UDESC

Projeto e Análise de Algoritmos

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

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

UDESC

Conceitos Básicos de Complexidade

Medidas de Complexidade

- Como calcular a quantidade de trabalho requerido por um algoritmo, ou seja, sua complexidade?

Medidas de Complexidade

- Como calcular a quantidade de trabalho requerido por um algoritmo, ou seja, sua complexidade?
  - Depende do tamanho da entrada;
  - Depende dos valores da entrada;

Ex: ordenação de uma lista de '*n*' elementos:

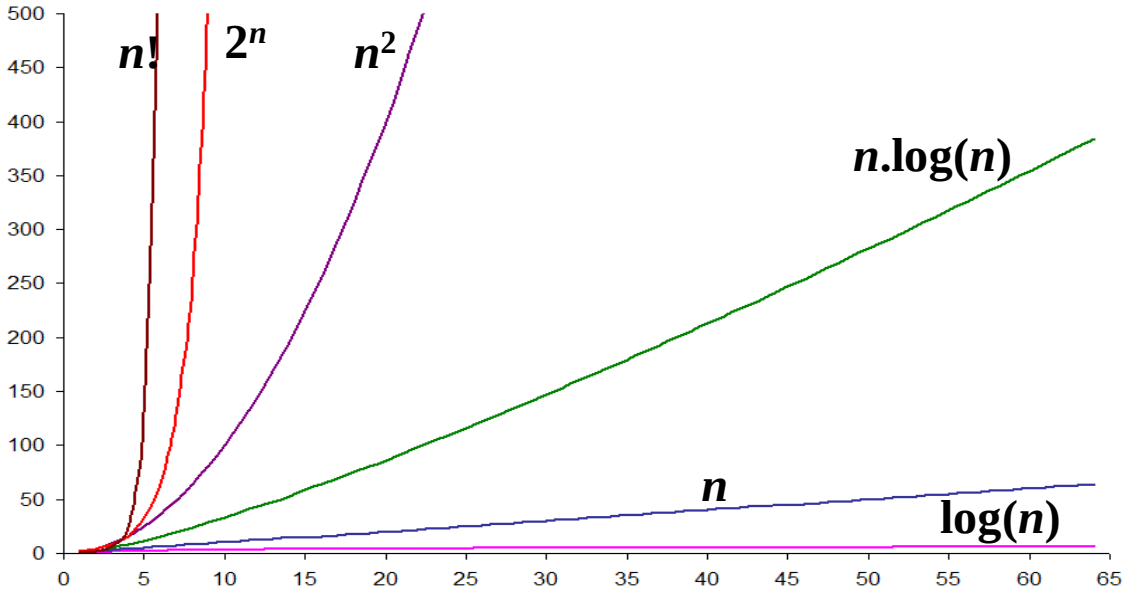
lista com elementos já ordenados

vs

lista com elementos totalmente desordenados

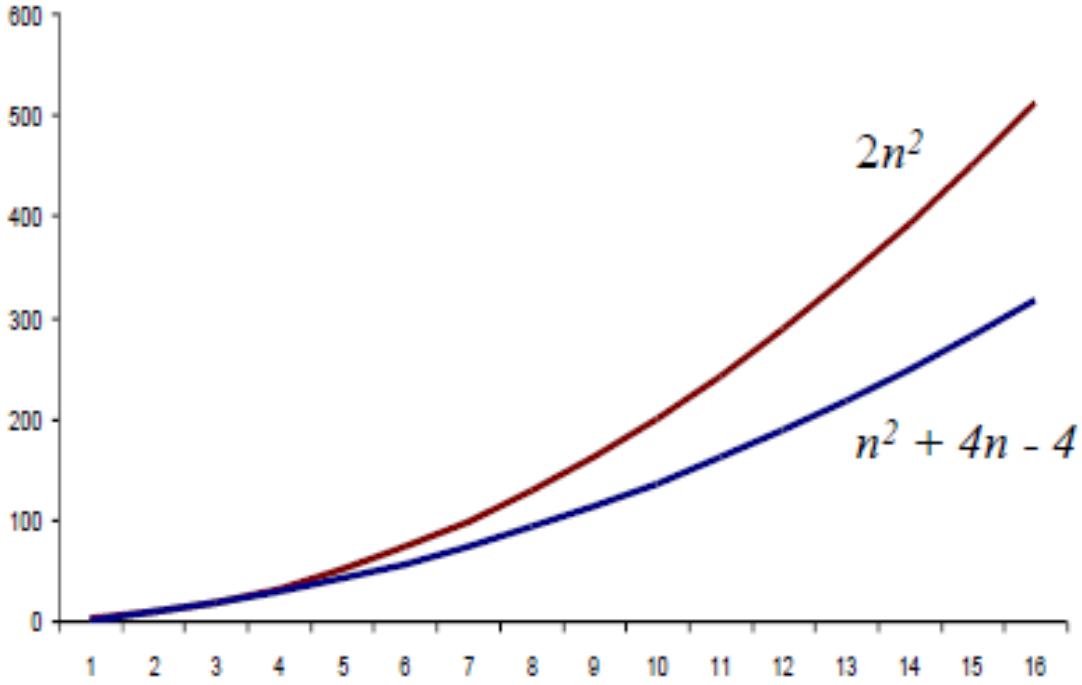
Medidas de Complexidade

CONSIDERAÇÃO I: trabalhar com valores grandes para ‘n’ (entrada). Assim, ordens de crescimento são destacadas.



