Complexidade de Algoritmos

Prof. Diego Buchinger
diego.buchinger@outlook.com
diego.buchinger@udesc.br

Prof. Cristiano Damiani Vasconcellos
cristiano.vasconcellos@udesc.br

Complexidade de Espaço, Notação Assintótica e Teorema Mestre

Complexidade de Espaço

Em alguns casos é importante considerarmos também a complexidade de espaço, i.e. o ‘espaço’ que é utilizado em função de ‘n’.

- Qual a complexidade dos algoritmos abaixo?

```
int n, f=1;
scanf("%i", &n);
for( ; n>0; n--)
   f = f * n;
return f;
```

```
int fatorial( int n ){
   if (n == 0)
      return 1;
   return n * fatorial( n-1 );
}
```

Complexidade de Espaço

- E qual a complexidade do algoritmo abaixo?

```
int fatorial( int n ){
   int a, b, c, d, e, f, g;
   if (n == 0)
      return 1;
   return n * fatorial( n-1 );
}
```

Complexidade de Espaço

- E qual a complexidade do algoritmo abaixo?

```
int fibonacci ( int n ){
    int i, fib[n+1];
    if( n<=1 )
        return 1;
    fib[0] = fib[1] = 1;
    for( i=2; i<=n; i++ )
        fib[i] = fib[i-1] + fib[i-2];
    return fib[n];
}
```

Complexidade de Espaço

- E deste algoritmo?

```
int pesqbin(int *v, int p, int r, int e){
   int q;
   if ( r < p )
       return -1;
   q = (p + r)/2;
   if (e == v[q])
       return q;
   if (e < v[q])
       return pesqbin(v, p, q-1, e);
   return pesqbin(v, q+1, r, e);
}
```

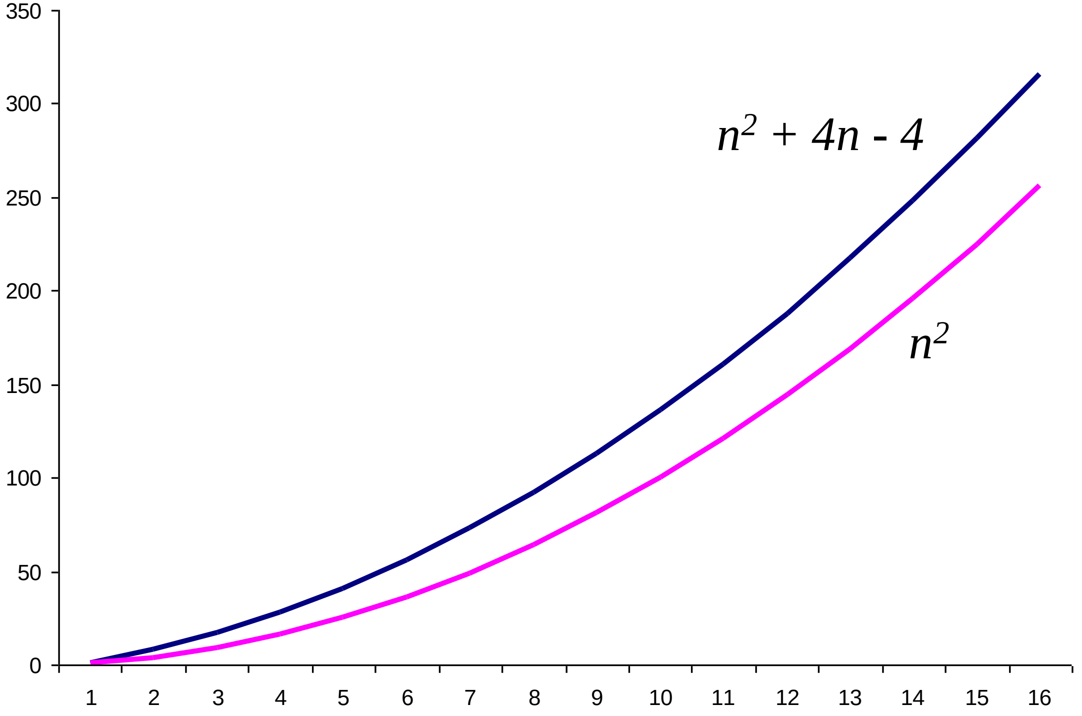
Complexidade de Espaço

- E deste outro?

```
/* Algoritmo muuuuito útil */
int foo( int n ){
    int v[n], i, s = 0;
    if ( n==0 ) return 0;
    for( i=0; i<n; i++ )
        v[i] = i*2;
    return  foo( n-1 );
}
```

Notação Assintótica
Limite Inferior (Notação Ω)

