Algoritmos de Ordenação

Estrutura de Dados II

Prof. João Dallyson

Email: Joao.dallyson@ufma.br

Ordenação

- **Objetivo:**
  - Rearranjar os itens de um vetor ou lista de modo que suas chaves estejam ordenadas de acordo com alguma regra
- **Estrutura:**

```
typedef int chave_t;
struct item {
     chave_t chave;
     /* outros componentes */
};
```

FONTE: [Cunha, 2012]

Introdução

- **Ordenar:** processo de rearranjar um conjunto de objetos em uma ordem ascendente ou descendente**.**

- **Objetivo:** facilitar a recuperação posterior de itens do conjunto ordenado.
  - Dificuldade de se utilizar um catálogo telefônico se os nomes das pessoas não estivessem listados em ordem alfabética.

- **A maioria dos métodos de ordenação é baseada em comparações das chaves.**

- **Existem métodos de ordenação que utilizam o princípio da distribuição.**

Conceitos Básicos

- **Exemplo de ordenação por distribuição: considere o problema de ordenar um baralho com 52 cartas na ordem:**

$$A < 2 < 3 < \cdots < 10 < J < Q < K$$

e

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit.$$

- **Algoritmo:**
  - 1. Distribuir as cartas abertas em treze montes: ases, dois, três, . . ., reis.
  - 2. Colete os montes na ordem especificada.
  - 3. Distribua novamente as cartas abertas em quatro montes: paus, ouros, copas e espadas.
  - 4. Colete os montes na ordem especificada.

Conceitos Básicos

- Métodos como o ilustrado são também conhecidos como ordenação digital, radixsort ou bucketsort.
- O método não utiliza comparação entre chaves.
- Uma das dificuldades de implementar este método está relacionada com o problema de lidar com cada monte.
- Se para cada monte nós reservarmos uma área, então a demanda por memória extra pode tornar-se proibitiva.
- O custo para ordenar um arquivo com n elementos é da ordem de O(n).

Conceitos Básicos

- **Notação utilizada nos algoritmos:**
  - Os algoritmos trabalham sobre os registros de um arquivo.
  - Cada registro possui uma **chave** utilizada para controlar a ordenação.
  - Podem existir outros componentes em um registro.

- Qualquer tipo de chave sobre a qual exista uma regra de ordenação bem definida pode ser utilizada.

- Um método de ordenação é estável se a ordem relativa dos itens com chaves iguais não se altera durante a ordenação.

- Alguns dos métodos de ordenação mais eficientes não são estáveis.

Conceitos Básicos

- **Classificação dos métodos quanto a localização:**
  - Ordenação interna: arquivo a ser ordenado cabe todo na memória principal (RAM).
  - Ordenação externa: arquivo a ser ordenado não cabe na memória principal. Dados armazenados em memória secundária.

- **Diferenças entre os métodos:**
  - Em um método de ordenação interna, qualquer registro pode ser imediatamente acessado.
  - Em um método de ordenação externa, os registros são acessados sequencialmente ou em grandes blocos.

Classificação: Estabilidade

- Método é *estável se a ordem relativa dos registros com* a mesma chave não se altera após a ordenação

Entrada

Adams	A
Black	B
Brown	D
Jackson	B
Jones	D
Smith	A
Thompson	D
Washington	B
White	C
Wilson	C

Não estável

Adams	A
Smith	A
Washington	B
Jackson	B
Black	B
White	C
Wilson	C
Thompson	D
Brown	D
Jones	D

estável

Adams	A
Smith	A
Black	B
Jackson	B
Washington	B
White	C
Wilson	C
Brown	D
Jones	D
Thompson	D

Classificação: Uso de Memória

- Métodos de **ordenação in situ** (utilizam vetor e realizam a permutação no próprio vetor) são os preferidos.

- Métodos que utilizam **listas encadeadas** não são muito utilizados.

- Métodos que armazenam cópias dos itens a serem ordenados são importantes porque executam em tempo linear na prática.

Ordenação Interna

- **Classificação dos métodos de ordenação interna:**
  - Métodos simples:
    - Adequados para pequenos arquivos.
    - Requerem $O(n^2)$ comparações.
    - Produzem programas pequenos.
  - Métodos eficientes:
    - Adequados para arquivos maiores.
    - Requerem $O(n \log n)$ comparações.
    - Usam menos comparações.
    - As comparações são mais complexas nos detalhes.
    - Métodos simples são mais eficientes para pequenos arquivos.

Critérios de Avaliação

- **Na escolha de um algoritmo de ordenação interna deve ser considerado o tempo gasto pela ordenação.**
- **Sendo n o número registros no arquivo, as medidas de complexidade relevantes são:**
  - Número de comparações C(n) entre chaves.
  - Número de movimentações M(n) de itens do arquivo.

```
for(i = 0; i < n-1; i++) {
   for(j = n-1; j > i; j--) {
      if(v[j-1].chave > v[j].chave) /* comparações */
         troca(v[j-1], v[j]); /* trocas */
   }
}
```

