Complexidade de Algoritmos

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

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

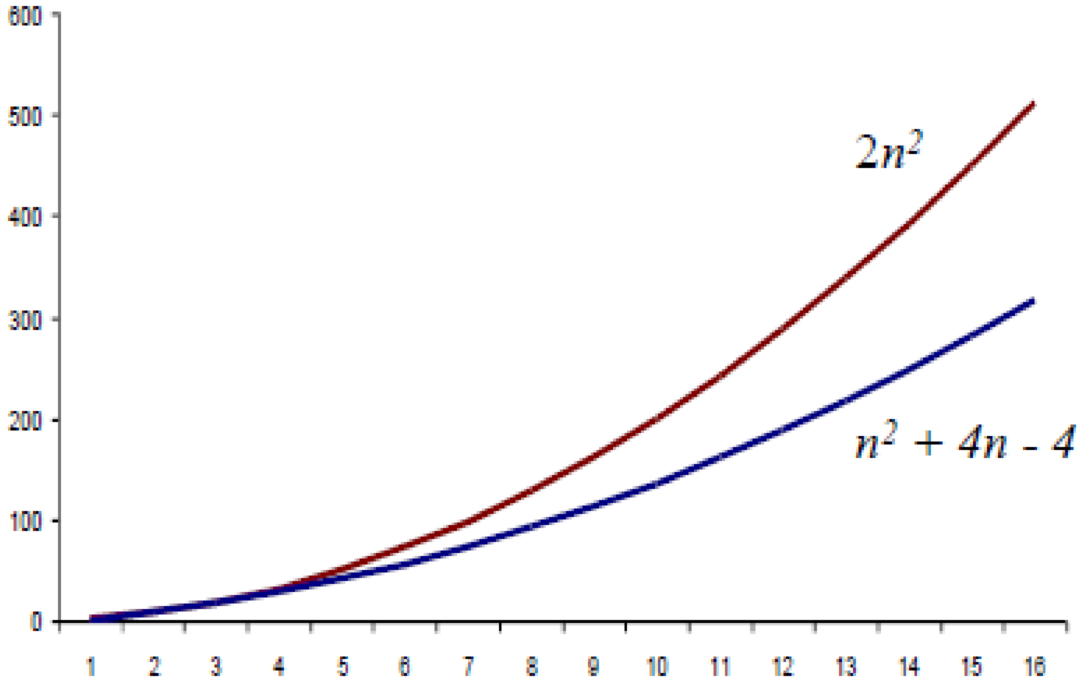
Funções de Complexidade

Considere que cada operação leva 1ns em média em um determinado processador. Determine o tempo das funções abaixo para os seguintes valores de operações:

$f(n) / n$	$n=10$	$n=100$	$n=1.000$	$n=10.000$	$n=100.000$	$n=1.000.000$
$\log_2 n$						
n						
$3n$						
$n \log_2 n$						
n^2						
2^n						
$n!$						

Notação Assintótica
(Notação O grande – Limite Superior)

Uma função *g(n)* domina assintoticamente outra função *f(n)* se existem duas constantes positivas *c* e n_0 tais que, para $n > n_0$, temos $|f(n)| \leq c.|g(n)|$ → $f(n) = O(g(n))$



$n^2 + 4n - 4 = O(n^2)$

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

Notação Assintótica
(Notação O grande – Limite Superior)

- $f(n) = n^2 + 4n - 4$ g(n) = $O(n^2)$
- $f(n) = 2n^2$ g(n) = $O(n^2)$ [O(n) ?]

$f(n) / c$ & n_0	$c=1$ $n_0=3$	$c=1$ $n_0=50$	$c=2$ $n_0=1$	$c=2$ $n_0=10$	$c=3$ $n_0=2$	$c=3$ $n_0=10$
$2n^2$					-	-
$c(n^2)$						
$n^2 + 4n - 4$						
$c(n)$						

Algumas Operações com Notação O

$c.O(f(n)) \; = \; O(f(n))$, onde c é uma constante.

$O(\ f(n)\ ) + O(\ g(n)\ ) = O(\ MAX(f(n), g(n)\ )$

$n.O(\ f(n)\ ) = \mathrm{O}(\ n\ .\ f(n)\ )$

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

Hierarquia de funções

Hierarquia de funções do ponto de vista assintótico:

$$1 \;<\; \log\log n \;<\; \log n \;<\; n^{\varepsilon} \;<\; n^{c} \;<\; n^{\log n} \;<\; c^{n}$$

onde ε e c são constantes arbitrárias tais que $0 < \varepsilon < 1 \leq c$.

Medidas de Complexidade

CONSIDERAÇÃO II: Ignorar o custo das instruções (tempo constante) e focar na análise do crescimento do uso de um recurso (tempo, espaço) em relação ao crescimento da entrada.