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 $|f(n)| \leq c.|g(n)|$ → $f(n) = O(g(n))$



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

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

UDESC

Algumas Operações com Notação *O*

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

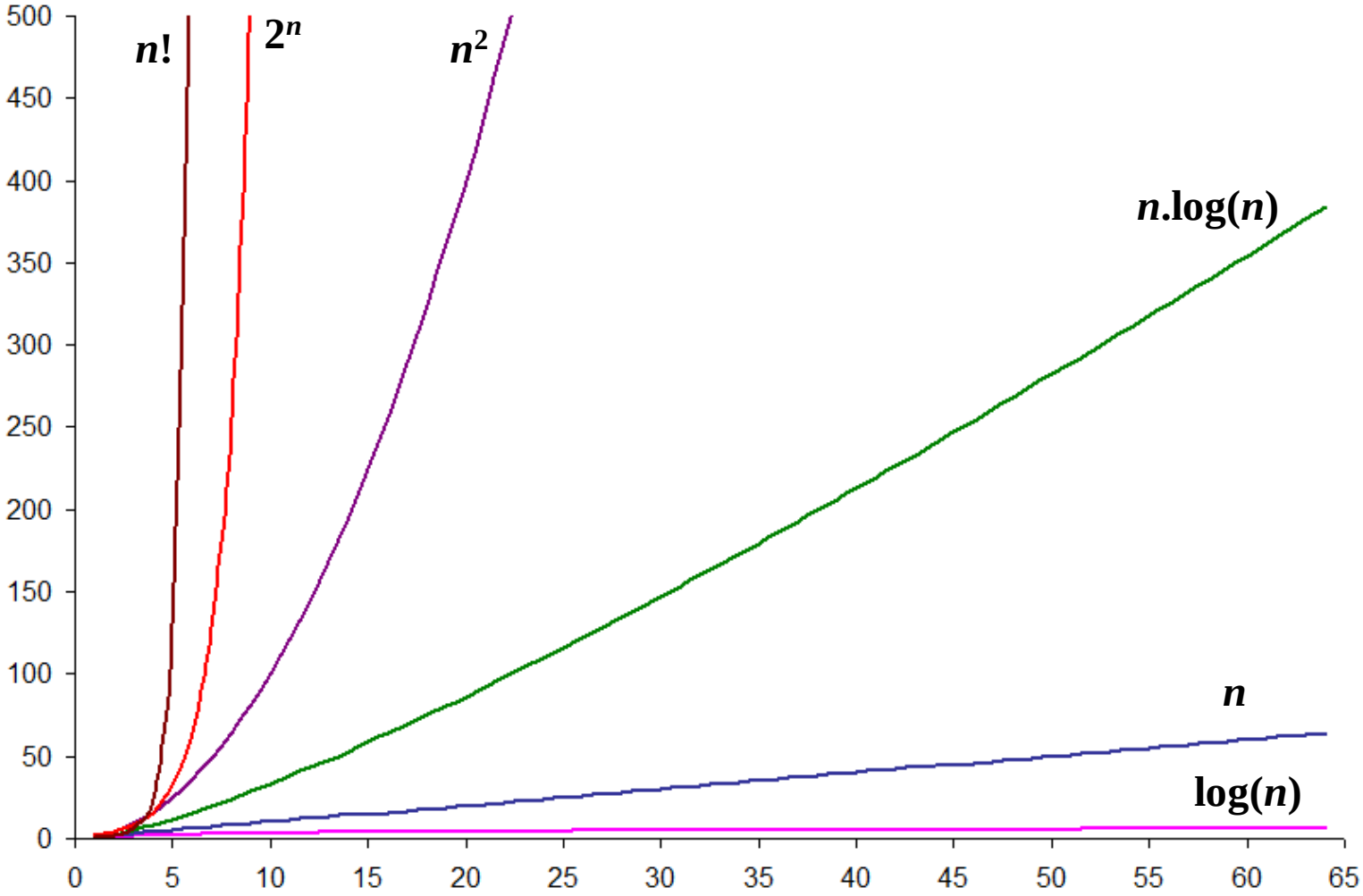
$O(f(n)) + O(g(n)) = O(MAX(f(n), g(n))$

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

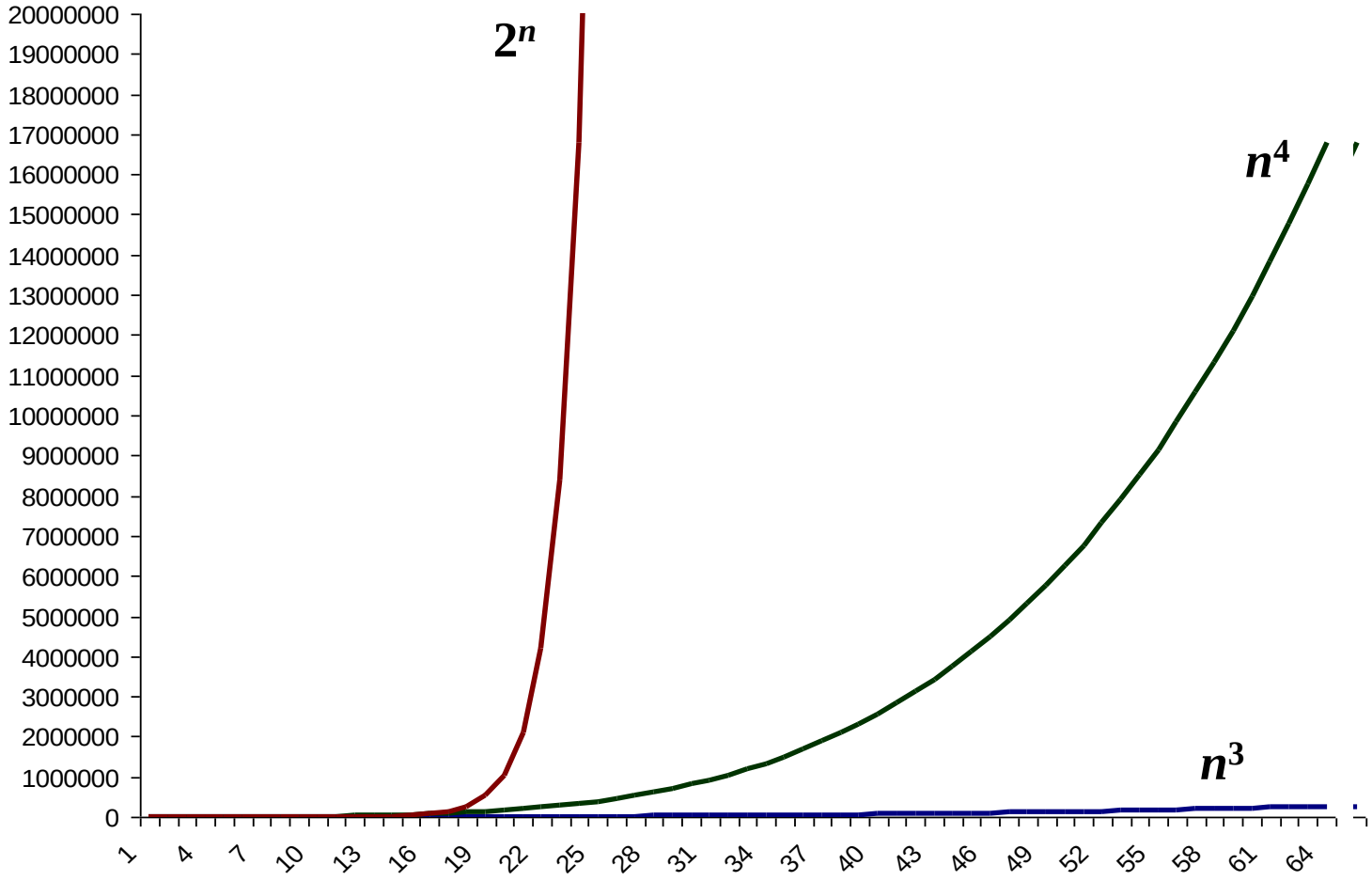
$O(f(n)).O(g(n)) = O(f(n).g(n))$

UDESC

Crescimento de Funções



Crescimento de Funções



UDESC

Hierarquia de funções

Hierarquia de funções do ponto de vista assintótico:

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

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

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 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?

UDESC

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**:

$$O(1)$$

(o correto seria usar outra letra grega para
o melhor caso mas, vamos por partes)

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:

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