Vídeos e Animações

- **Vídeos:**
  - https://www.youtube.com/user/AlgoRythmics
- **Animações:**

https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html

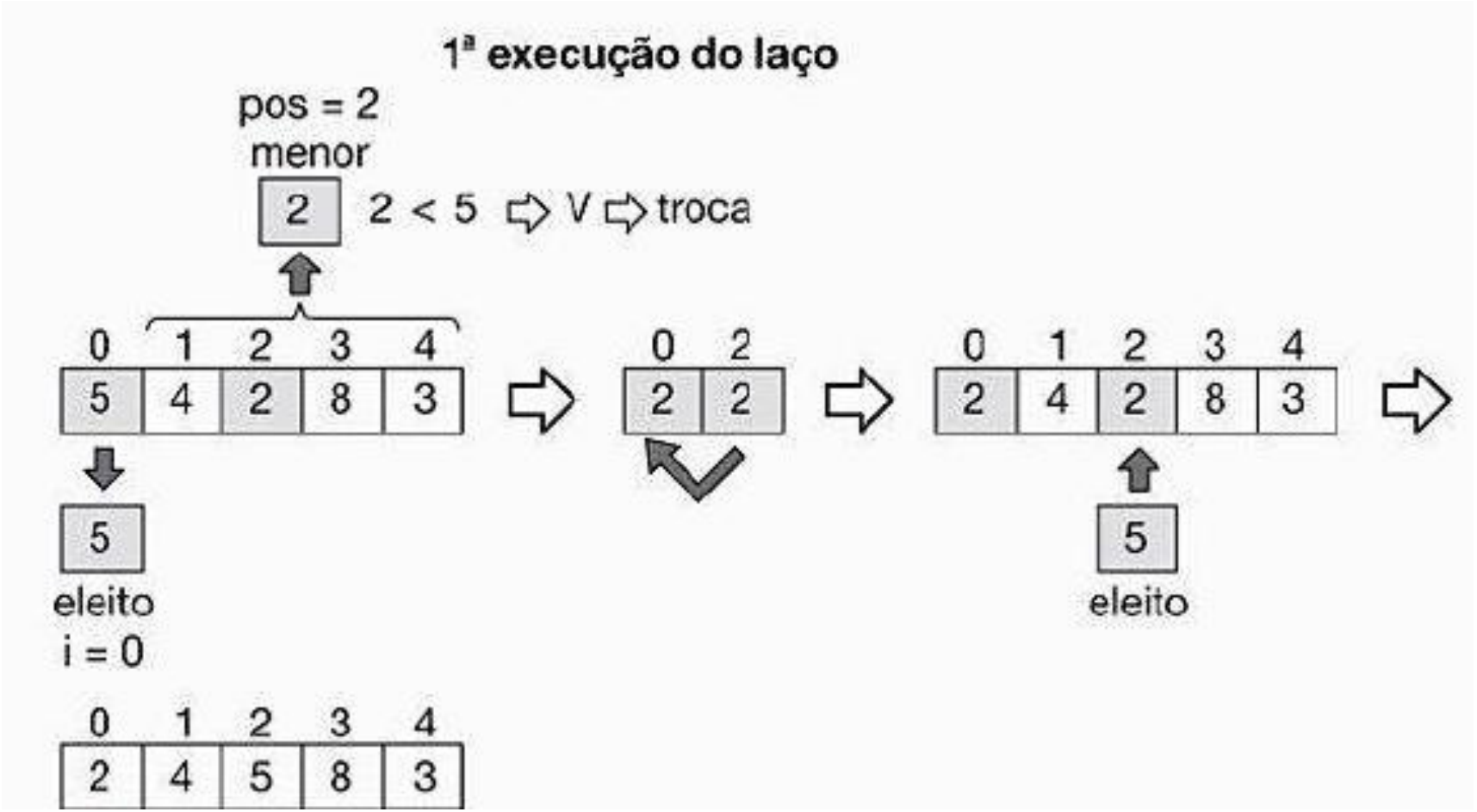
http://visualgo.net/

https://www.bluffton.edu/~nesterd/java/SortingDemo.html

Ordenação por Seleção

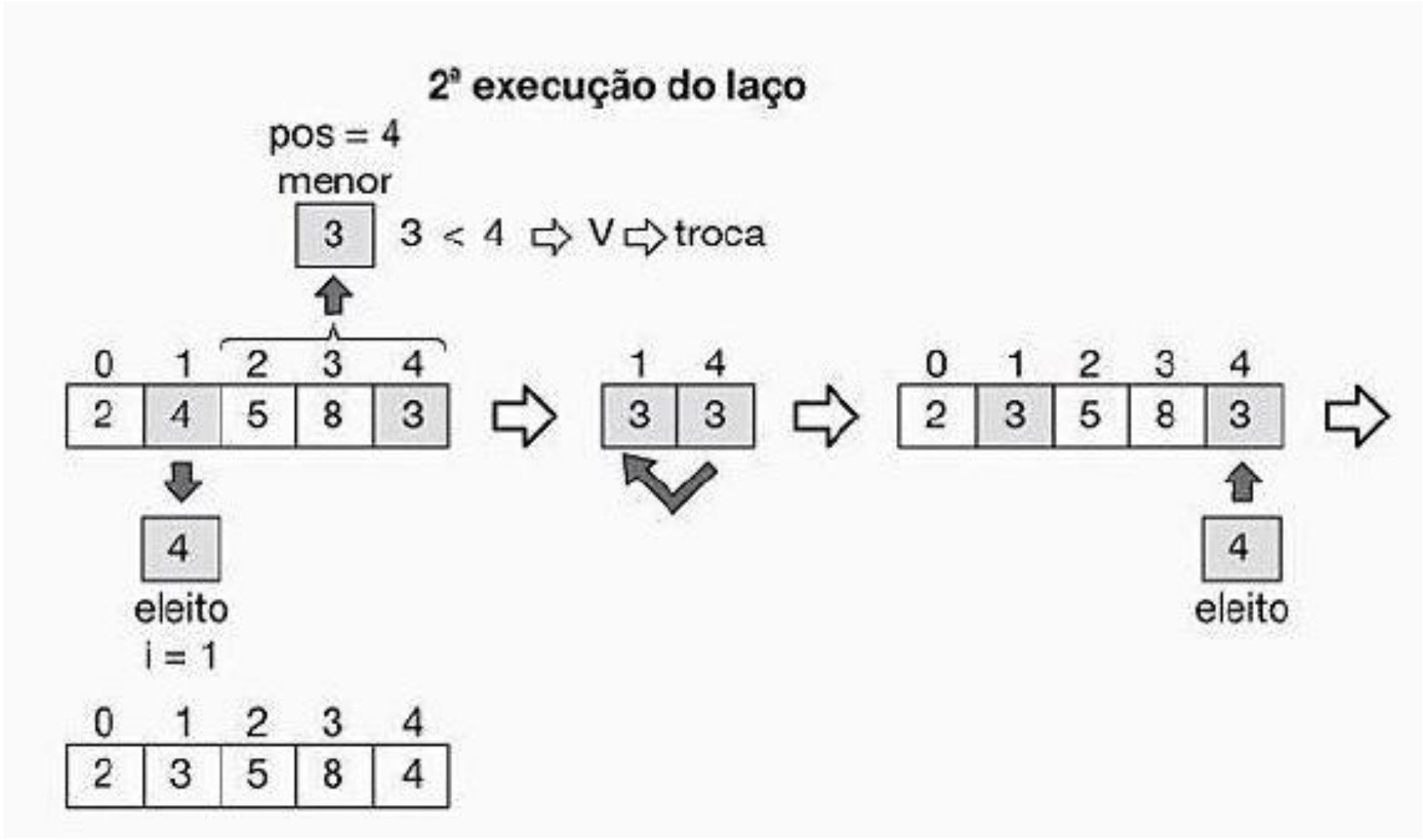
- **Um dos algoritmos mais simples de ordenação.**
- **Algoritmo:**
  - Selecione o menor item do vetor.
  - Troque-o com o item da primeira posição do vetor.
  - Repita essas duas operações com os n − 1 itens restantes, depois com os n − 2 itens, até que reste apenas um elemento.

