Análise e Projeto de Algoritmos

Prof. Eduardo Barrére

eduardo.barrere@ice.ufjf.br
www.barrere.ufjf.br

www.ufjf.br/pgcc
www.dcc.ufjf.br

Consumo de tempo assintótico

Seja *A* um algoritmo para um problema *P*. A quantidade de tempo que *A* consome para processar uma dada instância de *P* depende da máquina usada para executar *A*. Mas o efeito da máquina se resume a uma constante multiplicativa:

- se *A* consome tempo *t* numa determinada máquina, consumirá tempo 2*t* numa máquina duas vezes mais lenta e *t*/2 numa máquina duas vezes mais rápida.

Para eliminar o efeito da máquina, basta discutir o consumo de tempo de *A* sem as constantes multiplicativas.

A notação assintótica (O - ómicron, Ω - ómega, Θ - teta) é ideal para fazer isso.

Comparação assintótica de funções

- É desejável exprimir o consumo de tempo de um algoritmo de uma maneira que não dependa da linguagem de programação, nem dos detalhes de implementação, nem do computador empregado.
- Para tornar isto possível, é preciso introduzir um modo grosseiro de comparar funções. (dependência entre o consumo de tempo de um algoritmo e o tamanho de sua “entrada”)

Comparação assintótica de funções

- Essa comparação só leva em conta a “velocidade de crescimento” das funções. Assim, ela despreza fatores multiplicativos (pois a função $2n^2$, por exemplo, cresce tão rápido quanto $10n^2$) e despreza valores pequenos do argumento (a função n^2 cresce mais rápido que $100n$, embora n^2 seja menor que $100n$ quando n é pequeno).
- Dizemos que esta maneira de comparar funções é **assintótica**.
- Há três tipos de comparação assintótica: uma com sabor de “≥”, outra com sabor de “≤”, e uma terceira com sabor de “=”.

Análise assintótica: ordens O, Omega e Theta

Ao ver uma expressão como $n+10$ ou n^2+1, a maioria das pessoas pensa automaticamente em valores pequenos de n, próximos de zero. A análise de algoritmos faz exatamente o contrário: *ignora os valores pequenos* e concentra-se nos valores *enormes* de n. Para esses valores, as funções:

$$n^2 , \quad (3/2)n^2 , \quad 9999n^2 , \quad n^2/1000 , \quad n^2+100n , \quad \text{etc.}$$

crescem todas com a mesma velocidade e portanto são todas "equivalentes".

Esse tipo de matemática, interessado somente em valores *enormes* de n, é chamado *assintótica*. Nessa matemática, as funções são classificadas em "ordens"; todas as funções de uma mesma ordem são "equivalentes".

