Algoritmos de Ordenação

Algoritmos e Programação 2
Prof. Dr. Anderson Bessa da Costa
Universidade Federal de Mato Grosso do Sul

Problema de Ordenação

- Pré-processamento em muitas aplicações
- Fácil comparar dois algoritmos quando aplicados a uma mesma massa de dados
- Entretanto, massas de dados modelam situações
  - Particularidades
  - Alguns métodos se comportam melhor que outros, independente de sua avaliação em testes genéricos

Problema de Ordenação

- Um vetor $v[1..n]$ é crescente se $v[1] \leq v[2] \leq \ldots \leq v[n]$. O problema da ordenação de um vetor consiste no seguinte:

Rearranjar (ou seja, permutar) os elementos de um vetor $v[1..n]$ de tal modo que ele se torne crescente

Ordenação Bolha (*Bubble Sort*)

Ordenação Bolha

- Simples
- Mais difundido
- **Ideia:** Uma iteração percorre a tabela do início até o fim, sem interrupção
  - Troca dois elementos consecutivos se estiverem fora de ordem
  - Ao término, maior elemento estará na última posição
- Na segunda iteração, o segundo maior elemento é posicionado, e assim sucessivamente …
- Processamento é repetido $n - 1$ vezes

Visualização Iteração Ordenação Bolha

0	1	2	3	4	5	6	
40	37	95	42	39	51	60	Tabela inicial
40	**37**	95	42	39	51	60	i = 0 j = 0
37	**40**	**95**	42	39	51	60	i = 0 j = 1
37	40	**95**	**42**	39	51	60	i = 0 j = 2
37	40	42	**95**	**39**	51	60	i = 0 j = 3
37	40	42	39	**95**	**51**	60	i = 0 j = 4
37	40	42	39	51	**95**	**60**	i = 0 j = 5
37	40	42	39	51	60	**95**	i = 0 j = 6

Passo a passo de uma iteração de ordenação bolha.

Visualização Ordenação Bolha

0	1	2	3	4	5	6	
40	37	95	42	39	51	60	Tabela inicial
37	40	42	39	51	60	95	após 1ª iteração (i = 0)
37	40	39	42	51	60	95	após 2ª iteração (i = 1)
37	39	40	42	51	60	95	após 3ª iteração (i = 2)
37	39	40	42	51	60	95	após 4ª iteração (i = 3)

Um exemplo de ordenação bolha.