Ordenação por Seleção



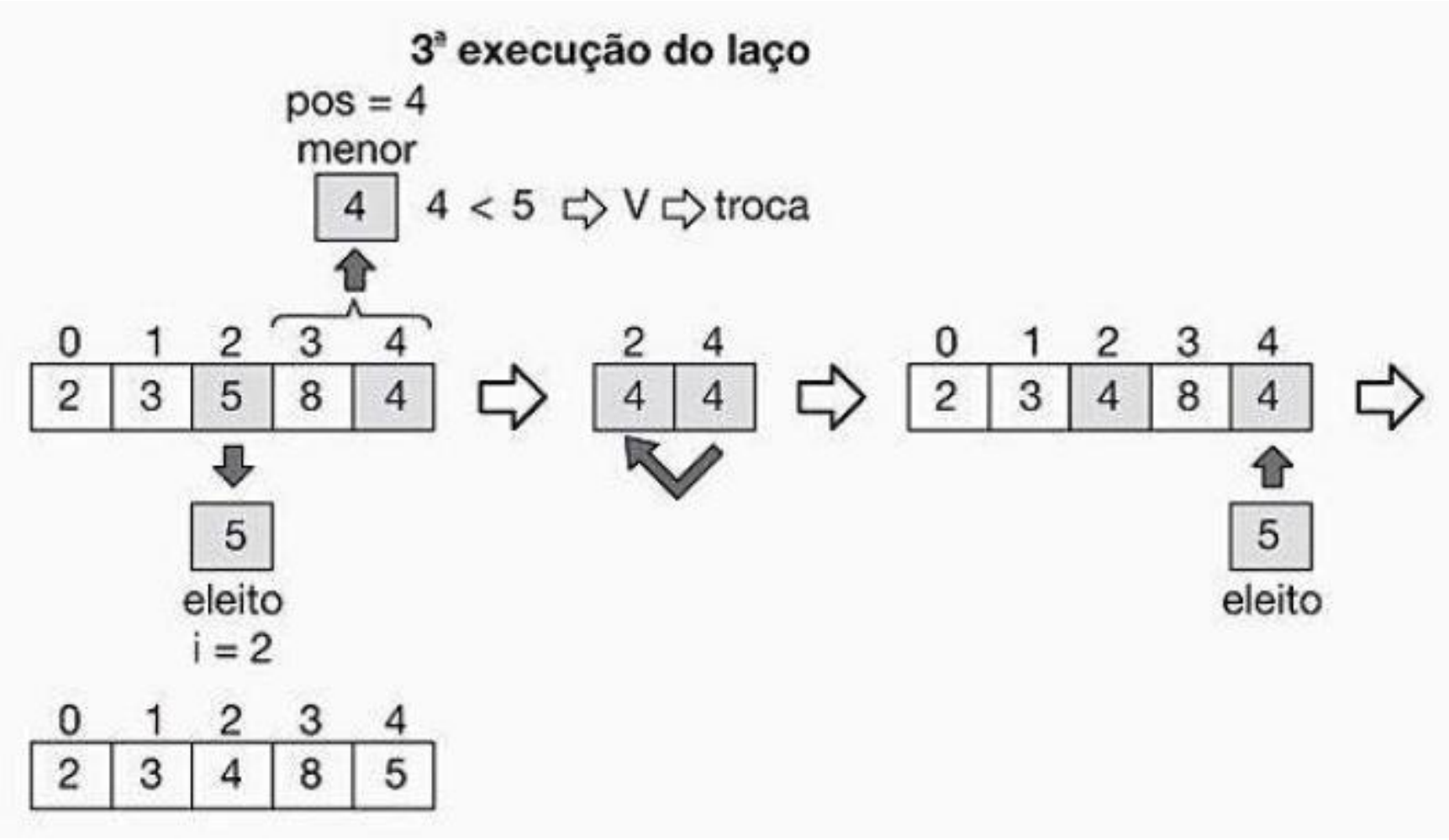
FONTE: [ASCENCIO, 2012]