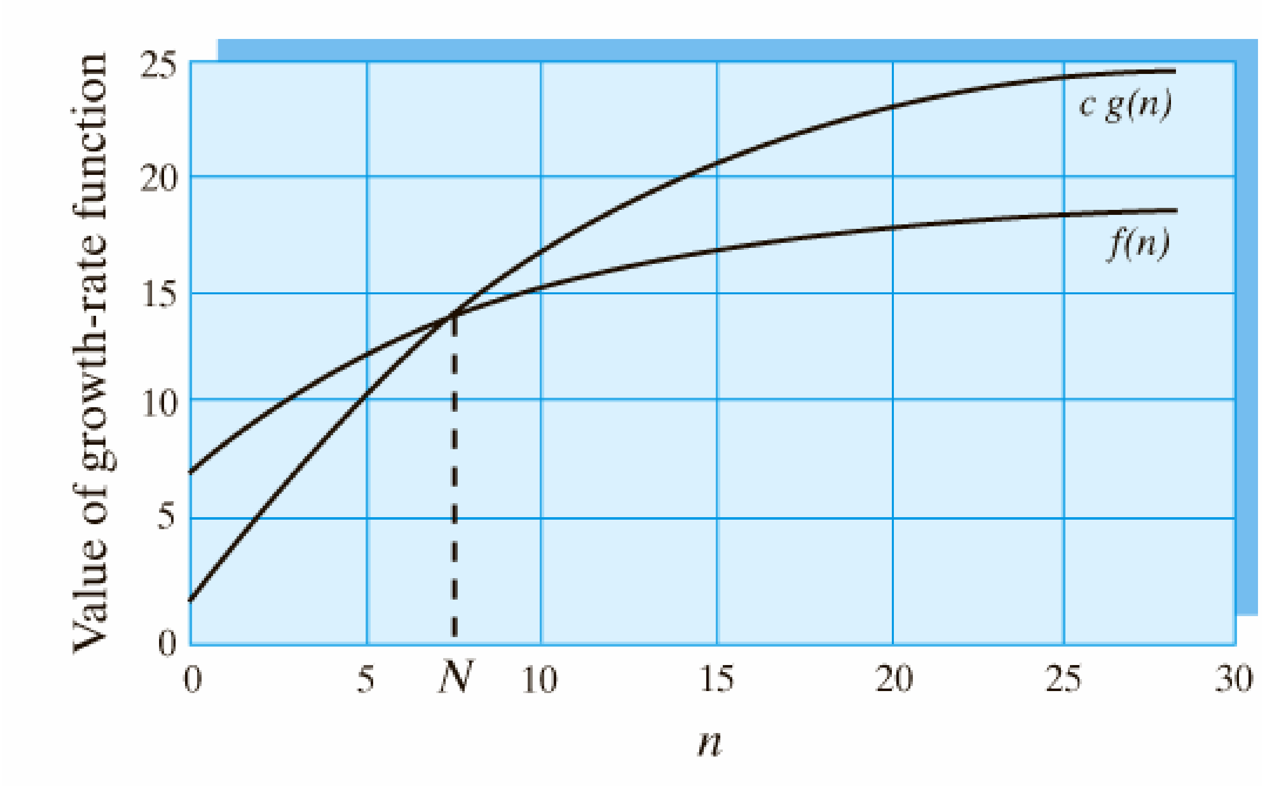
As cinco funções acima, por exemplo, pertencem à mesma ordem.

Notação O (*ômicron* grego maiúsculo) (big O)

Convém restringir a atenção a funções *assintoticamente não negativas*, ou seja, funções f tais que $f(n) \geq 0$ para todo n suficientemente grande. Mais explicitamente: f é assintoticamente não negativa se existe n_0 tal que $f(n) \geq 0$ para todo n maior que n_0.

Definição: Dadas funções assintoticamente não negativas f e g, dizemos que *f está na ordem O de g* e escrevemos $f = O(g)$ se $f(n) \leq c \cdot g(n)$ para *algum* c positivo e para *todo* n suficientemente grande. Em outras palavras, existe um número positivo c e um número n_0 tais que $f(n) \leq c \cdot g(n)$ para todo n maior que n_0.

Notação O: Diagrama



"F está em O(G)" tem sabor de "F ≤ G"

- tem sabor de "F não cresce mais que G"
- conceito sob medida para tratar de consumo de tempo de algoritmos

Notação O: Exemplo 1

- n2 +10n = O(n^2) *(f = O(g))* $f(n) \le c \cdot g(n)$

Prova: se $n \ge 10$ então $n^2 + 10n \le 2 * n^2$

• resumo: $n^2 + 10n \le 2n^2$ para todo $n \ge 10$

Como sabemos que 2 e 10 são bons valores para c e N?

• queremos: $n^2 + 10n \le c.n^2$

• dividindo por n^2, temos: $1 + 10/n \le c$

• se $n \ge 10$ então $1 + 10/n \le 2$

• parece que basta tomar $c \ge 2$ e $N \ge 10$

Notação O: Exemplo 2

- $9n+9 = O(n^2)$

Prova: para todo $n \geq 1$:

$$9n+9 \leq 18n^2$$

Notação O: Exemplo 3

- 100n está em $O(n^2)$

Prova: Para todo $n \geq 100$

$$
\begin{aligned}
100n &\leq n * n \\
&= n^2 \\
&= 1 * n^2
\end{aligned}
$$

Notação O: Exemplo 4

- $2n^3 + 100n$ está em $O(n^3)$

  Prova: Para todo $n \geq 1$

  $$2n^3 + 100n \leq 2n^3 + 100n^3$$
  $$\leq 102n^3$$

  Outra prova: Para todo $n \geq 100$

  $$2n^3 + 100n \leq 2n^3 + n * n * n$$
  $$\leq 3n^3$$

Notação O: Exemplo 5

- lg n está em O(n)

