

FCT/Unesp – Presidente Prudente
Departamento de Matemática e Computação

Análise de Algoritmos de Ordenação

Parte 1

Prof. Danilo Medeiros Eler
danilo.eler@unesp.br

Apresentação adaptada (ver referências)

Introdução

- Ordenação (ou classificação) é largamente utilizada
 - Listas telefônicas e dicionários
 - Grandes sistemas de BD e processamento de dados
 - Algoritmos de ordenação são ilustrativos
 - Como resolver problemas computacionais
 - Como desenvolver algoritmos elegantes e como analisar e comparar seus desempenhos

Introdução

- Ordenar (ou classificar)
 - *Definição: organizar uma seqüência de elementos de modo que os mesmos estabeleçam alguma relação de ordem*
 - *Diz-se que os elementos k_1, \dots, k_n estarão dispostos de modo que $k_1 \leq k_2 \leq \dots \leq k_n$*
 - Facilita a busca/localização/recuperação de um elemento dentro do conjunto a que pertence
 - Será?

Introdução

- Existem vários meios de implementar ordenação
- Dependendo do problema, um algoritmo apresenta vantagens e desvantagens sobre outro
- Como comparar?

Introdução

- Devemos comparar as complexidades dos algoritmos
- Qual a **operação dominante**?
 - Número de comparações entre elementos, na maioria dos casos

Algoritmos de Ordenação Baseados em Troca

Algoritmos de Ordenação Baseados em Troca

- Mais conhecidos algoritmos baseados em troca
 - **Bubble-sort**, também chamado método da bolha
 - **Quick-sort**, ou ordenação rápida ou, ainda, ordenação por troca de partição

Bubble-Sort

- É um dos métodos mais conhecidos e intuitivos
- Idéia básica
 - Percorrer o vetor várias vezes
 - A cada iteração, comparar cada elemento com seu sucessor ($\text{vetor}[i]$ com $\text{vetor}[i+1]$) e trocá-los de lugar caso estejam na ordem incorreta

Bubble-Sort

- $X = (25, 57, 48, 37, 12, 92, 86, 33)$
 - $X[0]$ com $X[1]$ (25 com 57) não ocorre permutação
 - $X[1]$ com $X[2]$ (57 com 48) ocorre permutação
 - $X[2]$ com $X[3]$ (57 com 37) ocorre permutação
 - $X[3]$ com $X[4]$ (57 com 12) ocorre permutação
 - $X[4]$ com $X[5]$ (57 com 92) não ocorre permutação
 - $X[5]$ com $X[6]$ (92 com 86) ocorre permutação
 - $X[6]$ com $X[7]$ (92 com 33) ocorre permutação

(25, 48, 37, 12, 57, 86, 33, 92)

Bubble-Sort

- Depois do primeiro passo
 - vetor = (24 , 48 , 37 , 12 , 57 , 86 , 33 , 92)
 - O maior elemento (92) está na posição correta
- Para um vetor de n elementos, são necessárias $n-1$ iterações
- A cada iteração, os elementos vão assumindo suas posições corretas

Bubble-Sort

```
for (j = 0; j < n-1; j++) {  
    for(i= 0; i < n-j-1; i++){  
        if (x[i] > x[i+1]){  
            aux = x[i];  
            x[i] = x[i+1];  
            x[i+1] = aux;  
        }  
    }  
}
```

Bubble-Sort

- Complexidade $O(n^2)$
- É possível melhorar o Bubble-sort?

```
for (j = 0; j < n-1; j++) {  
    for(i= 0; i < n-j-1; i++){  
        if (x[i] > x[i+1]){  
            aux = x[i];  
            x[i] = x[i+1];  
            x[i+1] = aux;  
        }  
    }  
}
```

Bubble-Sort

- Que melhorias podem ser feitas?