Ordenação por Seleção



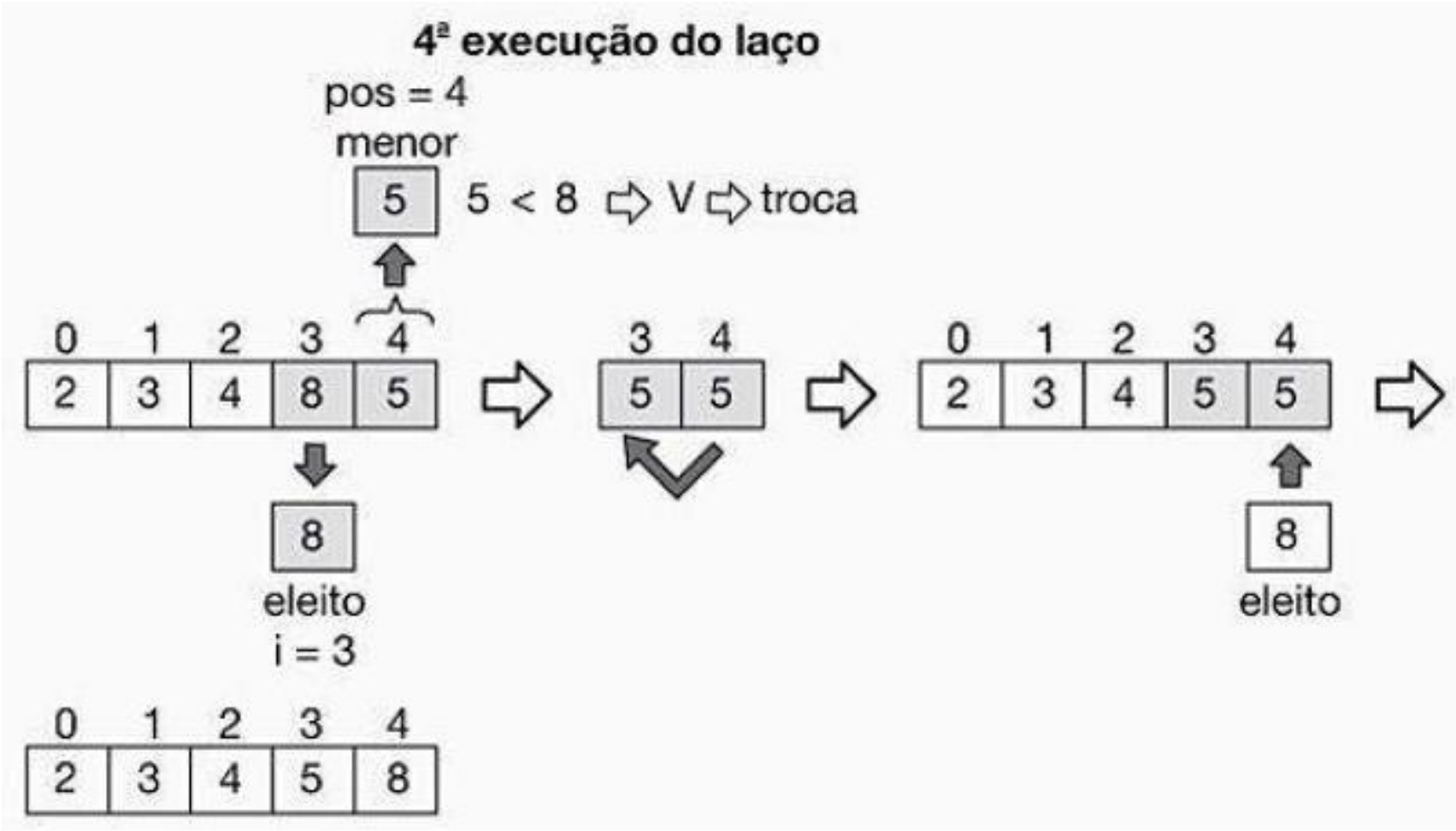
FONTE: [ASCENCIO, 2012]

Ordenação por Seleção



FONTE: [ASCENCIO, 2012]

Ordenação por Seleção



FONTE: [ASCENCIO, 2012]

Ordenação por Seleção

```
public static int[] SelectionSort(int [] A) {

    for (int i = 0; i < A.length -1 ; i++) {
        int min = i;
        for (int j = i+1 ; j < A.length ; j++)
            if (A[j] < A[min])
                min = j;

        // Realiza a Troca
        int temp = A[min];
        A[min] = A[i];
        A[i] = temp;
    }

    return A;
}
```

Ordenação por Seleção: complexidade

- **Análise**
  - Comparações:
  - Somatório de termos:
    - 1 + 2 + 3….+n-1

$$T(n) = \frac{(a_1 + a_n).n}{2}$$

$$T(n) = \frac{(1 + n - 1).(n - 1)}{2}$$

$$T(n) = \frac{n.(n - 1)}{2}$$

$$T(n) = \frac{n^2 - n}{2}$$

$$T(n) = \frac{n^2}{2} - \frac{n}{2}.$$

  - Movimentações:

$$M(n) = 3(n - 1)$$

  - A atribuição min = j é executada em média n log n vezes, Knuth (1973).

Ordenação por Seleção

- **Vantagens**
  - Custo linear no tamanho da entrada para o número de movimentos de registros.
  - É o algoritmo a ser utilizado para arquivos com registros muito grandes.
  - É muito interessante para arquivos pequenos.
- **Desvantagens:**
  - O fato de o arquivo já estar ordenado não ajuda em nada, pois o custo continua quadrático.
  - O algoritmo não é **estável**.