Prova: para todo $n \geq 1$

$$n \leq 2^n$$

$$\lg n \leq \lg 2^n$$

$$\lg n \leq n$$

Notação O: Exemplo 6

- $n^{1.5}$ está em $O(n^2)$

Prova: para todo $n \geq 1$

$$n^{1.5} \leq n^{0.5} * n^{1.5}$$

$$\leq n^2$$

Notação O: Exemplo 7

- Suponha que $f(n) = 2n^2 + 3n + 4$ e $g(n) = n^2$. Observe que:

$$2n^2 + 3n + 4 \;\leq\; 2n^2 + 3n^2 + 4n^2 \;=\; 9n^2$$

desde que $n \geq 1$.

- Resumindo, $f(n) \leq 9\, g(n)$ para todo ≥ 1.
- Portanto, $f(n) = O(g(n))$. $(f = O(g))$ $f(n) \;\leq\; c \cdot g(n)$

Notação O: Exemplo 8

- Suponha que $f(n) = 3 + 2/n$ e que $g(n) = n^0$, ou seja, g(n)=1. Então, para $n \geq 2$:

$$3 + 2/n \leq 3 + 1 = 4 = 4n^0$$

- Resumindo, $f(n) \leq 4\, g(n)$ para todo $n \geq 2$.

- Portanto, $f(n) = O(g(n))$.

Algumas Operações com a Notação O

$$
\begin{aligned}
f(n) &= O(f(n)) \\
c \times O(f(n)) &= O(f(n)) \qquad c = constante \\
O(f(n)) + O(f(n)) &= O(f(n)) \\
O(O(f(n)) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(max(f(n), g(n))) \\
O(f(n))O(g(n)) &= O(f(n)g(n)) \\
f(n)O(g(n)) &= O(f(n)g(n))
\end{aligned}
$$

Notação O: Exemplo 9

```
Procedure Verifica_Item(Lista: TipoLista; x: TipoItem; pos: integer);
Var i: integer;
Begin
    i:=1;                                                          O(1)
    achou := false;
    while (i <= Lista.Tamanho) and not achou do                    O(N)
        if Lista.Item[i] = x then
                achou := true;
    if achou then                                                  O(1)
        pos := i
    else
        for i:= Lista.Tamanho +1 to MaxTam;                        O(N)
                Lista.Item[i] := x;                                O(1)
```

$$f(N) = O(7 * O(1) + 2*O(N)) = O(O(1) + (O(N)) = O(N)$$

Notação Assintótica

Terminologia de classes mais comuns de funções:

- Logarítmica - O(log n)
- Linear - O(n)
- Quadrática - O(n^2)
- Polinomial – O(n^k), com k≥1
- Exponencial – O(a^n), com a>1

Função	Designação
c	Constante
log N	Logaritmo
$\log^2 N$	Logaritmo Quadrado
N	Linear
N log N	N logN
N^2	Quadrática
N^3	Cúbica
2^N	Exponencial

