

## Métodos de Ordenação

Parte 2

SCC-601 Introdução à Ciência da Computação II

Rosane Minghim  
2010

Baseado no material dos Professores Rudinei Goularte e Thiago Pardo

1

## Ordenação por Inserção

- Idéia básica: inserir um dado elemento em sua posição correta em um conjunto já ordenado
  - Inserção Simples, ou inserção direta
  - Shell-sort, ou classificação de shell ou, ainda, classificação de incremento decrescente

2

## Inserção Simples

- Idéia básica
  - Ordenar o conjunto inserindo os elementos em um subconjunto já ordenado
    - No  $i$ -ésimo passo, inserir o  $i$ -ésimo elemento na posição correta entre  $x[0], \dots, x[i-1]$ , que já estão em ordem
      - Elementos são realocados

3

## Inserção Simples

- Idéia básica
  - Exemplo

Vetor original  
10 | 30 | 31 | 15 | 50 | 60 | 5 | 22 | 35 | 14

Realocando o elemento 15  
10 | 30 | 31 | 15 | 50 | 60 | 5 | 22 | 35 | 14

30 e 31 são realocados e 15 é inserido  
10 | 15 | 30 | 31 | 50 | 60 | 5 | 22 | 35 | 14

Por que o método se chama **inserção simples**?

## Inserção Simples: exemplo

- $X = (44, 55, 12, 42, 94, 18, 06, 67)$
- passo 1 (55) 44 55 12 42 94 18 06 67
- passo 2 (12) 12 44 55 42 94 18 06 67
- passo 3 (42) 12 42 44 55 94 18 06 67
- passo 4 (94) 12 42 44 55 94 18 06 67
- passo 5 (18) 12 18 42 44 55 94 06 67
- passo 6 (06) 06 12 18 42 44 55 94 67
- passo 7 (67) 06 12 18 42 44 55 67 94

5

## Inserção Simples

- Em grupos de 3
  - Implementar Inserção Simples
  - Calcular complexidade

6

## Inserção Simples

```
void insercao(int X[], int n) {
    for (k = 1; k < n; k++) {
        y = X[k];
        for (i = k-1; i >= 0 && X[i] > y; i--)
            X[i+1] = X[i];
        X[i+1] = y;
    }
}
```

7

## Inserção Simples

- $O(n^2)$ 
  - $(n-1)+(n-2)+\dots+2+1 = (n-1) * n/2$  comparações
- Vetor ordenado:  $O(n)$
- Vetor ordenado inversamente:  $O(n^2)$
- Pouco espaço:  $O(n)$
- Realiza menos comparações que o *Bubble-sort*
  - A parte ordenada não é comparada novamente a cada iteração

8

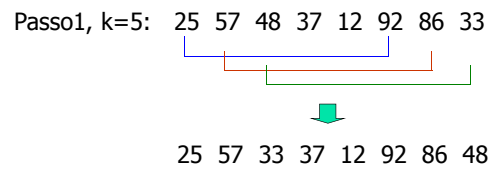
## Shell-sort

- Inserção simples é eficiente em arquivos quase ordenados
- Shell-sort: melhoria da inserção simples
  - Idéia básica: dividir a entrada em k sub-conjuntos e aplicar inserção simples a cada um, sendo que k é reduzido sucessivamente
    - A cada nova iteração, o vetor original está "mais" ordenado

9

## Shell-sort: exemplo

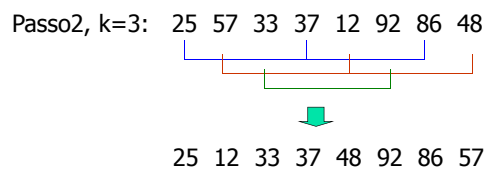
- 25 57 48 37 12 92 86 33



10

## Shell-sort: exemplo

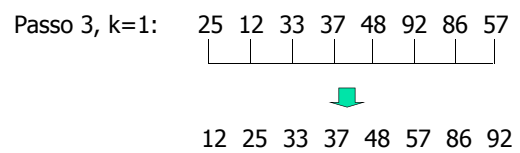
- 25 57 48 37 12 92 86 33



11

## Shell-sort: exemplo

- 25 57 48 37 12 92 86 33



12

## Shell-sort

- Os índices  $k$  são os incrementos que são adicionados a cada posição do vetor para se ter o próximo elemento do sub-conjunto
- A cada iteração,  $k$  decresce
  - Daí o nome "incrementos decrescentes" do método
  - Shell era o nome do criador do método
- O último incremento deve sempre ser 1

13

## Shell-sort

- $k = 5, n = 15$ 
  - 1 -  $x[0]$   $x[5]$   $x[10]$
  - 2 -  $x[1]$   $x[6]$   $x[11]$
  - 3 -  $x[2]$   $x[7]$   $x[12]$
  - 4 -  $x[3]$   $x[8]$   $x[13]$
  - 5 -  $x[4]$   $x[9]$   $x[14]$
- O  $i$ -ésimo elemento do  $j$ -ésimo conjunto é:  
 $x[(i-1) * k + j - 1]$

14

## Shell-sort

- 25 57 48 37 12 92 86 33
- Passo 1 (incremento 5):
  - $(x[0], x[5])$
  - $(x[1], x[6])$
  - $(x[2], x[7])$
  - $(x[3])$
  - $(x[4])$
- Passo 2 (incremento 3):
  - $(x[0], x[3], x[6])$
  - $(x[1], x[4], x[7])$
  - $(x[2], x[5])$
- Passo 3 (incremento 1):
  - $(x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7])$

15

## Shell-sort

Vetor com incrementos ( $k$ )  
Número de elementos no vetor  $incrmnts$

```
void shell-sort (int x[], int n, int incrmnts[], int numinc) {  
    int incr, j, k, span, y;  
    for (incr = 0; incr < numinc; incr++) {  
        k = incrmnts[incr];  
        for (j = k; j < n; j++) {  
            y = x[j];  
            for (i = j - k; i >= 0 && x[i] > y; i -= k)  
                x[i+k] = x[i];  
            x[i+k] = y;  
        }  
    }  
}
```

16

## Shell-sort

- Exercício
  - Executar o algoritmo anterior para o vetor (25 57 48 37 12 92 86 33)
  - 3 incrementos: 5, 3 e 1

17

## Shell-sort

- Foi demonstrado que, com uma seqüência adequada de incrementos de  $k$ , shell-sort é aproximadamente  $O(n(\log n)^2)$ 
  - Prova da eficiência do shell-sort está além do escopo desta disciplina

18

## Shell-sort

- Escolha dos incrementos
  - Knuth sugere:
    - Defina uma função recursiva  $h$  tal que:
      - $h(1) = 1$  e  $h(i + 1) = 3 * h(i) + 1$
    - Seja  $x$  o menor inteiro tal que  $h(x) \geq n$ :
      - $numinc$  será  $x - 2$
      - $incrmnts[i]$  será  $h(numinc - i + 1)$  para  $i$  de 1 até  $numinc$

19