■ passo 0 (vetor original)	25	57	48	37	12	92	86	33
■ passo 1	25	48	37	12	57	86	33	92
■ passo 2	25	37	12	48	57	33	86	92
■ passo 3	25	12	37	48	33	57	86	92
■ passo 4	12	25	37	33	48	57	86	92
■ passo 5	12	25	33	37	48	57	86	92
■ passo 6	12	25	33	37	48	57	86	92
■ passo 7	12	25	33	37	48	57	86	92

Bubble-Sort

- Que melhorias podem ser feitas?

■ passo 0 (vetor original)	25	57	48	37	12	92	86	33
■ passo 1	25	48	37	12	57	86	33	92
■ passo 2	25	37	12	48	57	33	86	92
■ passo 3	25	12	37	48	33	57	86	92
■ passo 4	12	25	37	33	48	57	86	92
■ passo 5	12	25	33	37	48	57	86	92
■ passo 6	12	25	33	37	48	57	86	92
■ passo 7	12	25	33	37	48	57	86	92

- Detectar se o vetor já está ordenado

Bubble-Sort

```
troca = 1;
for (j = 0; (j < n-1) && troca; j++) {
    troca = 0;
    for(i= 0; i < n-j-1; i++){
        if (x[i] > x[i+1]){
            troca = 1;
            aux = x[i];
            x[i] = x[i+1];
            x[i+1] = aux;
        }
    }
}
```

Bubble-Sort

```
troca = 1;
for (j = 0; (j < n-1) && troca; j++) {
    troca = 0;
    for(i= 0; i < n-j-1; i++){
        if (x[i] > x[i+1]){
            troca = 1;
            aux = x[i];
            x[i] = x[i+1];
            x[i+1] = aux;
        }
    }
}
```

Melhor Caso

$$\Theta(n)$$

Pior caso

$$\Theta(n^2)$$

Quick-Sort

Quick-Sort

- Melhoramento do bubble-sort
 - Troca de elementos distantes são mais efetivas
- Idéia básica: dividir para conquistar
 - Dividir o vetor em dois vetores menores que serão ordenados independentemente e combinados para produzir o resultado final

Quick-Sort

■ Primeiro passo

■ Elemento pivô: v

- Colocar v em sua posição correta
- Ordenar de forma que os elementos à esquerda do pivô são menores que o mesmo e os elementos à direita são maiores
- Percorrer o vetor X da esquerda para a direita até $X[i] > v$; e da direita para a esquerda até $X[j] < v$
 - Troca $X[i]$ com $X[j]$
- Quando i e j cruzarem, a iteração finaliza e v troca de lugar com i

→
down
25 57 48 37 12 86 92 33
←
up

■ Segundo passo

- Ordenar sub-vetores abaixo e acima do elemento pivô

Quick-Sort

→
down

25 57 48 37 12 86 92

←
up

33

ponteiros inicializados

Pivô = 25

Quick-Sort

Pivô = 25

→
down

25 57 48 37 12 86 92

down

25 57 48 37 12 86 92

←
up

33

up

33

ponteiros inicializados

procura-se down > que pivô

Quick-Sort

Pivô = 25

\rightarrow down	25	57	48	37	12	86	92	\leftarrow up	33	ponteiros inicializados
	25	57	48	37	12	86	92	up	33	procura-se down > que pivô
	25	57	48	37	12	86	92	down	33	procura-se up < que pivô

Quick-Sort

Pivô = 25

→ down	25	57	48	37	12	86	92	← up	33	ponteiros inicializados
	25	57	48	37	12	86	92	33		procura-se down > que pivô
	25	57	48	37	12	86	92	33		procura-se up < que pivô
	25	12	48	37	57	86	92	33		*troca*

Quick-Sort

Pivô = 25

→ down	25	57	48	37	12	86	92	← up	33	ponteiros inicializados
	25	57	48	37	12	86	92		33	procura-se down > que pivô
	25	57	48	37	12	86	92		33	procura-se up < que pivô
	25	12	48	37	57	86	92		33	*troca*
	25	12	48	37	57	86	92		33	procura-se down > que pivô