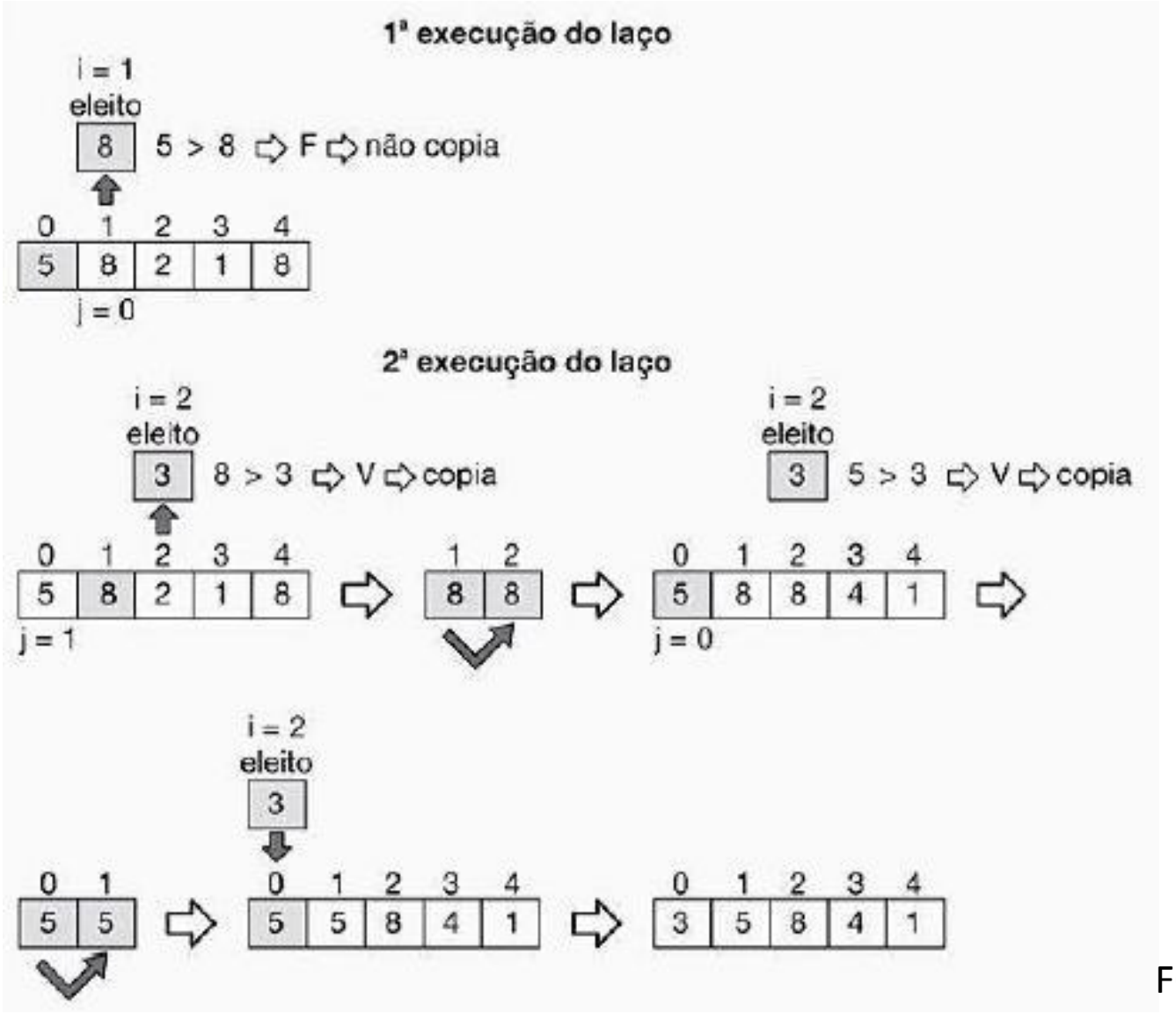
Exercício

1) **Suponha que você tenha a seguinte lista para ordenar [11, 7, 12, 19, 1, 6, 18, 8, 20] qual será a representação parcial da lista após quatro passos completos do algoritmo de ordenação por seleção?**
2) **Mostre um exemplo de entrada que demonstra que o método de ordenação seleção não é estável.**

Ordenação por Inserção

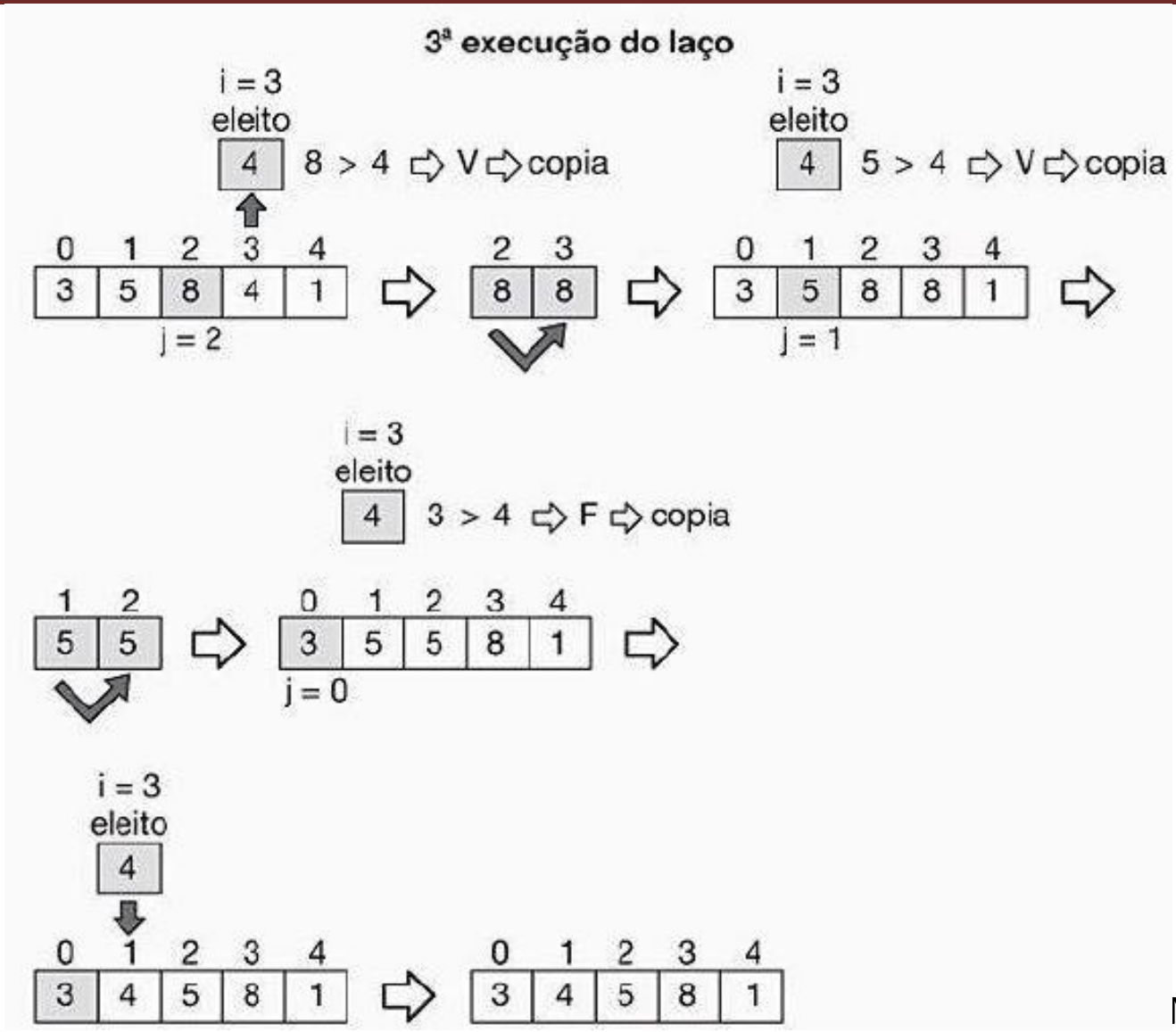
- **Método preferido dos jogadores de cartas.**
  - Algoritmo:
    - Em cada passo a partir de i=2 faça:
      - Selecione o i-ésimo item da seqüência fonte.
      - Coloque-o no lugar apropriado na seqüência destino de acordo com o critério de ordenação.
    - Os elementos à esquerda do número eleito estão sempre ordenados de forma crescente ou decrescente