Ordens mais comuns

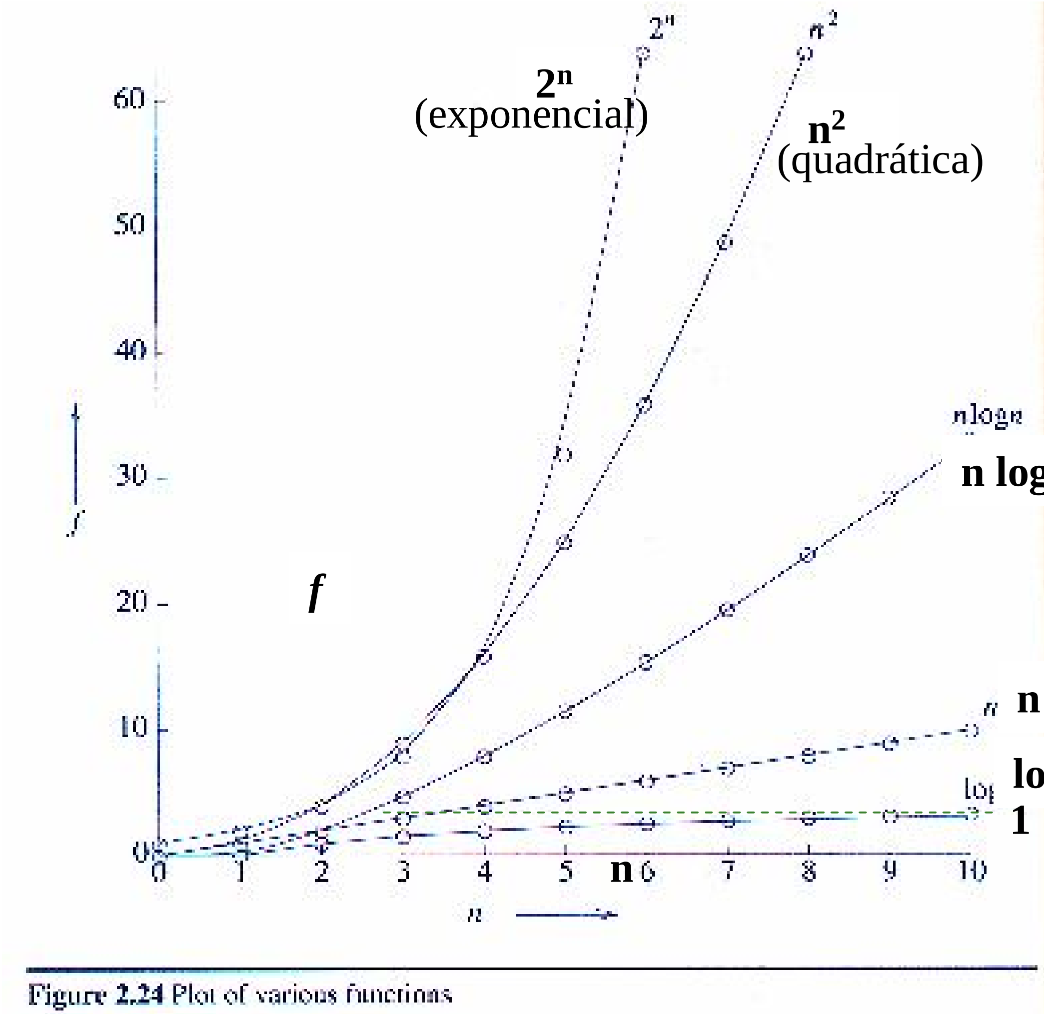


Figure 2.24 Plot of various functions

Fonte: Sahni, "Data Structures, Algorithms and Applications in C++"