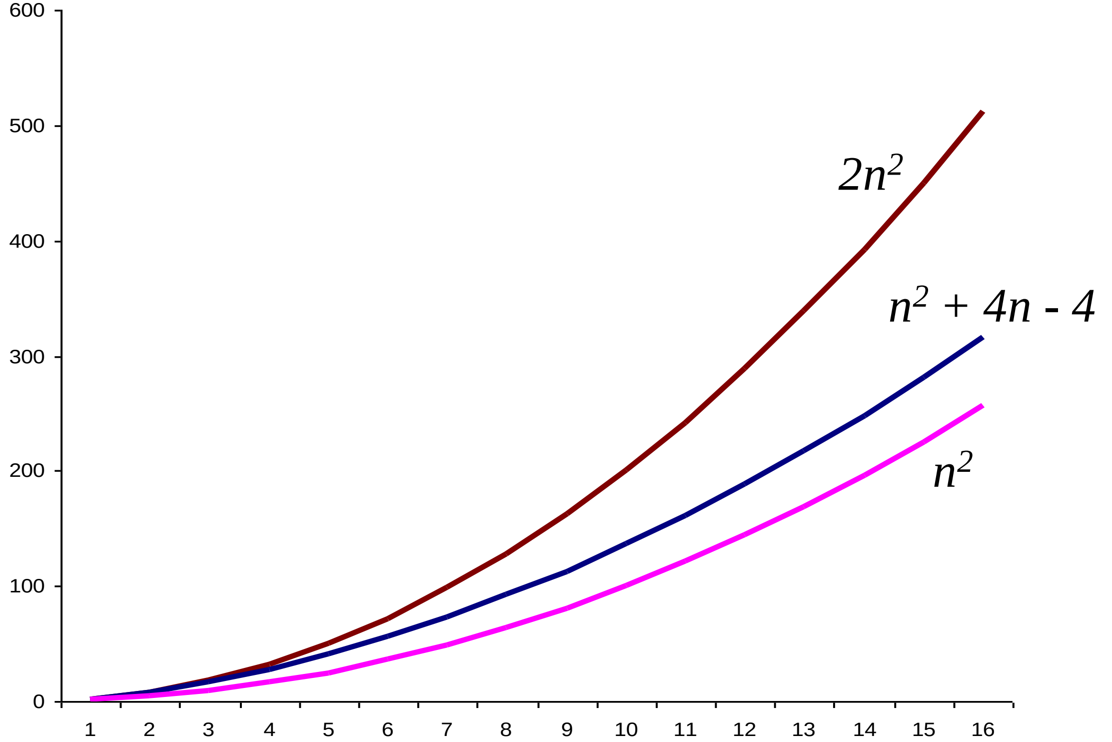
Uma função *f(n)* é o limite inferior de outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| \geq c.|f(n)|$, $g(n) = \Omega(f(n))$.



$$n^2 + 4n - 4 = \Omega(n^2)$$

Notação Assintótica
Limite Firme (Notação Θ)

Uma função $f(n)$ é o limite restrito (ou exato) de outra função $g(n)$ se existem três constantes positivas c_1, c_2, e n_0 tais que, para $n > n_0$, temos $c_1.|f(n)| \geq |g(n)| \geq c_2.|f(n)|$, $\quad g(n) = \Theta(f(n))$



[funções crescem com mesma rapidez]

$$n^2 + 4n - 4 = \Theta(n^2)$$

Notação Assintótica

Agora considere as funções f(x) = $n^{\in}$ (onde $\in > 0$ e $\in < 1$) e g(x) = $\log n$.

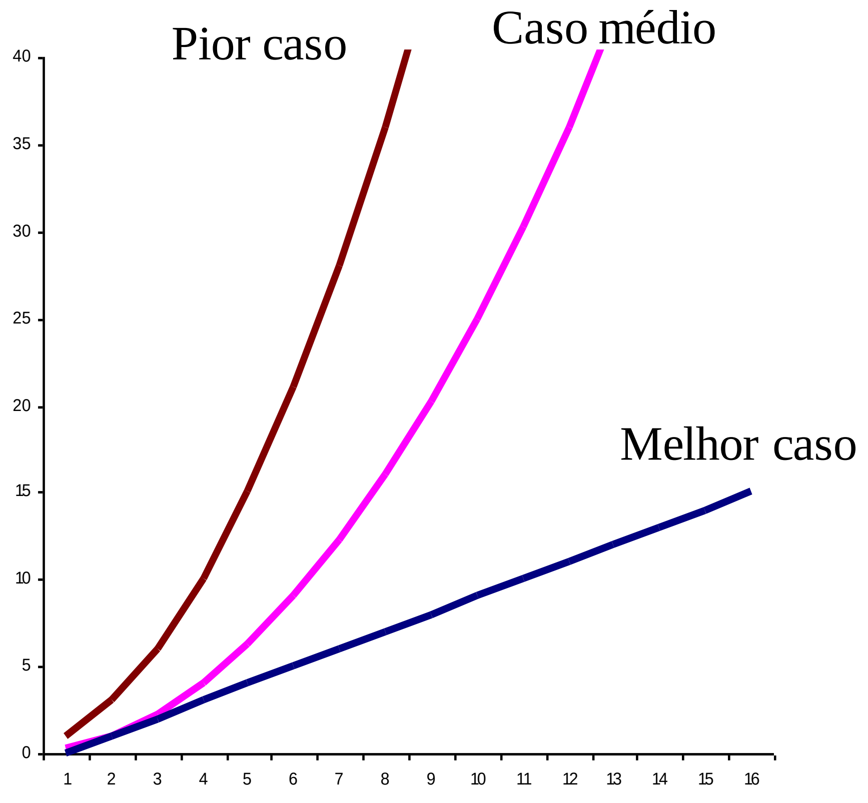
Qual a relação entre f(x) e g(x)?