Algoritmo Ordenação Bolha

```c
void bubble_sort(int v[], int n) {
    int i, j, aux;

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

Complexidade Ordenação Bolha

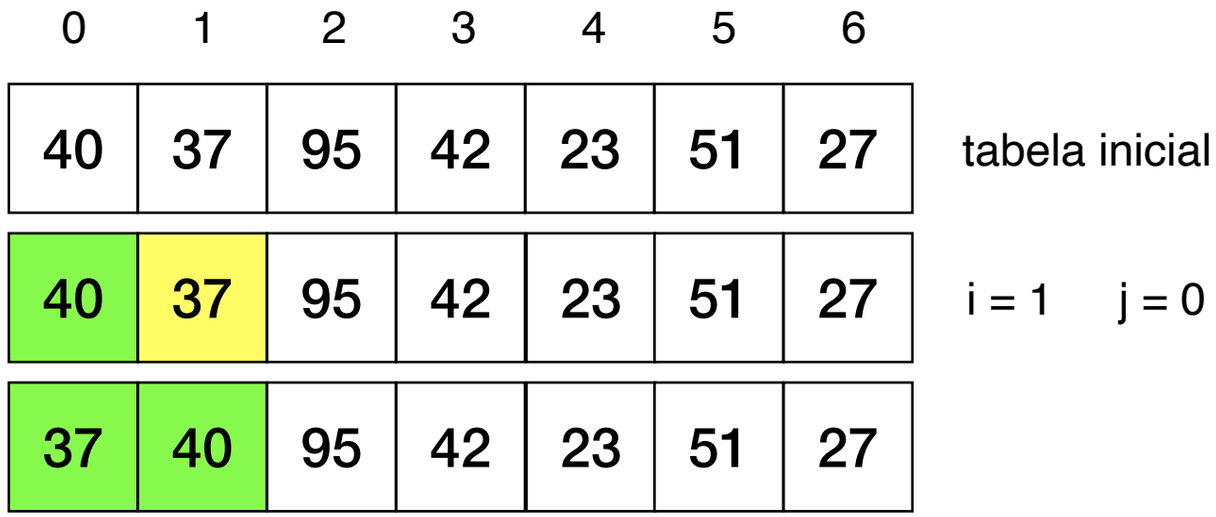
- Complexidade **pior caso** é $O(n^2)$
- Complexidade **melhor caso** é $O(n^2)$
- Complexidades são as mesmas devido aos percursos estipulados para as variáveis i e j
- É possível alterar e melhorar o **melhor caso**?

Ordenação por Inserção (*Insertion Sort*)

Ordenação por Inserção (*Insertion Sort*)

- Simples
- Complexidade equivalente à ordenação bolha
- **Ideia:** Imagine uma tabela ordenada até o i-ésimo elemento
- Ordenação pode ser estendida até o $(i+1)$-ésimo elemento por meio de comparações sucessivas deste $(i+1)$ com os elementos anteriores,
  - isto é, com o i-ésimo elemento, com o $(i-1)$-ésimo elemento etc., procurando sua posição correta na parte da tabela que já está ordenada

Visualização Iteração Ordenação por Inserção



Passo a passo de uma iteração de ordenação por inserção.

Visualização Iteração Ordenação por Inserção

Um exemplo de ordenação por inserção.

Algoritmo Ordenação por Inserção

```c
void insertion_sort(int v[], int n) {
    int i, chave, j;
    for (i = 1; i < n; i++) {
        chave = v[i];
        j = i - 1;