Ex: ordenar uma lista de 'n' elementos e mostrar a lista ordenada

n	Ordenação Bolha	printf vetor
100	37,8 µs	8,532 ms
200	148,4 µs	17,847 ms
1.000	3,748 ms	91,569 ms
10.000	247 ms	860,205 ms
50.000	5,307 s	4,277 s
100.000	20,422 s	8,693 s

Medidas de Complexidade

CONSIDERAÇÃO III: pode-se analisar os valores de entrada com perspectivas diferentes:

- **Melhor caso** => menor complexidade para um valor de ‘n’;
- **Pior caso** => maior complexidade para um valor de ‘n’;
- **Complexidade esperada ou média** => leva-se em conta a probabilidade de ocorrência de cada entrada de um mesmo tamanho ‘n’.

Pode-se antecipar alguma relação ($<$, $\leq$, $>$, $\geq$) entre as complexidades média e pior caso de um algoritmo qualquer?

Medidas de Complexidade

```
int pesquisa(Estrutura *v, int n, int chave) {
    int i;
    for (i = 0; i < n; i++)
        if (v[i].chave == chave)
            return i;
    return -1;
}
```

Em que situação ocorre o melhor caso?

Em que situação ocorre o pior caso? E o caso médio?

ATENÇÃO: não assuma um valor fixo e pequeno para *n* ao considerar o melhor caso!

Medidas de Complexidade

Melhor caso: Caso o primeiro registro seja o registro procurado será necessária apenas uma comparação.

Logo, podemos dizer que a complexidade é **constante**

[OBS: existe uma notação especial para indicar melhor caso – veremos ela mais adiante]

Medidas de Complexidade

Pior caso: Caso o último registro acessado seja aquele que se procura:

```
int pesquisa(Estrutura *v, int n, int chave) {
    int i;                                  // O(1)
    for (i = 0; i < n; i++)                 // O(n)
        if (v[i].chave == chave)            // O(1)
            return i;                       // O(1)
    return -1;                              // O(1)
}
```

Logo, podemos dizer que a função pesquisa tem complexidade **O(n)** para o pior caso.

Medidas de Complexidade

Caso médio: Caso o *i-ésimo* registro seja o registro procurado são necessárias i comparações. Sendo p_i a probabilidade de procurarmos o *i-ésimo* registro temos:

$$f(n) = 1.p_1 + 2.p_2 + \ldots + n.p_n.$$

Considerando que a probabilidade de procurar qualquer registro é a mesma probabilidade, temos:

$$p_i = 1 / n \quad \text{para todo } i.$$

$$f(n) = \frac{1}{n}(1 + 2 + \ldots + n) = \frac{1}{n}\left(\frac{n(n+1)}{2}\right) = \frac{(n+1)}{2}$$

Logo, temos uma complexidade linear: $\frac{(n+1)}{2} = (\boldsymbol{n})$

Medidas de Complexidade

CONSIDERAÇÃO IV: pode-se analisar a complexidade em relação a diferentes recursos. Os mais usuais são: tempo e espaço.

Complexidade de espaço:

Devemos considerar todo o espaço adicional criado pelo algoritmo assim como a quantidade de chamadas de função (geralmente recursivas) realizadas.

Por enquanto vamos focar no recurso “tempo”!

Exemplo (Bubble Sort)

```
void bubble(int *v, int n){
    int i, j, aux;
    for (i = n - 1; i > 0; i--){
        for (j = 0; j < i; j++){
            if (v[j] > v[j+1]){
                aux = v[j];
                v[j] = v[j+1];
                v[j+1] = aux;
            }
        }
    }
}
```

Exemplo
(Ordenação por Seleção)

```
void selectionSort(int *v, int n){
    int i, j, x, aux;
    for (i = 0; i < n; i++){
        x = i;
        for (j = i+1; j < n; j++){
            if( v[j] < v[x] )
                x = j;
        }
        aux = v[i];
        v[i] = v[x];
        v[x] = aux;
    }
}
```

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Exemplo
(Ordenação por Inserção)

```c
void insercao(int *v, int n){
    int i, j, x;
    for (i = 1; i < n; i++){
        x = v[i];
        j = i - 1;
        while (j >= 0 && v[j] > x){
            v[j+1] = v[j];
            j--;
        }
        v[j+1] = x;
    }
}
```

6 5 3 1 8 7 2 4

Atividade

- Implemente as funções de ordenação analisadas e faça um quadro comparativo do tempo de execução para ordenar:
  - (n=)100.000 números aleatórios

    (OBS: utilize a mesma entrada para cada algoritmo)

  - (n=)100.000 números já em ordem crescente

    (OBS: pode ser uma sequencia simples como 1,2,3, ..., 100.000)

Atividade

Elabore os seguintes algoritmos e analise o seu tempo de execução para diferentes entradas e determine a sua complexidade de tempo.

- Implemente um algoritmo (função) que recebe como parâmetro dois valores inteiros a e b e calcula a^b.
- Implemente um algoritmo (função) que recebe duas matrizes quadradas de mesma ordem (n x n) e realiza a multiplicação entre elas.

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)