# Algoritmos e Estruturas de Dados CES-11

Prof. Paulo André Castro
pauloac@ita.br
Sala 110 – Prédio da Computação
www.comp.ita.br/~pauloac
IECE - ITA

## Cap. 5. Técnicas de Ordenação

**5.1. Introdução**

**5.2. Métodos simples de ordenação**

**5.3. O método Shell-Sort**

**5.4. O método Quick-Sort**



## 5.1 – Introdução

- **Ordenação** é uma atividade importantíssima em processamento automático de informações.
- Que tal **procurar**
  - Um nome numa lista telefônica
  - Um nome numa lista de contas bancárias
  - Uma palavra em um dicionário lingüístico

se as **relações** não estiverem em **ordem alfabética**?



- O processo de ordenação se aplica a um conjunto de **objetos** em que se possa verificar entre eles a relação

  ≤ (**menor ou igual**), ou ≥ (**maior ou igual**).

- Os conjuntos mais comuns são vetores de
  - Números
  - Nomes ou cadeias de caracteres em geral



- **Vetores de estruturas** também podem ser ordenados
- **Exemplo:** relação com diversas informações sobre os funcionários de uma empresa:

| Nome | Data de Nascimento | | | Setor | Salário |
|---|---|---|---|---|---|
| | Dia | Mês | Ano | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

| Nome | Data de Nascimento | | | Setor | Salário |
|---|---|---|---|---|---|
| | Dia | Mês | Ano | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |

- Este vetor pode ser ordenado por **qualquer campo** dessas estruturas:
  - Ordem alfabética dos **nomes** ou dos **setores**
  - Ordem crescente ou decrescente de **salários** ou de **idade**
- O **campo** de um vetor de estruturas usado para **ordená-lo** é denominado **chave** da ordenação.

- Existem vários algoritmos para fazer ordenação de vetores:
  - Alguns mais **simples**, porém **ineficientes**
  - Outros mais **eficientes**, porém mais **complexos**
- Alguns **critérios** para julgar a **eficiência** dos algoritmos de ordenação:
  - Número de **comparações** de chaves
  - Número de **movimentações** de elementos

7

- O **tempo de comparação** depende da **chave** usada:
  - Para números, basta um dos operadores **≤** ou **≥**;
  - Para cadeias de caracteres, usa-se funções para compará-las; exemplo: **strcmp** da linguagem C
- Há casos de **chaves múltiplas**, como a ordenação por **idade**, sendo dada a **data de nascimento**:
  - Compara-se primeiramente o **ano**, depois o **mês** e finalmente o **dia**
  - Nestes casos, a comparação é mais **complexa** e pode ser conveniente um **subprograma** específico

8

- O **tempo de movimentação** depende do **tipo** e **tamanho** dos elementos do vetor a ser ordenado:
  - Para **números**, a movimentação é **pequena**
  - Para **grandes estruturas**, há **grande** movimentação de dados

9

- No caso de **grandes estruturas**, usa-se um **vetor de ponteiros**, cada um para uma estrutura do vetor

| Pont | Nome | Data de Nascimento | | | Setor | Salário |
|---|---|---|---|---|---|---|
| | | Dia | Mês | Ano | | |
| 1 | Gilberto | 03 | 05 | 1980 | Almox | 850 |
| 2 | Antonio | 05 | 12 | 1970 | Admin | 2300 |
| 3 | Pedro | 23 | 07 | 1975 | Fabric | 1050 |
| 4 | Augusto | 05 | 10 | 1960 | Diret | 8000 |
| 5 | Claudio | 07 | 03 | 1984 | Ferr. | 1850 |
| 6 | Marcos | 17 | 11 | 1976 | Recep | 850 |
| 7 | Nilton | 20 | 01 | 1982 | Proj | 2300 |
| 8 | Vicente | 15 | 08 | 1979 | Contr | 2000 |

- Para a ordenação basta **alterar os apontamentos**, evitando-se a movimentação dessas estruturas

| Pont | Nome | Data de Nascimento | | | Setor | Salário |
|---|---|---|---|---|---|---|
| | | Dia | Mês | Ano | | |
| 1 | Gilberto | 03 | 05 | 1980 | Almox | 850 |
| 2 | Antonio | 05 | 12 | 1970 | Admin | 2300 |
| 3 | Pedro | 23 | 07 | 1975 | Fabric | 1050 |
| 4 | Augusto | 05 | 10 | 1960 | Diret | 8000 |
| 5 | Claudio | 07 | 03 | 1984 | Ferram. | 1850 |
| 6 | Marcos | 17 | 11 | 1976 | Recep | 850 |
| 7 | Nilton | 20 | 01 | 1982 | Proj | 2300 |
| 8 | Vicente | 15 | 08 | 1979 | Contr | 2000 |

- No restante deste capítulo, para maior **simplicidade**, os **algoritmos** irão atuar sobre **vetores de inteiros**.
- A seguinte declaração será usada por esses algoritmos:

```
typedef int vetor [nmax]
```

(**nmax** é o número máximo de elementos dos vetores)

- Para vetores de **outros tipos**, basta alterar as **comparações** e os **movimentos** de elementos.
- São apresentados a seguir **três métodos simples** de ordenação e, depois, **quatro** outros mais **eficientes**.