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

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

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

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

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:

- Se os dados possuem representação natural, (ex. matriz) considera-se apenas o espaço extra utilizado pelo algoritmo;
- Se os dados podem ser representados de várias formas (ex. grafo) deve-se considerar o espaço utilizado por sua representação (matriz ou lista encadeada).

UDESC

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;
            }
        }
    }
}
```

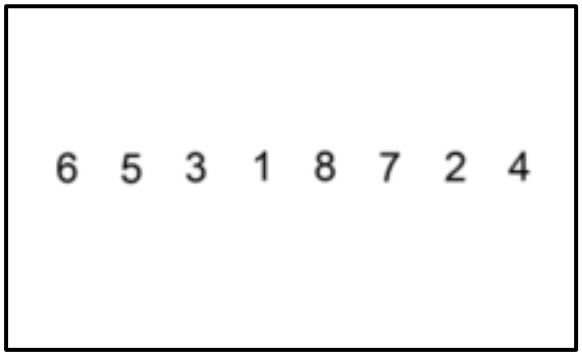
Exemplo (Ordenação por Seleção)

```
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)

```
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;
    }
}
```



Atividade

Elabore os seguintes algoritmos e analise o seu tempo de execução para diferentes entradas e 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) e realiza a multiplicação entre elas.

UDESC

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)