$$f(x) = O(\ g(x)\ )$$

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

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

Ordenação por Inserção



Pior Caso: <u>$O(n^2)$</u>

Caso Médio: $n^2/4$

Melhor Caso: <u>$\Omega(n)$</u>.

Teorema Mestre

Livro de receitas para resolver recorrências da forma:

$$T(n) = a\,T(\,n / b\,) + f(n)$$

Sejam $a \geq 1$ e $b > 1$ constantes, seja f(n) uma função e seja T(n) definida sobre os inteiros não negativos pela recorrência acima, então T(n) pode ser limitado assintoticamente como:

Teorema Mestre

Caso 1:

$$Se$$

$$f(n) = \Theta\left(n^{\log_b a}\right)$$

então:

$$\boxed{T(n) = \Theta(n^{\log_b a} \times \log n)}$$

Exemplo: T(n) = T(2n / 3) + 1

Teorema Mestre

Caso 2:

$$Se$$

$$f(n) = O\,(n^{\log_b a - \epsilon})$$

$$para\ alguma\ constante\ \epsilon > 0, \text{ então:}$$

$$T(n) = \Theta(n^{\log_b a})$$

Exemplo: T(n) = 9T(n / 3) + n

Teorema Mestre

Caso 3:

$$Se$$

$$f(n) = \Omega\left(n^{\log_b a+\epsilon}\right)$$

para alguma constante $\epsilon > 0$, e se
$af(n / b) \leq cf(n)$ para alguma constante $c < 1$ e para
todo n suficientemente grande, então:

$$\boxed{T(n) = \Theta(f(n))}$$

Exemplo: $T(n) = 3T(n / 4) + n \log n$

Teorema Mestre

Casos especiais:

Se as condições do caso 1, 2 ou 3 não forem satisfeitas então não é possível resolver a recorrência usando o teorema mestre!

Exemplo: $T(n) = 2T(n / 2) + n \log n$

Atividade

Considere o algoritmo abaixo e descreva a sua relação de recorrência, sua complexidade de tempo e espaço para o melhor e pior caso:

```
double foo( int* v, int n, int p ){
    if( n<=0 )
        return 0;
    int i, soma = 0;
    for( int i=0; i<n ; i=i+p )
        soma += v[i];
    return sqrt(soma) + foo( v, soma%n, p );
}
```

Atividade

Considere o algoritmo abaixo e descreva a sua relação de recorrência, sua complexidade de tempo e espaço para o melhor e pior caso:

```
int pow(int base, int exp){
    if( exp==0 ) return 1;
    int ret = pow( base, exp/2 );
    ret = ret * ret;
    if( exp%2 == 1 ) ret = ret * base;
    return ret;
}
```

Atividade

Use o método mestre para fornecer limites assintóticos restritos para as seguintes recorrências:

a. $T(n) = T(n/2) + \Theta(1)$

b. $T(n) = 4T(n/2) + \Theta(n)$

c. $T(n) = 4T(n/2) + \Theta(n^3)$

Resolva as seguintes recorrência por substituição:

d. $T(n) = T(n-1) + \Theta(\log n)$

$T(1) = \Theta(\log 1)$

e. $T(n) = T(\sqrt{n}) + \Theta(1)$

$T(2) = \Theta(1)$

Referências

Algoritmos. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliford Stein. Campus.

Algorithms. Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani. McGraw Hill.

Concrete Mathematics: A Foundation for Computer Science (2nd Edition).
Ronald L. Graham, Donald E. Knuth, Oren Patashnik. Addison Wesley.

M. R. Garey and D. S. Johnson. 1978. *"Strong" NP-Completeness Results: Motivation, Examples, and Implications*. J. ACM 25, 3 (July 1978)