12

## 5.2 – Métodos Simples de Ordenação

- Tais métodos são considerados lentos $\mathbf{0(n^2)}$ no pior caso e no caso médio)
- São usados para ordenar vetores pequenos
- São apresentados aqui três desses métodos:
  - Bubble-Sort
  - Insertion-Sort
  - Selection-Sort

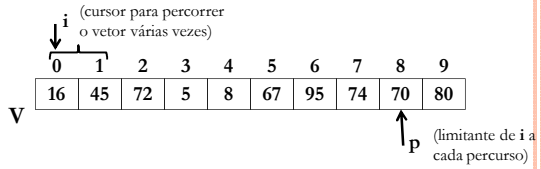
13

### 5.2.1 – Bubble-Sort

- O método consiste em:
  - **Percorrer o vetor** várias vezes
  - Durante cada percurso, efetuar troca de posição de **elementos adjacentes**, caso o elemento da esquerda seja maior que o da direita
  - Se, durante um dos percursos, não houver trocas, considera-se que o vetor está ordenado
  - Em cada percurso, um elemento atinge sua posição definitiva

14

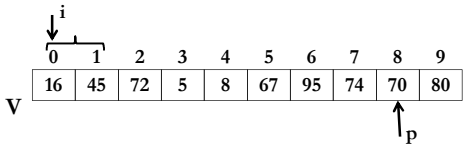
**Exemplo:** seja o vetor abaixo e o método em seu início




trocou
(semáforo que acende quando há troca)

15

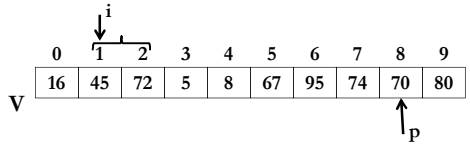
**Não trocar**




trocou

16

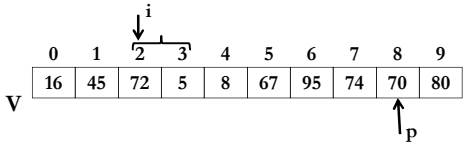
**Não trocar**




trocou

17

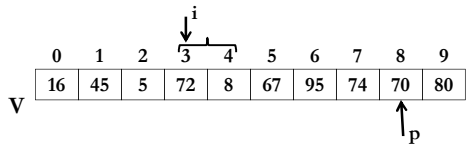
**Trocar**




trocou

18

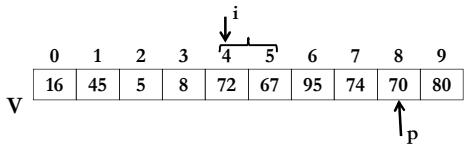
**Trocar**




trocou

19

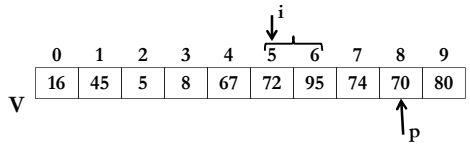
**Trocar**




trocou

20

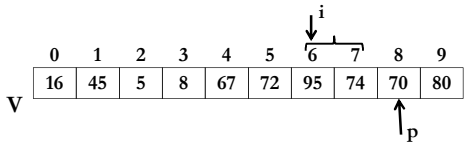
**Não trocar**




trocou

21

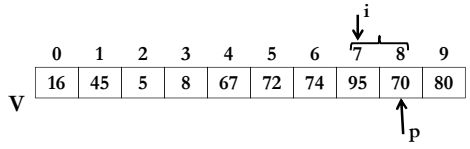
**Trocar**




trocou

22

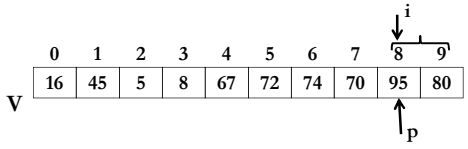
**Trocar**




trocou

23

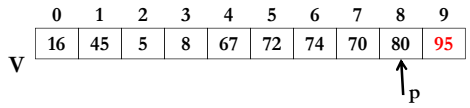
**Trocar**




trocou

24

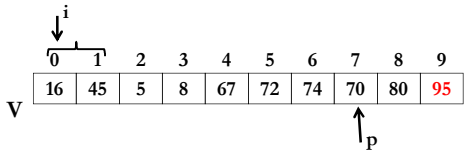
**95 em sua posição definitiva**
**trocou acesa: começar novo percurso**
**retroceder p**