Quick-Sort

Pivô = 25

→ down	25	57	48	37	12	86	92	33	← up	ponteiros inicializados
	25	57	48	37	12	86	92	33		procura-se down > que pivô
	25	57	48	37	12	86	92	33		procura-se up < que pivô
	25	12	48	37	57	86	92	33		*troca*
	25	12	48	37	57	86	92	33		procura-se down > que pivô
	25	12	48	37	57	86	92	33		procura-se up < que pivô

Quick-Sort

Pivô = 25

→							←	
down							up	
25	57	48	37	12	86	92	33	ponteiros inicializados
	down						up	
25	57	48	37	12	86	92	33	procura-se down > que pivô
	down			up				
25	57	48	37	12	86	92	33	procura-se up < que pivô
	down			up				
25	12	48	37	57	86	92	33	*troca*
		down		up				
25	12	48	37	57	86	92	33	procura-se down > que pivô
	up	down						
25	12	48	37	57	86	92	33	procura-se up < que pivô
	up	down						
12	25	48	37	57	86	92	33	Down e Up cruzaram Troca com pivo (posição ideal)

Quick-Sort

- Todo elemento à esquerda de 25 é ≤ 25
- Todo elemento à direita de 25 é ≥ 25
- Ordenar os dois subvetores (12) e (48 37 57 86 92 33)

Quick-Sort

(12) 25 (48 37 57 86 92 33)

12 25 (48 37 57 86 92 33)

Quick-Sort

(12) 25 (48 37 57 86 92 33)

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Down > Pivo

12 25 (**48** 37 57 86 92 33)

Quick-Sort

(12) 25 (48 37 57 86 92 33)

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Down > Pivo

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Up < Pivo

12 25 (**48** 37 57 86 92 33)

Quick-Sort

(12) 25 (48 37 57 86 92 33)

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Down > Pivo

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Up < Pivo

12 25 (**48** 37 57 86 92 33)

Down

Up

Troca Down com UP

12 25 (**48** 37 33 86 92 57)

Quick-Sort

(12) 25 (48 37 57 86 92 33)

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Down > Pivo

12 25 (**48** 37 57 86 92 33)

Down

Up

Procura Up < Pivo

12 25 (**48** 37 57 86 92 33)

Down

Up

Troca Down com UP

12 25 (**48** 37 33 86 92 57)

Down Up

Procura Down > Pivo

12 25 (**48** 37 33 86 92 57)

Quick-Sort

Down Up

Procura Up < Pivo

12 25 (48 37 33 86 92 57)

Quick-Sort

Down Up

Procura Up < Pivo

12 25 (48 37 33 86 92 57)

Up Down

Procura Up < Pivo

12 25 (48 37 33 86 92 57)

Quick-Sort

12 25 (48 37 33 86 92 57)

Down Up

Procura Down > Pivo

Up Down

Procura Up < Pivo

12 25 (48 37 33 86 92 57)

Up Down

Não troca, pois cruzaram

12 25 (48 37 33 86 92 57)

Quick-Sort

	Down Up	Procura Down > Pivo
12 25 (48 37 33 86 92 57)		
	Up Down	Procura Up < Pivo
12 25 (48 37 33 86 92 57)		
	Up Down	Não troca, pois cruzaram
12 25 (48 37 33 86 92 57)		
	Up Down	Pivo troca com Up
12 25 (33 37 48 86 92 57)		

Quick-Sort

	Down Up	Procura Down > Pivo
12 25 (48 37 33 86 92 57)		
	Up Down	Procura Up < Pivo
12 25 (48 37 33 86 92 57)		
	Up Down	Não troca, pois cruzaram
12 25 (48 37 33 86 92 57)		
	Up Down	Pivo troca com Up
12 25 (33 37 48 86 92 57)		
	Up Down	Novas Partições
12 25 (33 37) 48 (86 92 57)		

Quick-Sort

Down UP

Procura Down maior que Pivo

12 25 (33 37) 48 (86 92 57)

