UDESC

Complexidade de Algoritmos

Prof. Rafael Alceste Berri
rafaelberri@gmail.com

Prof. Diego Buchinger
diego.buchinger@outlook.com

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

UDESC

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 );
}
```

UDESC

Complexidade de Espaço

- Um programa requer uma área para guardar suas instruções, suas constantes, suas variáveis e os dados.
  - **Desconsiderados**: Dados têm uma *representação natural* (ex: matriz); O(1)
  - **Considerados**: *espaço extra*, espaço utilizado para guardar as instruções do programa e os dados. Dados irregulares, como grafos, são levados em conta.
- Em outras palavras: “Quanto cresce em função de **n**?”

UDESC

Entendendo o espaço

- **Parte Fixa**: o espaço requerido para armazenar dados e variáveis que são independentes do tamanho do problema (n):
  - Mesmo espaço ocupado para classificar 2GB ou 1MB de texto;
  - O(1)
- **Parte Variável**: espaço necessário para variáveis que dependem do tamanho do problema (n) :
  - Importar 2GB de texto ou Importar 1MB de texto…

UDESC

Complexidade de Espaço

- Exemplos:
  - Entrada(int *v, int n)
    - O(n) (alguns não consideram se for por referência)
  - Variáveis int i, j, x;
    - O(1) → independente de n

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?

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

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

Complexidade de Espaço

- E qual a complexidade de espaço deste algoritmo?

```
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 qual a complexidade de espaço deste algoritmo?

```
// O(n)
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 qual a complexidade de espaço deste algoritmo?

```
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 de espaço deste algoritmo?

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

UDESC

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 algoritmo?

```
// O(n log n) (por referência o vetor pode ser O(logn)
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 );
}
```