trocou

25

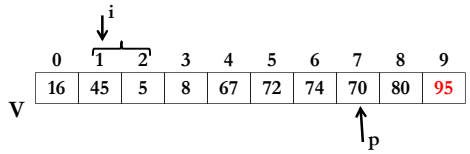
**Não trocar**




trocou

26

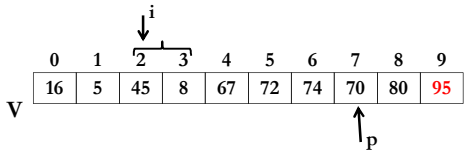
**Trocar**




trocou

27

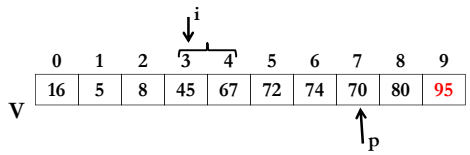
**Trocar**




trocou

28

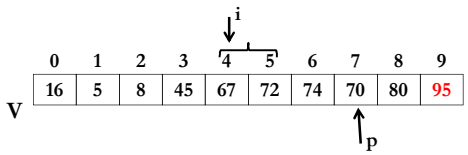
**Não trocar**




trocou

29

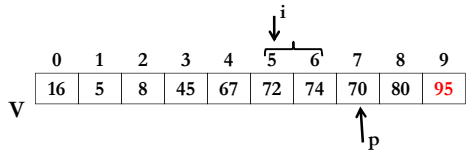
**Não trocar**




trocou

30

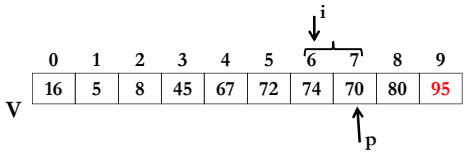
**Não trocar**




trocou

31

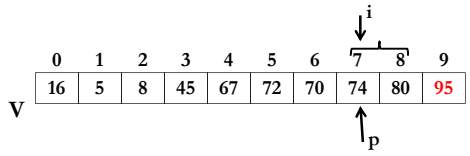
**Trocar**




trocou

32

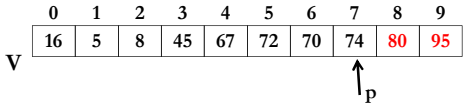
**Não trocar**




trocou

33

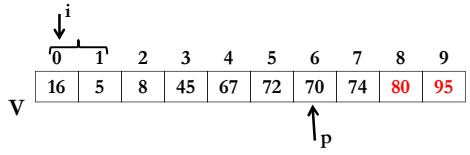
**trocou acesa: iniciar novo percurso**




trocou

34

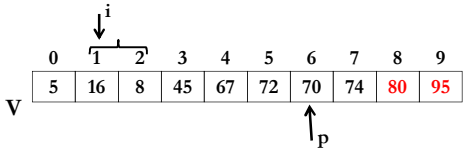
**Trocar**




trocou

35

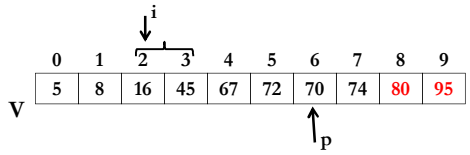
**Trocar**




trocou

36

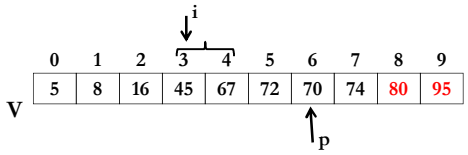
**Não trocar**




trocou

37

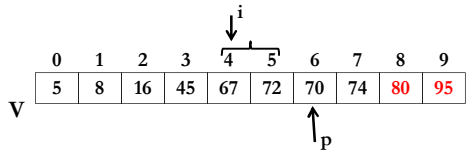
**Não trocar**




trocou

38

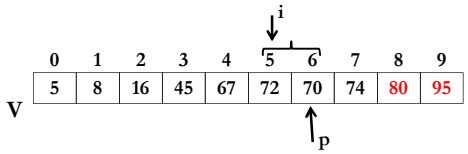
**Não trocar**




trocou

39

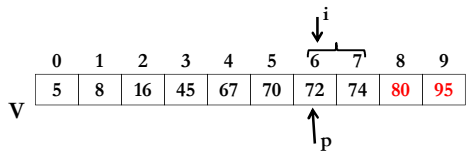
**Trocar**




trocou

40

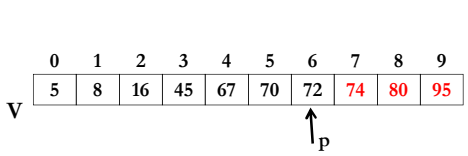
**Não trocar**




trocou

41

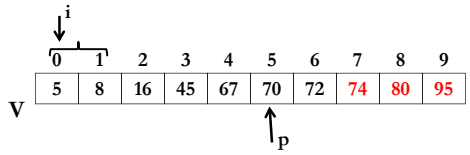
**trocou acesa: novo percurso**




trocou

42

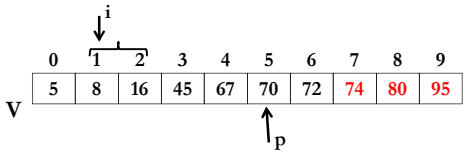
**Não trocar**




trocou

43

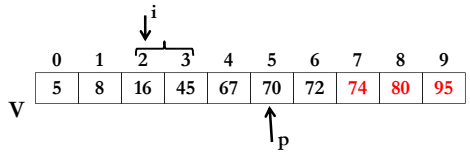
**Não trocar**




trocou

44

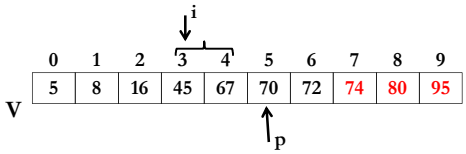
**Não trocar**




trocou

45

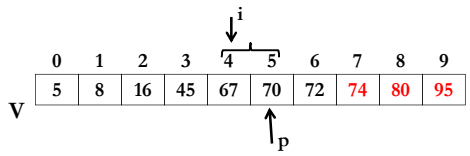
**Não trocar**




trocou

46

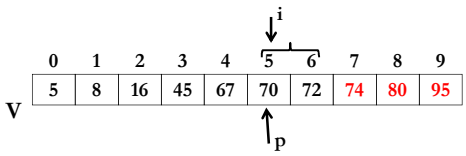
**Não trocar**




trocou

47

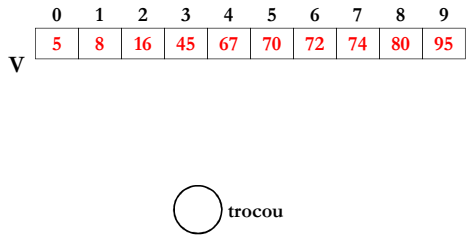
**Não trocar**




trocou