Ordenação por Inserção



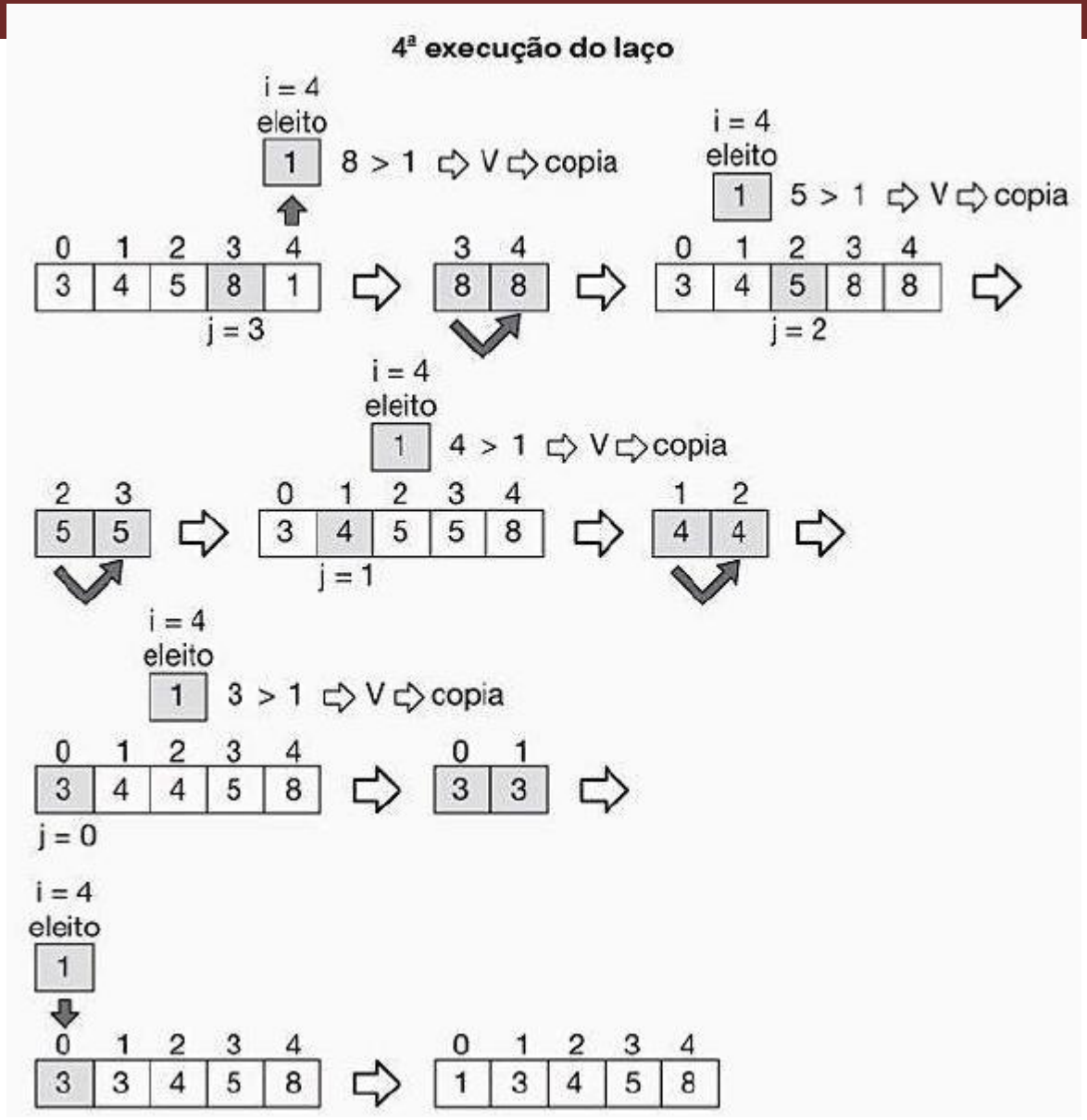
FONTE: [ASCENCIO, 2012]

Ordenação por Inserção



NTE: [ASCENCIO, 2012]

Ordenação por Inserção



FONTE: [ASCENCIO, 2012]

Ordenação por Inserção

```
public static int[] InsertionSort(int [] A) {
    int i, j, chave;

    for (j = 1; j < A.length; j++) {

        chave = A[j];

        i = j - 1;
        while ( i>= 0 && A[i] > chave){
            A[i+1] = A[i];
            i--;
        }
        A[i+1] = chave;
    }

    return A;
}
```

Ordenação por Inserção

- **Análise**
  - Seja C(n) a função que conta o número de comparações.
  - No anel mais interno, na i-ésima iteração, o valor de Ci é:

$$\begin{aligned}
&\text{melhor caso} && : C_i(n) = 1 \\
&\text{pior caso} && : C_i(n) = i \\
&\text{caso médio} && : C_i(n) = \tfrac{1}{i}(1 + 2 + \cdots + i) = \tfrac{i+1}{2}
\end{aligned}$$

Ordenação por Inserção

- **Assumindo que todas as permutações de n são igualmente prováveis no caso médio, temos:**

$$\text{melhor caso} : C(n) = (1 + 1 + \cdots + 1) = n - 1$$

$$\text{pior caso} : C(n) = (2 + 3 + \cdots + n) = \frac{n^2}{2} + \frac{n}{2} - 1$$

$$T(n) = 2 + 3 + 4 + \ldots + n$$

$$T(n) = (\sum_{i=1}^{n} i) - 1$$

$$T(n) = \frac{(1+n)n}{2} - 1$$

$$T(n) = \frac{n^2 + n}{2} - 1$$

$$T(n) = O(n^2), \text{para } c = 2, n \geq 1.$$

$$\text{caso médio} : C(n) = \frac{1}{2}(3 + 4 + \cdots + n + 1) = \frac{n^2}{4} + \frac{3n}{4} - 1$$

Ordenação por Inserção

- O número mínimo de comparações e movimentos ocorre quando os itens estão originalmente em ordem.
- O número máximo ocorre quando os itens estão originalmente na ordem reversa.
- É o método a ser utilizado quando o arquivo está “quase” ordenado.
- É um bom método quando se deseja adicionar uns poucos itens a um arquivo ordenado, pois o custo é linear.
- O algoritmo de ordenação por inserção é **estável**.