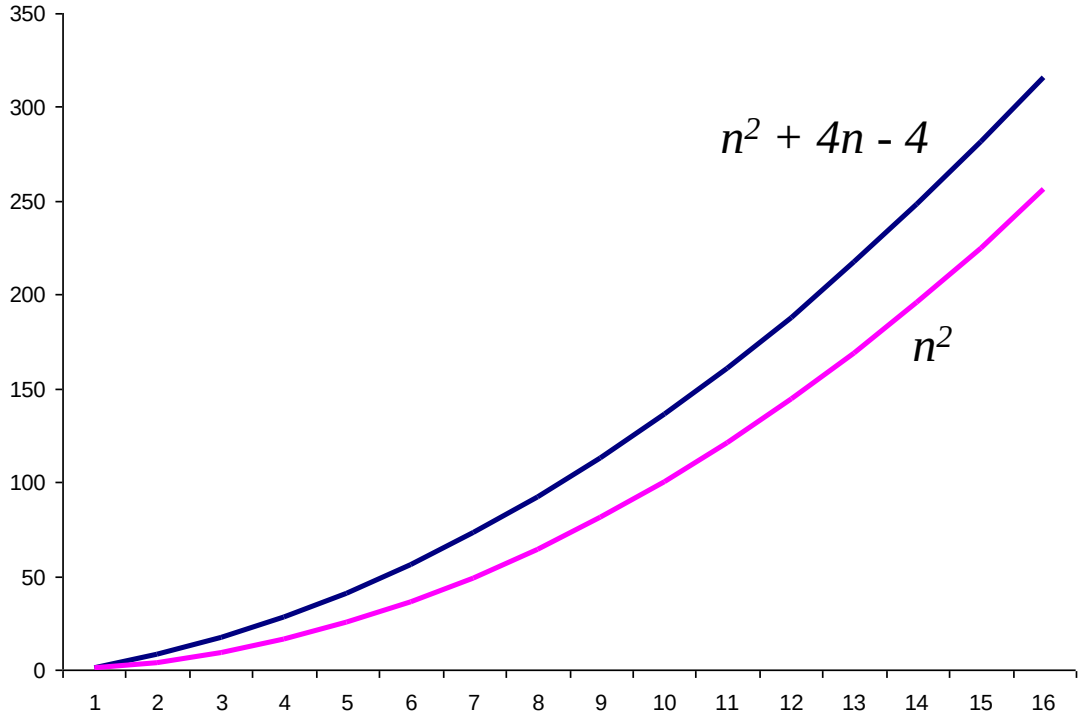
Complexidade de Espaço

- E deste outro?

```
// O(n²)
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)

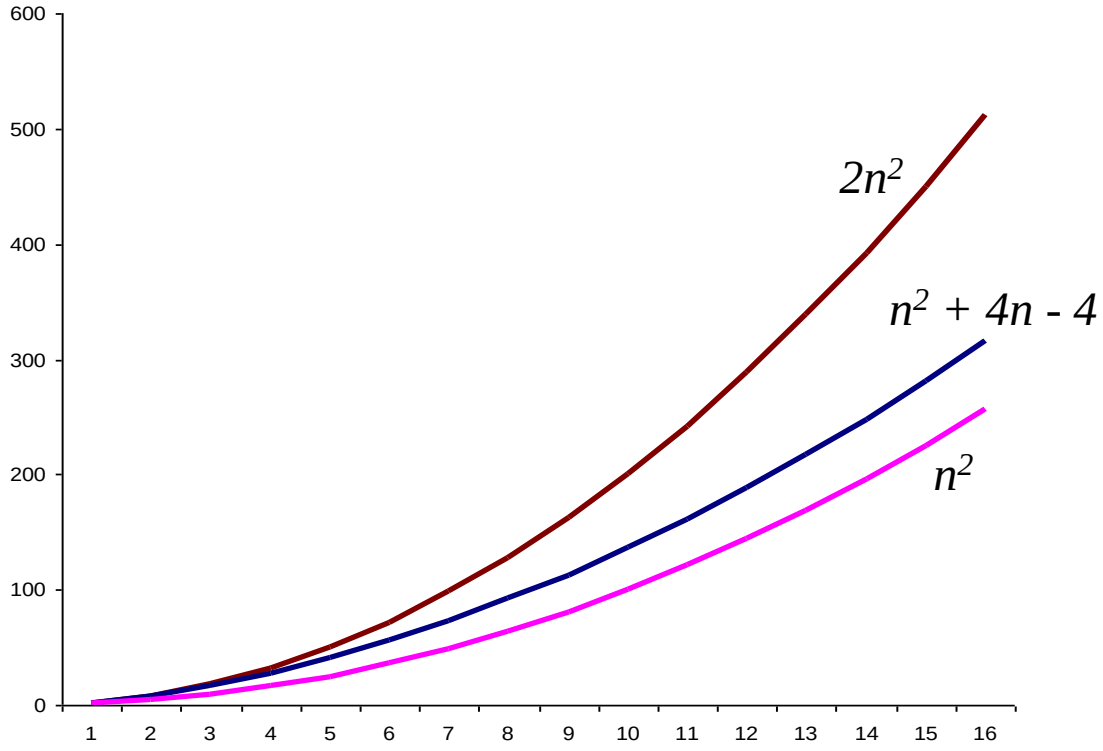
Uma função *f(n)* é o limite inferior (ômega) de outra função *g(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|g(n)| \geq c.|f(n)|$, $g(n) = \Omega(f(n))$.



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

Notação Assintótica
Limite Firme (Notação Θ)

Uma função *f(n)* é o limite restrito (teta) (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 $c_1.|f(n)| \geq |g(n)| \geq c_2.|f(n)|$, $g(n) = \Theta(f(n))$



[funções crescem com mesma rapidez]

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

Notação Assintótica

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

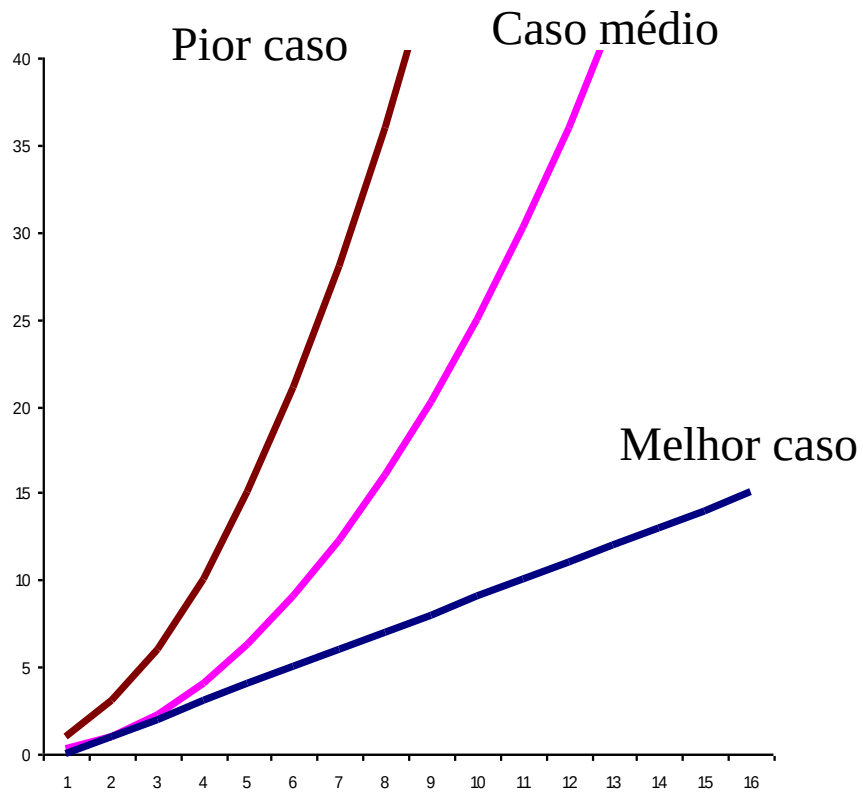
Qual a relação entre f(x) e g(x)?

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

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

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

Ordenação por Inserção



Pior Caso: $\underline{O(n^2)}$

Caso Médio: $n^2/4$

Melhor Caso: $\underline{\Omega(n)}$.

UDESC

Teorema Mestre

Livro de receitas para resolver recorrências da forma:

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

peso do número de chamadas recursivas
vs.
peso de cada chamada recursiva individual

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:

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

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

UDESC

Teorema Mestre

Caso 2:

Se

$$f(n) = \mathrm{O}\left(n^{\log_b a - \epsilon}\right)$$

para alguma constante $\epsilon > 0$, 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:

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

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

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

Atividade 5

1) 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;
}
```

Envie os exercícios para rafaelberri@gmail.com Assunto: "TC-CAL05"
Anexar: resultados/respostas

Atividade 5

2) 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 );
}
```

Quais são os possíveis resultados para uma operação de resto de divisão? (Pense para um número fixo – ex: x%7)