Quick-Sort

Down UP

Procura Down maior que Pivo

12 25 (33 37) 48 (86 92 57)

UP Down

Procura UP menor que Pivo

12 25 (33 37) 48 (86 92 57)

Quick-Sort

Down Up

Procura Down maior que Pivo

12 25 (33 37) 48 (86 92 57)

UP Down

Procura UP menor que Pivo

12 25 (33 37) 48 (86 92 57)

UP Down

Não troca, pois cruzaram

12 25 (33 37) 48 (86 92 57)

Quick-Sort

Down UP

Procura Down maior que Pivo

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Procura UP menor que Pivo

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Não troca, pois cruzaram

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Pivo troca com Up

12 25 (**33** 37) 48 (**86** 92 57)

Quick-Sort

Down UP

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Procura Down maior que Pivo

Procura UP menor que Pivo

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Não troca, pois cruzaram

12 25 (**33** 37) 48 (**86** 92 57)

UP Down

Pivo troca com Up

12 25 (**33** 37) 48 (**86** 92 57)

Nova Partição com único elemento

12 25 **33** (37) 48 (**86** 92 57)

Quick-Sort

Nova Partição com único elemento

12 25 33 (37) 48 (86 92 57)

Quick-Sort

Nova Partição com único elemento

12 25 33 (37) 48 (86 92 57)

Down Up Procura Down > Pivo

12 25 33 37 48 (86 92 57)

Quick-Sort

Nova Partição com único elemento

12 25 33 (37) 48 (86 92 57)

Down Up Procura Down > Pivo

12 25 33 37 48 (86 92 57)

Down Up Procura Up <Pivo

12 25 33 37 48 (86 92 57)

Quick-Sort

Nova Partição com único elemento

12 25 33 (**37**) 48 (**86** 92 57)

Down Up

Procura Down > Pivo

12 25 33 37 48 (**86** 92 57)

Down Up

Procura Up <Pivo

12 25 33 37 48 (**86** 92 57)

Down Up

Troca Down com Up

12 25 33 37 48 (**86** 57 92)

Quick-Sort

Nova Partição com único elemento

12 25 33 (37) 48 (86 92 57)

Down Up

Procura Down > Pivo

12 25 33 37 48 (86 92 57)

Down Up

Procura Up < Pivo

12 25 33 37 48 (86 92 57)

Down Up

Troca Down com Up

12 25 33 37 48 (86 57 92)

Up Down

Não troca, pois cruzaram

12 25 33 37 48 (86 57 92)

Quick-Sort

Up Down Não troca, pois cruzaram

12 25 33 37 48 (86 57 92)

Quick-Sort

Up Down Não troca, pois cruzaram

12 25 33 37 48 (86 57 92)

Up Down Pivo troca com Up

12 25 33 37 48 (57 86 92)

Quick-Sort

Up Down Não troca, pois cruzaram

12 25 33 37 48 (86 57 92)

Up Down Pivo troca com Up

12 25 33 37 48 (57 86 92)

Novas Partições com um elemento

12 25 33 37 48 (57) 86 (92)

Quick-Sort

Up Down Não troca, pois cruzaram

12 25 33 37 48 (86 57 92)

Up Down Pivo troca com Up

12 25 33 37 48 (57 86 92)

Novas Partições com um elemento

12 25 33 37 48 (57) 86 (92)

Fim do algoritmo: vetor está ordenado

12 25 33 37 48 57 86 92

Quick-Sort

- 25 57 48 37 12 86 92 33
- (12) 25 (48 37 57 86 92 33)
- 12 25 (48 37 57 86 92 33)
- 12 25 (33 37) 48 (86 92 57)
- 12 25 33 (37) 48 (86 92 57)
- 12 25 33 37 48 (86 92 57)
- 12 25 33 37 48 (57) 86 (92)
- 12 25 33 37 48 57 86 (92)
- 12 25 33 37 48 57 86 92