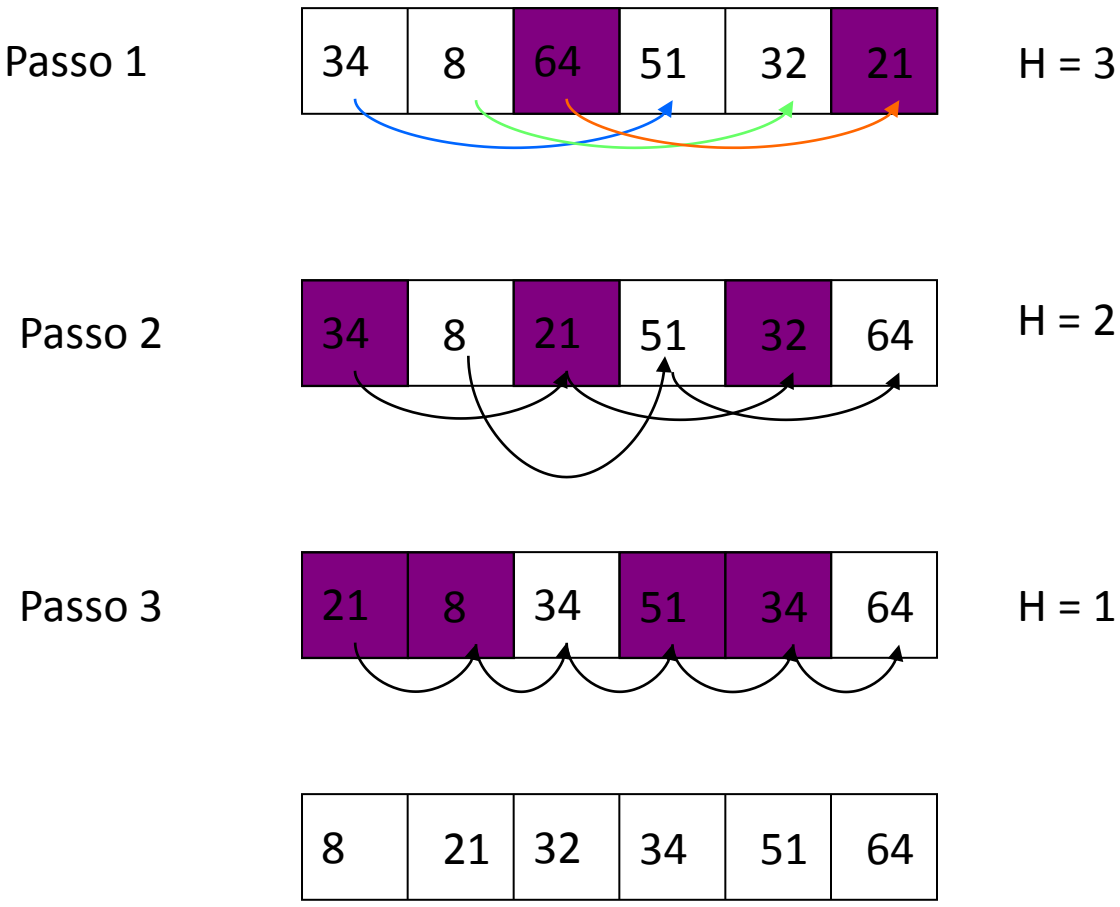
Shellsort

- **Proposto por Shell em 1959.**
- **É uma extensão do algoritmo de ordenação por inserção.**
- **Problema com o algoritmo de ordenação por inserção:**
  - Troca itens adjacentes para determinar o ponto de inserção.
  - São efetuadas n − 1 comparações e movimentações quando o menor item está na posição mais à direita no vetor.
- **O método de Shell contorna este problema permitindo trocas de registros distantes uns dos outros.**

