reduza-a a uma instância menor do mesmo problema
  → Aplique o método à instância menor
  e volte à instância original.

A aplicação do método produz um algoritmo recursivo.

(*) Fonte: P. Feofiloff. Algoritmos em Linguagem C. 1ª Edição, Editora Campos, 2008.

Números de Fibonacci

Números de Fibonacci

Números de Fibonacci

Quantas vezes é chamada
a função Fib para
n dado como entrada?

Números de Fibonacci

```
10
55
177
```

```
20
6765
21891
```

```
30
832040
2692537
```

```
40
102334155
331160281
```

Números de Fibonacci

(*) fonte http://britton.disted.camosun.bc.ca/fibslide/jbfibslide.htm

Números de Fibonacci

Números de Fibonacci

$$Fib(n) = O(2^n)$$