Eficiência de um Algoritmo

Três algoritmos para calcular: 1 + 2 + … n para um n > 0

Algorithm A

```
sum = 0
for i = 1 to n
  sum = sum + i
```

Algorithm B

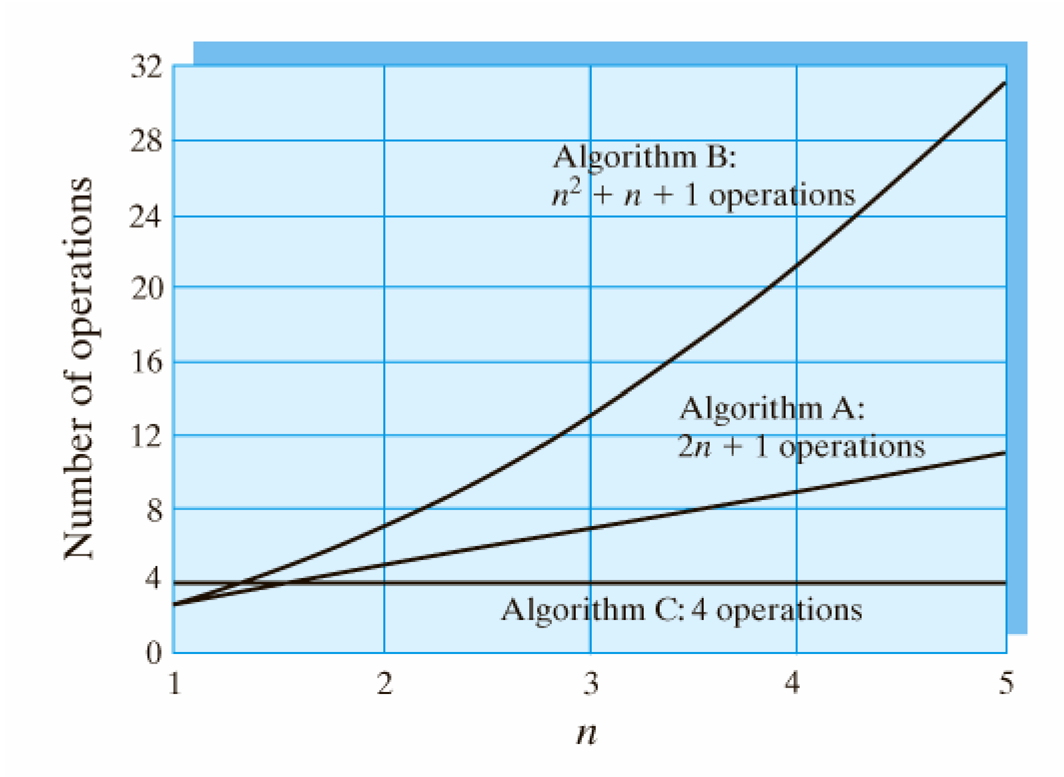
```
sum = 0
for i = 1 to n
{ for j = 1 to i
    sum = sum + 1
}
```

Algorithm C

```
sum = n * (n + 1) / 2
```

O(n)	$O(n^2)$	O(1)