Shellsort

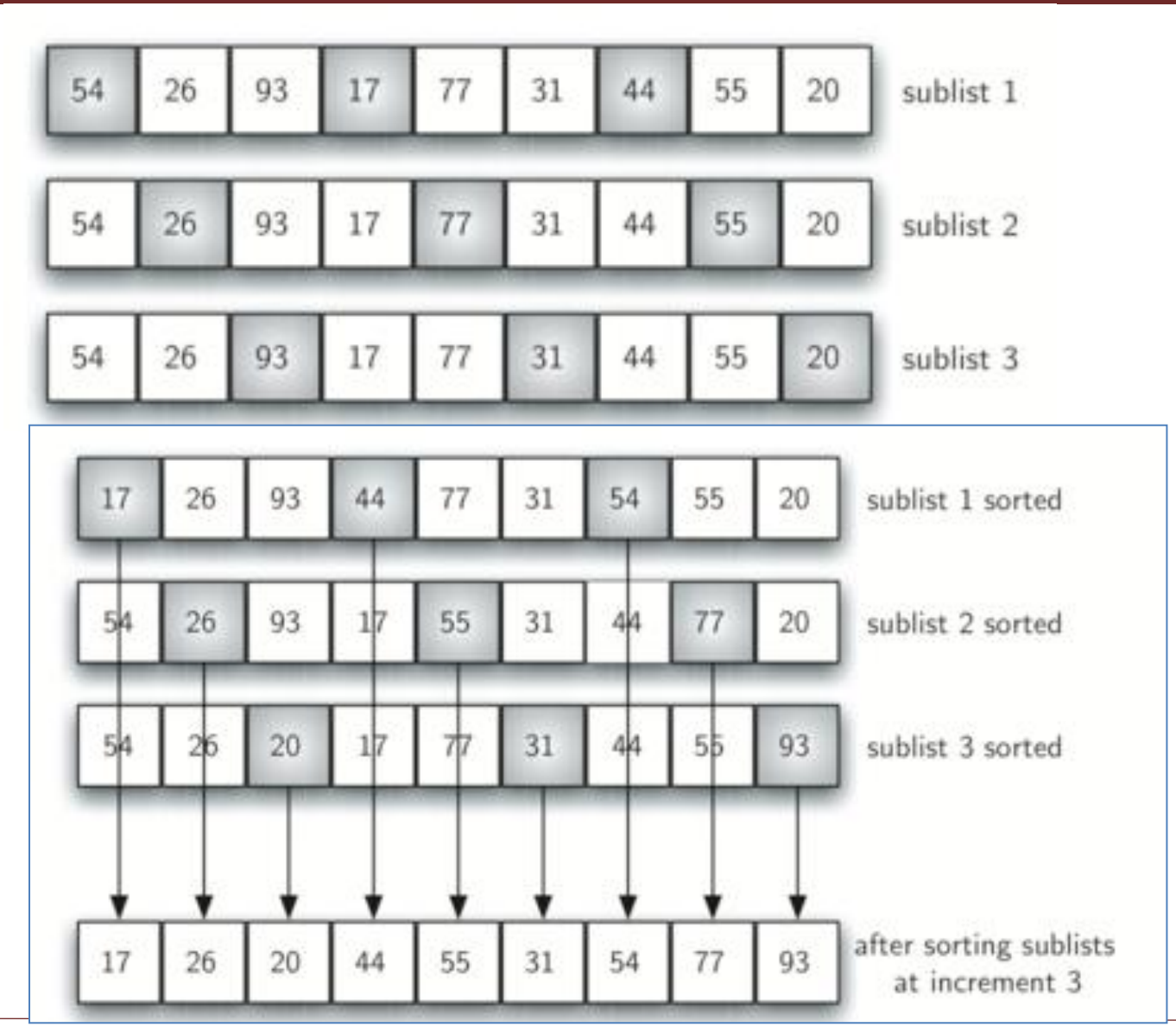
- **Os itens separados de h posições são rearranjados.**
- **Todo h-ésimo item leva a uma sequencia ordenada.**
- **Tal sequencia é dita estar h-ordenada.**
  - Baseado na diminuição dos incrementos
    - uma sequencia:
      - $h_1, h_2, h_3, ..., h_t$
      - 9, 5, 3, 2, 1
    - A única regra é que o último deve ser 1
- **Qualquer sequência terminando com h = 1 garante ordenação correta (h = 1 é ordenação por inserção)**

Shellsort


Passo 1
34 8 64 51 32 21
H = 3
Passo 2
34 8 21 51 32 64
H = 2
Passo 3
21 8 34 51 34 64
H = 1
8 21 32 34 51 64

Shellsort

- **H = 3**



Shellsort

- **Como escolher o valor de h:**
  - Seqüência para h:

$$h(s) = 3h(s-1) + 1, \quad para\ s > 1$$

$$h(s) = 1, \quad para\ s = 1.$$

  - Knuth (1973, p. 95) mostrou experimentalmente que esta seqüência é difícil de ser batida por mais de 20% em eficiência.
  - A seqüência para h corresponde a 1, 4, 13, 40, 121, 364, 1.093, 3.280, . . .
    - Outras sequências têm desempenho similar

Shellsort

```
public static int[] ShellSort(int [] A) {
    int h, temp, j;
    /** Calcula o valor de h inicial*/
    for (h = 1 ; h < A.length; h = (3*h)+1);

        while (h > 0) {
            h = (h-1)/3; //Atualiza o valor de H

            for (int i = h; i < A.length; i++) {
                temp = A[i];
                j = i;

                /** Efetua comparações entre elementos com distância H*/
                while (A[j-h] > temp) {
                    A[j] = A[j-h];
                    j = j - h;
                    if (j < h) break;
                }
                A[j] = temp;

            }
        }

    return A;
}
```

Shellsort

- **Análise**
  - A razão da eficiência do algoritmo ainda não é conhecida.
  - Ninguém ainda foi capaz de analisar o algoritmo.
  - A sua análise contém alguns problemas matemáticos muito difíceis.
  - A começar pela própria seqüência de incrementos.
  - O que se sabe é que cada incremento não deve ser múltiplo do anterior.
  - Deduções referente ao número de comparações para a seqüência de Knuth:

$$Conjetura\ 1 \quad : C(n) = O(n^{1,25})$$

$$Conjetura\ 2 \quad : C(n) = O(n(\ln n)^2)$$

Shellsort

- **Vantagens:**
  - Shellsort é uma ótima opção para arquivos de tamanho moderado.
  - Sua implementação é simples e requer uma quantidade de código pequena.
- **Desvantagens:**
  - O tempo de execução do algoritmo é sensível à ordem inicial do arquivo.
  - O método não é **estável**.

Exercício

1) Dada a seguinte lista de números: [5,16,20,12,3,8,9,17,19,7] qual será o conteúdo da lista após todas as trocas completas para um h = 3

Referências

Básica

- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: Teoria e Prática. Editora Campus, 2002
- Ziviani, N. Projeto de Algoritmos Com Implementações em Pascal e C, Cengage Learning, 2004.

Complementar

- TENENBAUM, Aaron; LANGSAM, Yedidyah; AUGENSTEIN, Moshe J. *Estruturas de dados usando C. São Paulo: Makron Books, 1995. ISBN: 9788534603485*
- ASCENCIO, Ana Fernanda Gomes; ARAUJO, Graziela Santos. Estruturas de Dados: Algoritmos, análise da complexidade e implementações em Java e C/C++. Pearson Prentice Hall, 2010
- DROZDEK, Adam. Adam Drozdek. Data Structures and Algorithms in Java. 2. Cengage Learning. 2004. 2. Cengage Learning. 2004
- GOODRICH, Michael T. Estruturas de dados e algoritmos em java. 4 ED. Porto Alegre: Bookman, 2007. 600.
- SKIENA, Steven S.. **The Algorithm Design Manual**. 2. Springer-Verlag. 2008
- Notas de aula: prof. Ítalo Cunha – UFMG. 2012.

Perguntas....