48

**trocou apagada: vetor ordenado**




49

A função **BubbleSort**:

```
void BubbleSort (int n, vetor V) {
  int i, p;
  logic trocou;
  for (p = n-2, trocou = TRUE; p >= 0 && trocou; p--)
      for (trocou = FALSE, i = 0; i <= p; i++)
            if (V[i] > V[i+1]) {
                  Trocar (&V[i], &V[i+1]);
                  trocou = TRUE;
            }
}

void Trocar (int *x, int *y) {
  int temp;
  temp = *x; *x = *y; *y = temp;
}
```

A cada vez que a condição do if se verificar, haverá movimentação de 3 elementos

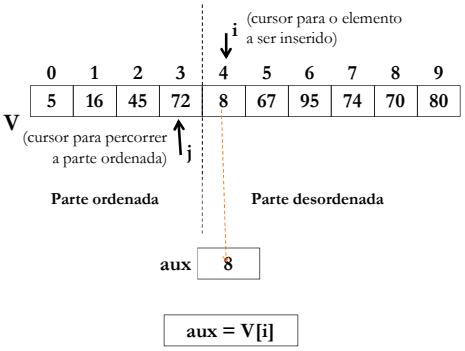
50

**5.2.2 – Insertion-Sort**

- O método considera o vetor sempre **dividido** em duas partes: a parte **ordenada**, do lado esquerdo, e a parte **desordenada**, do lado direito.
- Inicialmente, a parte **ordenada** contém apenas **V[0]**; depois, cada elemento da parte desordenada é inserido na parte ordenada, mantendo a ordenação
- A cada elemento a ser inserido, percorre-se a parte ordenada até encontrar a **posição correta** para inserção.

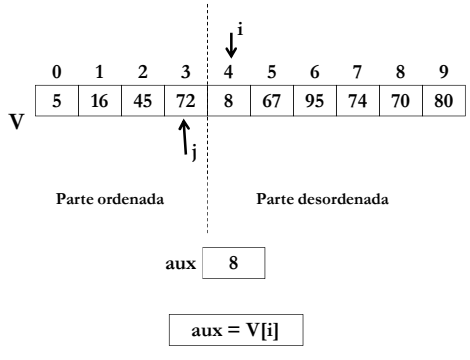
51

**Exemplo:** seja o vetor abaixo e o método no início da inserção de um elemento genérico



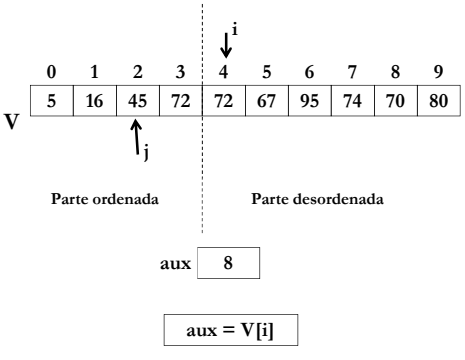

52

**aux < V[j]: V[j+1] recebe V[j]; retroceder j**



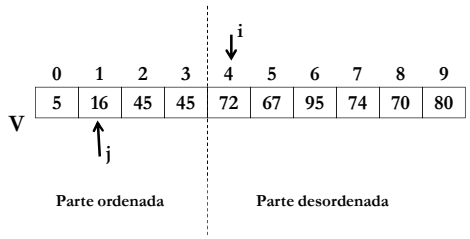

53

**aux < V[j]: V[j+1] recebe V[j]; retroceder j**




54

**aux < V[j]: V[j+1] recebe V[j]; retroceder j**

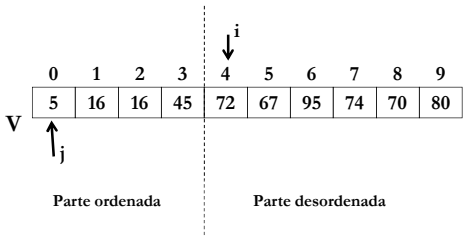



aux 8

aux = V[i]

55

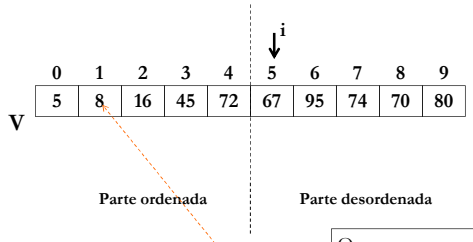
**aux >= V[j]: V[j+1] recebe aux; avançar a parte ordenada**




aux 8

aux = V[i]

56




aux 8

O mesmo se repete até que tudo fique ordenado

aux = V[i]