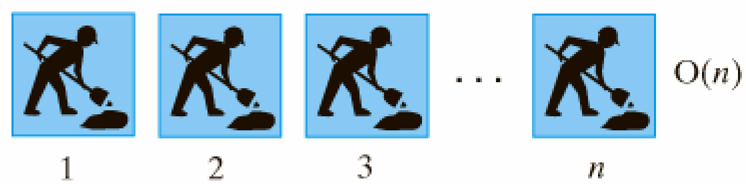
Eficiência de um Algoritmo



O número de operações em função de n

Eficácia

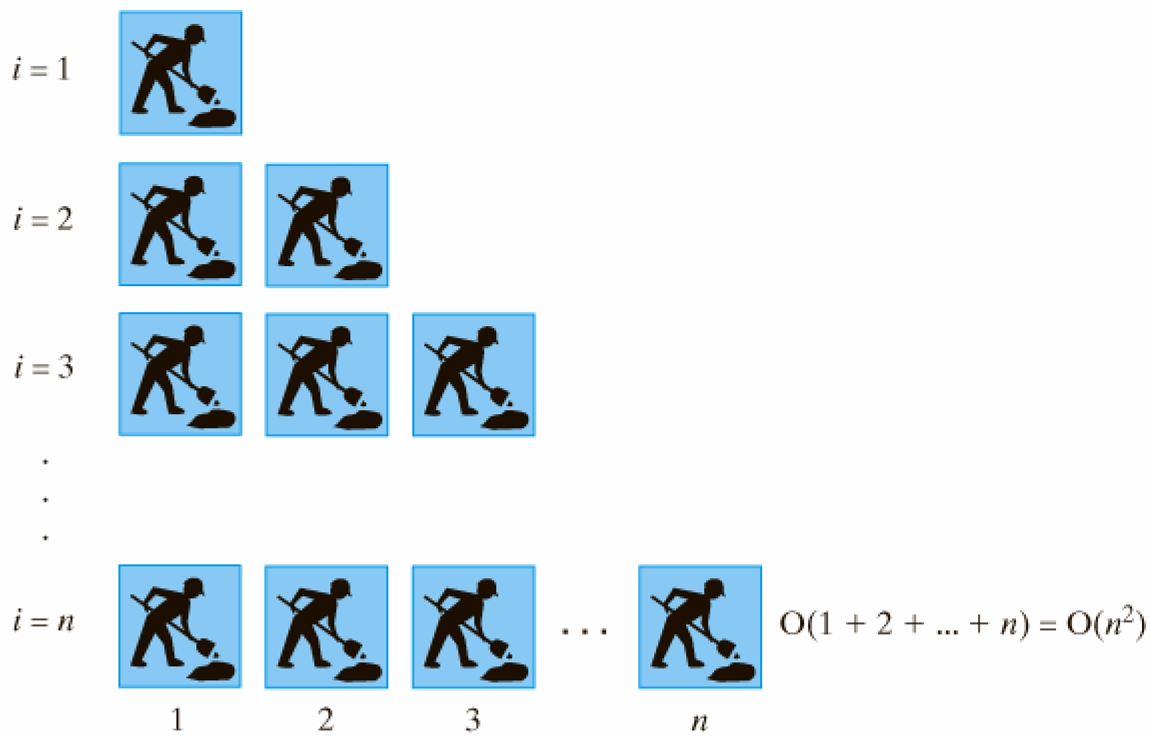
```
for i = 1 to n
   sum = sum + i
```



O(n)

Eficácia

```
for i = 1 to n
{   for j = 1 to i
        sum = sum + 1
}
```



$O(n^2)$

Notação Ω (ômega)

- Dadas funções F e G de $N^{>}$ em $R^{\geq}$, dizemos que F **está em Ω**(G) se existe c em $N^{>}$ tal que:

  $F(n) \geq c\, G(n)$, para todo n suficientemente grande.

- É claro que é Ω "inversa" de O, ou seja, uma função F está em Ω(G) se e somente se G está em O(F) (F=Ω(G) se e somente se G=O(F))

- Exemplo: $n^3 + 100n$ está em $\Omega(n^3)$, pois $n^3 + 100n \geq 1n^3$, para todo n ($n \geq 1$).

Notação Ω : Exemplos

- n^2 está em $\Omega(n)$, pois $n^2 \geq n$

- $n \lg n$ está em $\Omega(n)$. De fato, para todo $n \geq 2$ tem-se:

  $$\lg n \geq 1, \text{ portanto } n \lg n \geq n.$$

....

Notação θ (Teta)

- Dizemos que $f = \theta(g)$ se $f = O(g)$ e $f = \Omega(g)$

Provar que:

Para qualquer a em $R^{>}$ e qualquer b em R, a função $an^2 + bn$ está em $\theta(n^2)$.

Consumo de tempo de algoritmos

Seja A um algoritmo para um problema cujas instâncias têm tamanho n. Se a função que mede o consumo de tempo de A no pior caso está em $O(n^2)$, por exemplo, podemos usar expressões como:

"A consome $O(n^2)$ unidades de tempo no pior caso"

e "A consome tempo $O(n^2)$ no pior caso".

- Podemos usar expressões semelhantes com θ ou Ω no lugar de O e "melhor caso" no lugar de "pior caso".

Consumo de tempo de algoritmos

Algoritmo linear:

- consome θ(n) unidades de tempo no pior caso
- n multiplicado por 10 -> tempo multiplicado por 10
- algoritmos lineares são considerados muito rápidos

Algoritmo quadrático:

- consome θ(n^2) no pior caso
- n multiplicado por 10 -> tempo multiplicado por 100
- se o tamanho é multiplicado por uma constante c, o consumo é multiplicado por c^2.

Consumo de tempo de algoritmos

Algoritmo ene-log-ene:

- consome θ(n log n) unidades de tempo no pior caso
- n multiplicado por 2 -> tempo multiplicado por 2n
- se o tamanho é multiplicado por uma constante c, o consumo é multiplicado por c e acrescido de um pouco mais que cn.

Algoritmo polinomial:

- consome tempo $O(n^k)$ para algum k
- exemplos: $O(n)$, $O(n \lg n)$, $O(n^2)$, $O(n^{100})$
- algoritmos polinomiais são considerados rápidos.

Consumo de tempo de algoritmos

Algoritmo exponencial:

- consome tempo $\Omega(a^n)$ para algum $a > 1$
- exemplos: $\Omega(2^n)$, $\Omega(1.1^n)$
- n multiplicado por 10 -> tempo elevado a 10.