Quick-Sort

```
1 void Quicksort (int X[], int p, int r)
2 {
3     if (p < r) {
4         q = Partição(X, p, r);
5         Quicksort(X, p, q-1);
6         Quicksort(X, q+1, r);
7     }
8 }
```

```
1 int Partição (int X[], int p, int r) {
2     x = X[p];  up = r;  down = p;
3     while (down < up) {
4         while (X[down] <= x) {
5             down = down+1;
6         }
7         while(X[up] > x) {
8             up = up - 1;
9         }
10        if (down < up)
11            troca(X[down], X[up]);
12    }
13    X[p]= X[up];
14    X[up] = x;
15    return(up);
16 }
```

Quick-Sort

- Se o particionamento gerar dois subconjuntos de tamanho $n/2$ temos a recorrência $T(n) = 2T(n/2) + n$
- Caso 2 do teorema mestre
 - $f(n) = \Theta(n)$

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

Quick-Sort

D

U

(1 2 3 4 5 6 7 8 9)

D

U

(1 2 3 4 5 6 7 8 9)

Quick-Sort

D U
(1 2 3 4 5 6 7 8 9)

D U
(1 2 3 4 5 6 7 8 9)

U D
(1 2 3 4 5 6 7 8 9)

Quick-Sort

D U
(1 2 3 4 5 6 7 8 9)

D U
(1 2 3 4 5 6 7 8 9)

U D
(1 2 3 4 5 6 7 8 9)

1 (2 3 4 5 6 7 8 9)

1 (2 3 4 5 6 7 8 9)

Quick-Sort

D

U

1 (2 3 4 5 6 7 8 9)

Quick-Sort

D

U

1 (2 3 4 5 6 7 8 9)

U D

1 (2 3 4 5 6 7 8 9)

1 2 (3 4 5 6 7 8 9)

Quick-Sort

D

U

1 (2 3 4 5 6 7 8 9)

U D

1 (2 3 4 5 6 7 8 9)

1 2 (3 4 5 6 7 8 9)

D

U

1 2 (3 4 5 6 7 8 9)

Quick-Sort

D U

1 (2 3 4 5 6 7 8 9)

U D

1 (2 3 4 5 6 7 8 9)

1 2 (3 4 5 6 7 8 9)

D U

1 2 (3 4 5 6 7 8 9)

D U

1 2 (3 4 5 6 7 8 9)

Quick-Sort

 D U
1 2 (3 4 5 6 7 8 9)

 U D
1 2 (3 4 5 6 7 8 9)

1 2 3 (4 5 6 7 8 9)

Quick-Sort

(1 2 3 4 5 6 7 8 9)

1 (2 3 4 5 6 7 8 9)

1 2 (3 4 5 6 7 8 9)

1 2 3 (4 5 6 7 8 9)

1 2 3 4 (5 6 7 8 9)

1 2 3 4 5 (6 7 8 9)

1 2 3 4 5 6 (7 8 9)

1 2 3 4 5 6 7 (8 9)

1 2 3 4 5 6 7 8 (9)

1 2 3 4 5 6 7 8 9

Quick-Sort

- O pior caso do quick-sort é quando o particionamento criar uma partição com 1 elemento e outra com $n-1$
 - Caso em que o vetor está ordenado
- Se isso se repetir, temos a recorrência
 - $T(n) = T(n-1) + n$
 - Resultando em uma complexidade

$$T(n) = \Theta(n^2)$$

Adaptado de

Métodos de Ordenação



SCC-201 Introdução à Ciência da Computação II

Rosane Minghim

2010/2011

Baseado no material dos Professores Rudinei Goularte e Thiago Pardo

Referências Bibliográficas

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; (2002). Algoritmos –Teoria e Prática. Tradução da 2ª edição americana. Rio de Janeiro. Editora Campus
- TAMASSIA, ROBERTO; GOODRICH, MICHAEL T. (2004). Projeto de Algoritmos -Fundamentos, Análise e Exemplos da Internet
- ZIVIANI, N. (2007). Projeto e Algoritmos com implementações em Java e C++. São Paulo. Editora Thomson