57

A função **InsertionSort**:

```
void InsertionSort (int n, vetor V) {
  int i, j, aux;
  int achou;
  for (i = 1; i < n; i++)  {
       aux = V[i];
       achou = FALSE; j = i-1;
       while (j >= 0 && !achou)
             if (aux < V[j]) {
                   V[j+1] = V[j];
                   j--;
             }
             else achou = TRUE;
       if (j+1 != i)
             V[j+1] = aux;
  }
}
```

A cada vez que a condição do if se verificar, haverá movimentação de 1 elemento

Antes e depois do while mais uma

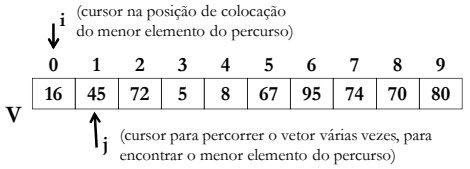
58

**5.2.3 – Selection-Sort**

- O método consiste em **percorrer o vetor** várias vezes
- Durante cada percurso, **seleciona o menor elemento**, colocando-o em sua **posição definitiva**.

59

**Exemplo:** seja o vetor abaixo e o método em seu início

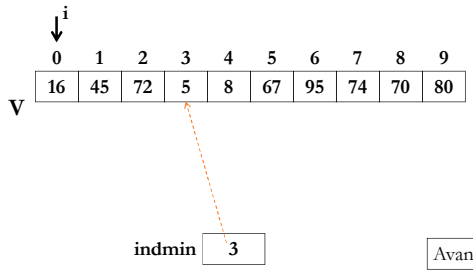



indmin

(destinado a guardar o índice do menor elemento de um percurso)

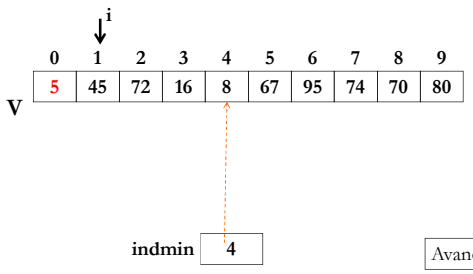
60

**Após o 1º percurso:**
**indmin ≠ i: trocar V[i] com V[indmin]**



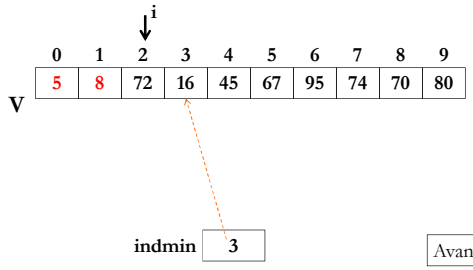

61

**Após o 2º percurso:**
**indmin ≠ i: trocar V[i] com V[indmin]**

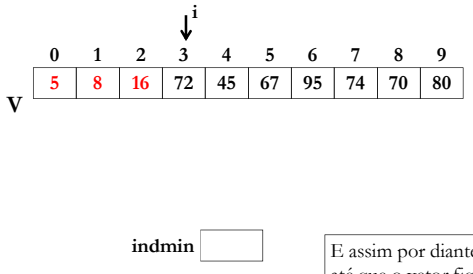



62

**Após o 3º percurso:**
**indmin ≠ i: trocar V[i] com V[indmin]**




63




64

A função **SelectionSort**:

```
void SelectionSort (int n, vetor V) {
  int i, j, indmin;
  for (i = 0; i < n-1; i++) {
      indmin = i;
      for (j = i+1; j <= n-1; j++)
            if (V[j] < V[indmin])
                  indmin = j;
      if (indmin != i)
            Trocar (&V[i], &V[indmin]);
  }
}
```

Só ocorre movimentação de elementos quando o índice do menor for encontrado

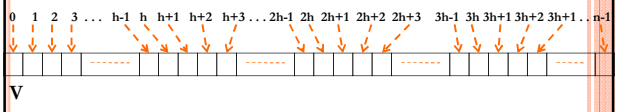
65

## 5.3 – O MÉTODO SHELL-SORT

- O método Shell-Sort é uma **generalização** do **Insertion-Sort**.
- No Insertion-Sort, a maioria dos **deslocamentos** de elementos é de apenas **uma casa**.
- Se o **menor** elemento estiver no **final** do vetor, são necessários **n deslocamentos** para colocá-lo ordenadamente.
- **Shell-Sort** faz movimentos de **passos maiores**

66

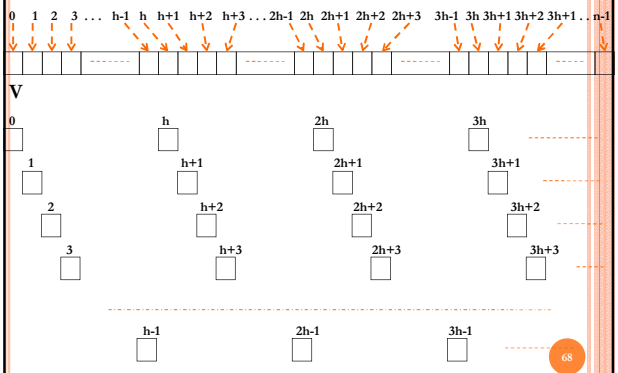
- O vetor **V** pode ser assim visualizado:




- O valor de **h < n** é empírico

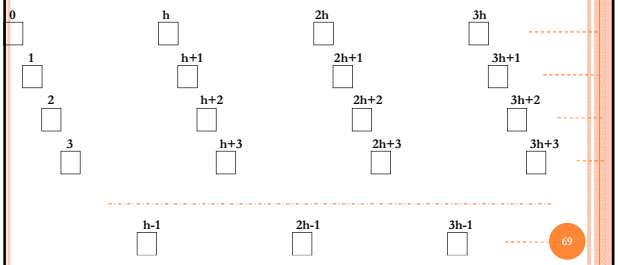
67

- Pode-se dividir **V** em **h** sub-vetores intercalados:



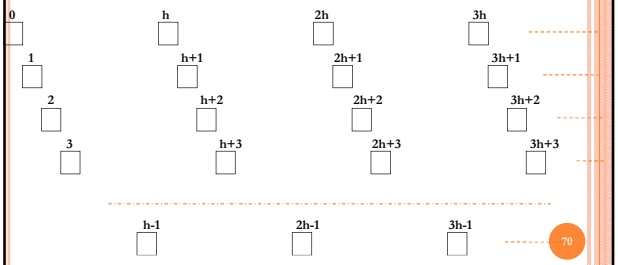

68

- O valor de **h** é inicialmente grande, mas vai caindo até 1
- Para cada valor de **h**, ordena-se cada um desses sub-vetores pelo Insertion-Sort



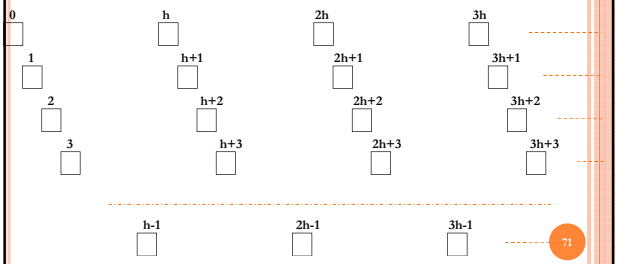

69

- Quando **h = 1**, tem-se o **Insertion-Sort original**:
  - Então **boa porção** do vetor já estará **ordenada** pelos passos anteriores (maiores valores de **h**)




70

- O **valor inicial** de **h** e a razão de seu **decréscimo** é escolhida **empiricamente**; para cada escolha, um **comportamento** diferente.




71

- O algoritmo a seguir escolhe a seguinte seqüência de valores de h, que tem dado bons valores de desempenho:

. . . , 1093, 364, 121, 40, 13, 4, 1

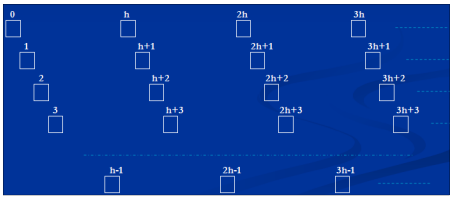
O **valor inicial** é obtido pela fórmula

**h(novo) = 3*h(antigo) + 1**

até encontrar o maior valor de **h < n**

Os valores seguintes de **h** são obtidos pela fórmula

**h(novo) = h(antigo) / 3**




72

```
void ShellSort (int n, vetor V) {
  int i, j, h, aux; int achou;

  h = 1;
  do h = 3*h + 1; while (h < n);
  do {
       h = h/3;
       for (i = h; i <= n-1; i++) {
              aux = V[i]; j = i; achou = FALSE;
             while (!achou && j >= h)
                    if (aux < V[j-h]) {
                          V[j] = V[j-h]; j = j - h;
                    }
                    else achou = TRUE;
             if (j != i) V[j] = aux;
       }
  } while (h > 1);
}
```

Insertion-Sort de passo **h**, intercalado

73

- **Comparação** entre o número de movimentos de elementos no Insertion-Sort e Shell-Sort, usando vetor de caracteres:

**a) Insertion-Sort:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | S | O | R | T | I | N | G | E | X | A | M | P | L | E |

aux

nº movtos: 1

nº acumulado movtos: 1

Sempre há um movimento para aux

```
S -> aux
```

74

**a) Insertion-Sort:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | S | O | R | T | I | N | G | E | X | A | M | P | L | E |

aux

nº movtos: 3

nº acumulado movtos: 4

```
O -> aux;
S -> 2
aux -> 1
```

75

**a) Insertion-Sort:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | O | S | R | T | I | N | G | E | X | A | M | P | L | E |

aux

nº movtos: 3

nº acumulado movtos: 7

```
R -> aux;
S -> 3
aux -> 2
```

76

**a) Insertion-Sort:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |

aux

nº movtos: 1

nº acumulado movtos: 8

```
T -> aux;
```

77

**a) Insertion-Sort:**

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | O | R | S | T | I | N | G | E | X | A | M | P | L | E |

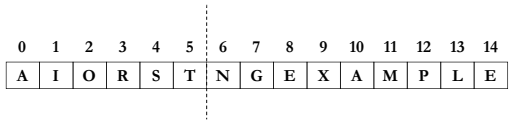
aux

nº movtos: 6

nº acumulado movtos: 14

78

**a) Insertion-Sort:**



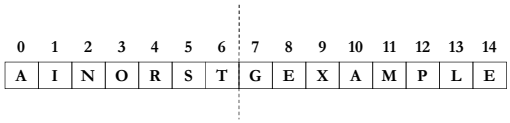

aux

nº movtos: 6

nº acumulado movtos: 20

79

**a) Insertion-Sort:**



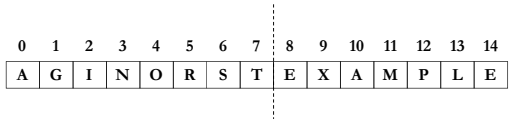

aux

nº movtos: 8

nº acumulado movtos: 28

80

**a) Insertion-Sort:**



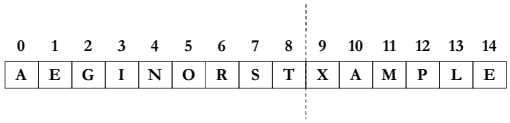

aux

nº movtos: 9

nº acumulado movtos: 37

81

**a) Insertion-Sort:**



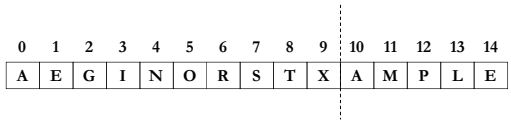

aux

nº movtos: 1

nº acumulado movtos: 38

82

**a) Insertion-Sort:**



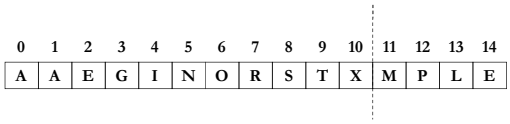

aux

nº movtos: 11

nº acumulado movtos: 49

83

**a) Insertion-Sort:**



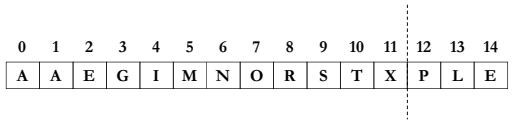

aux

nº movtos: 8

nº acumulado movtos: 57

84

**a) Insertion-Sort:**



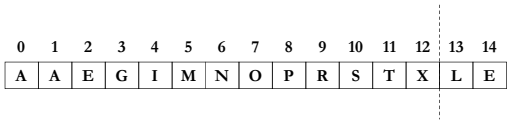

aux

nº movtos: 6

nº acumulado movtos: 63

85

**a) Insertion-Sort:**



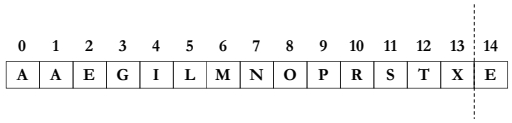

aux

nº movtos: 10

nº acumulado movtos: 73

86

**a) Insertion-Sort:**



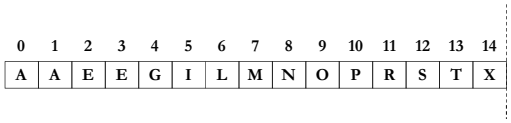

aux

nº movtos: 13

nº acumulado movtos: 86

87

**a) Insertion-Sort:**




aux

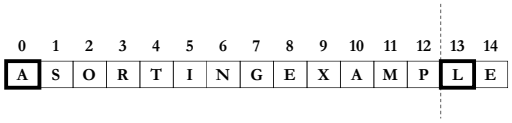
nº movtos:

nº acumulado movtos: 86

88

**b) Shell-Sort:**

Ordenação para h = 13




aux

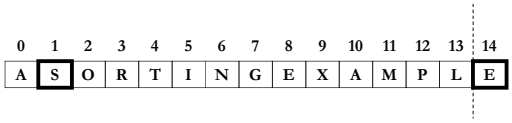
nº movtos: 1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 1 | | | 1 |

89

**b) Shell-Sort:**

Ordenação para h = 13




aux

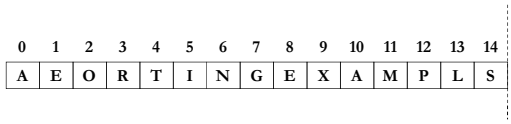
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | | | 4 |

90

## b) Shell-Sort:

Fim da ordenação para h = 13




aux

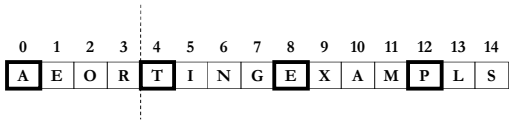
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | | | 4 |

91

## b) Shell-Sort:

Ordenação para h = 4




aux

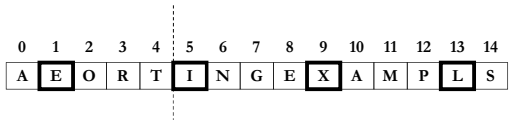
nº movtos: 1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 1 | | 5 |

92

## b) Shell-Sort:

Ordenação para h = 4




aux

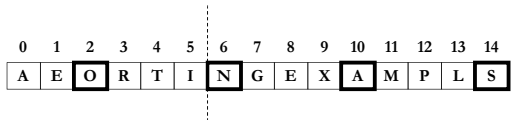
nº movtos: 1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 2 | | 6 |

93

## b) Shell-Sort:

Ordenação para h = 4




aux

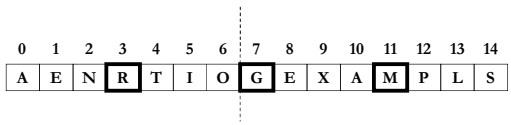
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 5 | | 9 |

94

## b) Shell-Sort:

Ordenação para h = 4




aux

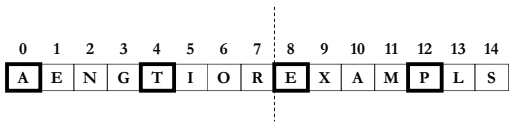
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 8 | | 12 |

95

## b) Shell-Sort:

Ordenação para h = 4




aux

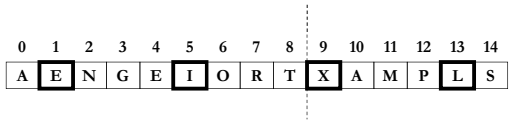
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 11 | | 15 |

96

**b) Shell-Sort:**

Ordenação para h = 4




aux

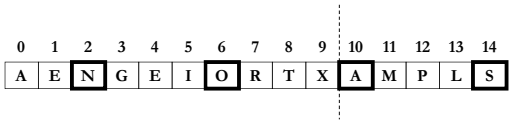
nº movtos: 1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 12 | | 16 |

97

**b) Shell-Sort:**

Ordenação para h = 4




aux

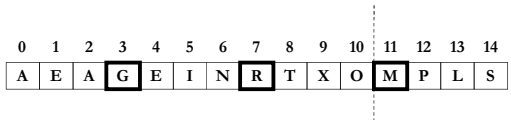
nº movtos: 4

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 16 | | 20 |

98

**b) Shell-Sort:**

Ordenação para h = 4




aux

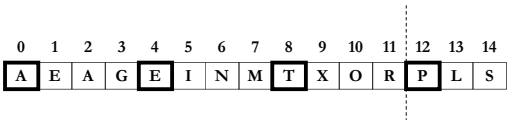
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 19 | | 23 |

99

**b) Shell-Sort:**

Ordenação para h = 4




aux

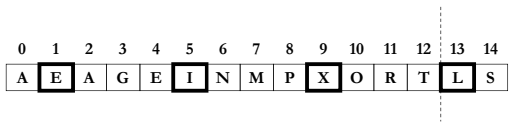
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 22 | | 26 |

100

**b) Shell-Sort:**

Ordenação para h = 4




aux

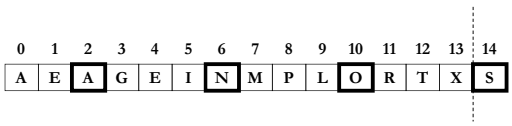
nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 25 | | 29 |

101

**b) Shell-Sort:**

Ordenação para h = 4




aux

nº movtos: 1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | | 30 |

102

**b) Shell-Sort:**

Fim da ordenação para h = 4

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | E | A | G | E | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | | 30 |

103

**b) Shell-Sort:**

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | E | A | G | E | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 1 | 31 |

104

**b) Shell-Sort:**

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | E | A | G | E | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 4 | 34 |

105

**b) Shell-Sort:**

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | G | E | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

1

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 5 | 35 |

106

**b) Shell-Sort:**

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | G | E | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 8 | 38 |

107

**b) Shell-Sort:**

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | E | G | I | N | M | P | L | O | R | T | X | S |

aux

nº movtos

5

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 13 | 43 |

108

## b) Shell-Sort:

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | E | G | I | M | N | P | L | O | R | T | X | S |

aux

nº movtos: 6

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 19 | 49 |

109

## b) Shell-Sort:

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | E | G | I | L | M | N | P | O | R | T | X | S |

aux

nº movtos: 3

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 22 | 52 |

110

## b) Shell-Sort:

Ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | E | G | I | L | M | N | O | P | R | T | X | S |

aux

nº movtos: 7

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 29 | 59 |

111

## b) Shell-Sort:

Fim da ordenação para h = 1 (Insertion-Sort)

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| A | A | E | E | G | I | L | M | N | O | P | R | S | T | X |

aux

nº movtos:

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 29 | 59 |

112

## Insertion-Sort: Shell-Sort:

nº acumulado de movtos: 86

| nº acumulado de movtos | | | |
|---|---|---|---|
| h = 13 | h = 4 | h = 1 | total |
| 4 | 26 | 29 | 59 |

- No Shell-Sort, o número de movimentos para **h = 1** é 29
- **(86-29=57) 57** movimentos do Insertion-Sort foram compensados com (59-29=30) 30 movimentos do Shell-Sort com **h > 1**.

113

## Observações:

- Para vetor inicialmente **já ordenado**, Shell-Sort é pior que Insertion-Sort, devido ao número de **comparações**
- **Outros métodos** simples de ordenação poderiam ser usados para cada valor de **h**, mas o mais usado é o **Insertion-Sort**.
- Há quem use um método para **h > 1** e outro para **h = 1**

114

**Observações:**

- Referências bibliográficas indicam que o algoritmo anterior é **$O(n (\log n)^2)$** e também **$O(n^{1.5})$**.
- Donald Knuth prova que, mesmo usando apenas **2 valores de h**, a saber,

  $$(16\,n / \pi)^{1/3} \quad \text{e} \quad 1$$

  O método resultante é **$O(n^{5/3})$**, o que é melhor que o pior caso do Insertion-Sort, que é **$O(n^2)$**.
- O tempo de execução do algoritmo apresentado **não** é muito **sensível** ao **estado inicial** do vetor, ao contrário do **Insertion-Sort** que é **O(n)** para vetor já ordenado e **$O(n^2)$** para vetor ordenado inversamente.

115