        // Mover os elementos de v[0..i-1], que são maiores que chave,
        // uma posição à frente de sua posição atual
        while (j >= 0 && v[j] > chave) {
            v[j + 1] = v[j];
            j = j - 1;
        }
        v[j + 1] = chave;
    }
}
```

Complexidade Ordenação por Inserção

- Percurso em cada iteração termina exatamente quando a inversão não ocorre
- Então, o algoritmo tem complexidade de melhor caso $O(n)$, que ocorre quando o número de inversões é zero
- No pior caso, quando a tabela está em ordem inversa, são executadas $n-1$ iterações (para $i = 2,\ldots,n$) e, em cada uma delas, executadas $i-1$ inversões. Logo

$$\sum_{i=2}^{n} Inv(i) = 1 + 2 + \ldots + (n-1) = O(n^2)$$

Ordenação por Comparação

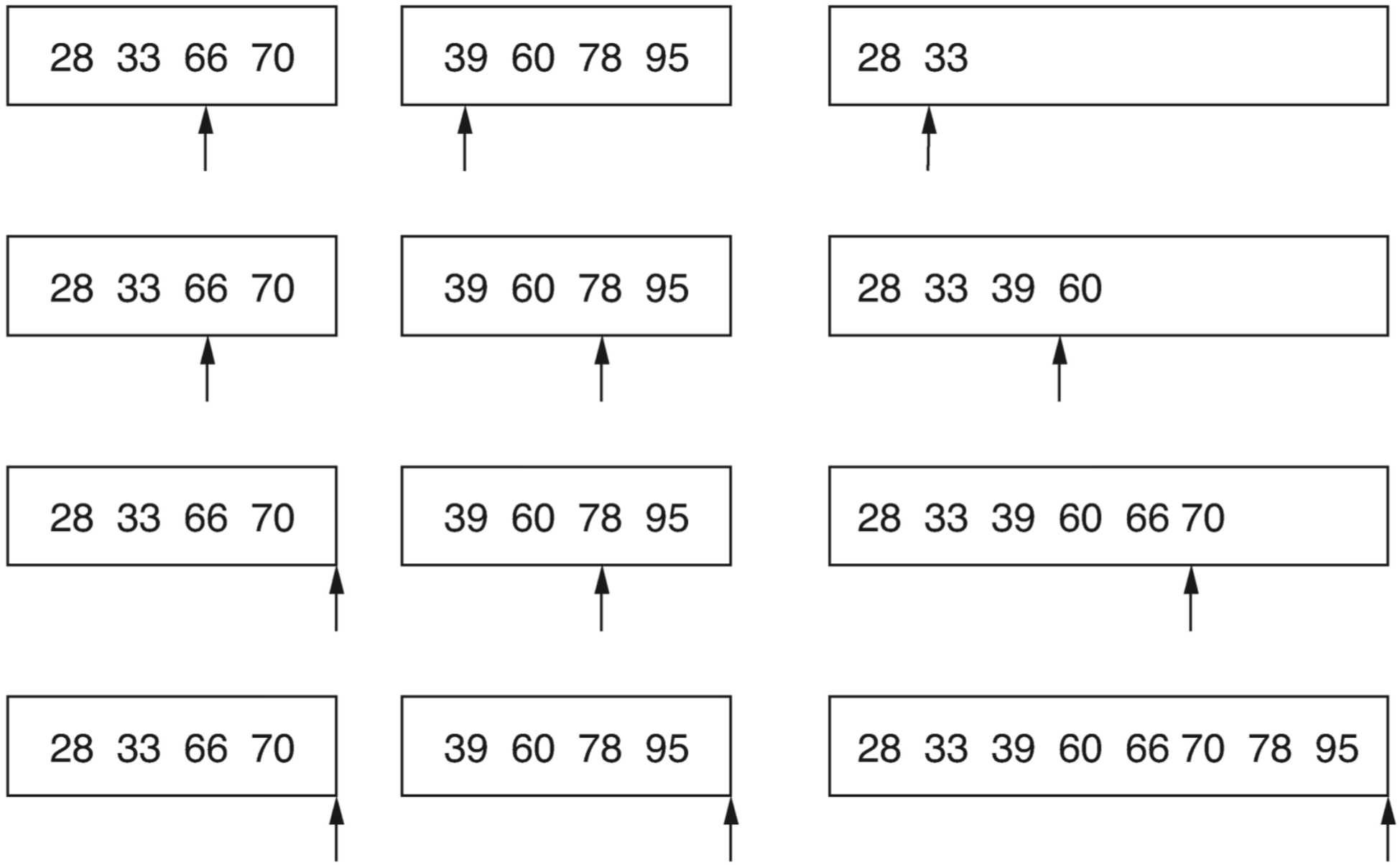
- Apenas comparações entre elementos para ganhar informação de ordem sobre sequência de entrada $\{a_1, a_2, \ldots, a_n\}$.
- Ordenação bolha, ordenação por inserção, Mergesort, Quicksort são algoritmos de comparação por ordenação
- Pior caso é $\Omega(n \log n)$
  - Ou seja, o pior caso sempre irá ter complexidade assintótica limitada inferiormente por $n \log n$ (prova em "Introduction to Algorithms", Cormen)

Ordenação por Intercalação (*Merge sort*)

Ordenação por Intercalação

- Procedimento básico intercalação de listas
- **Ideia:** intercalar as duas metades da lista desejada quando estas já se encontram ordenadas
  - Na realidade, deseja-se então ordenar primeiramente as duas metades, o que pode ser feito utilizando recursivamente o mesmo conceito

Merge sort: Processo de Intercalação



Merge sort: Método de Ordenação

- $\mathscr{L}$ uma lista que se deseja ordenar
- Divide-se $\mathscr{L}$ em duas metades e as ordena
  - Resultado são duas listas ordenadas que podem ser intercaladas
- Para ordenar cada uma das metades o processo considerado é o mesmo, sendo o problema dividido em problemas menores, que são sucessivamente solucionados

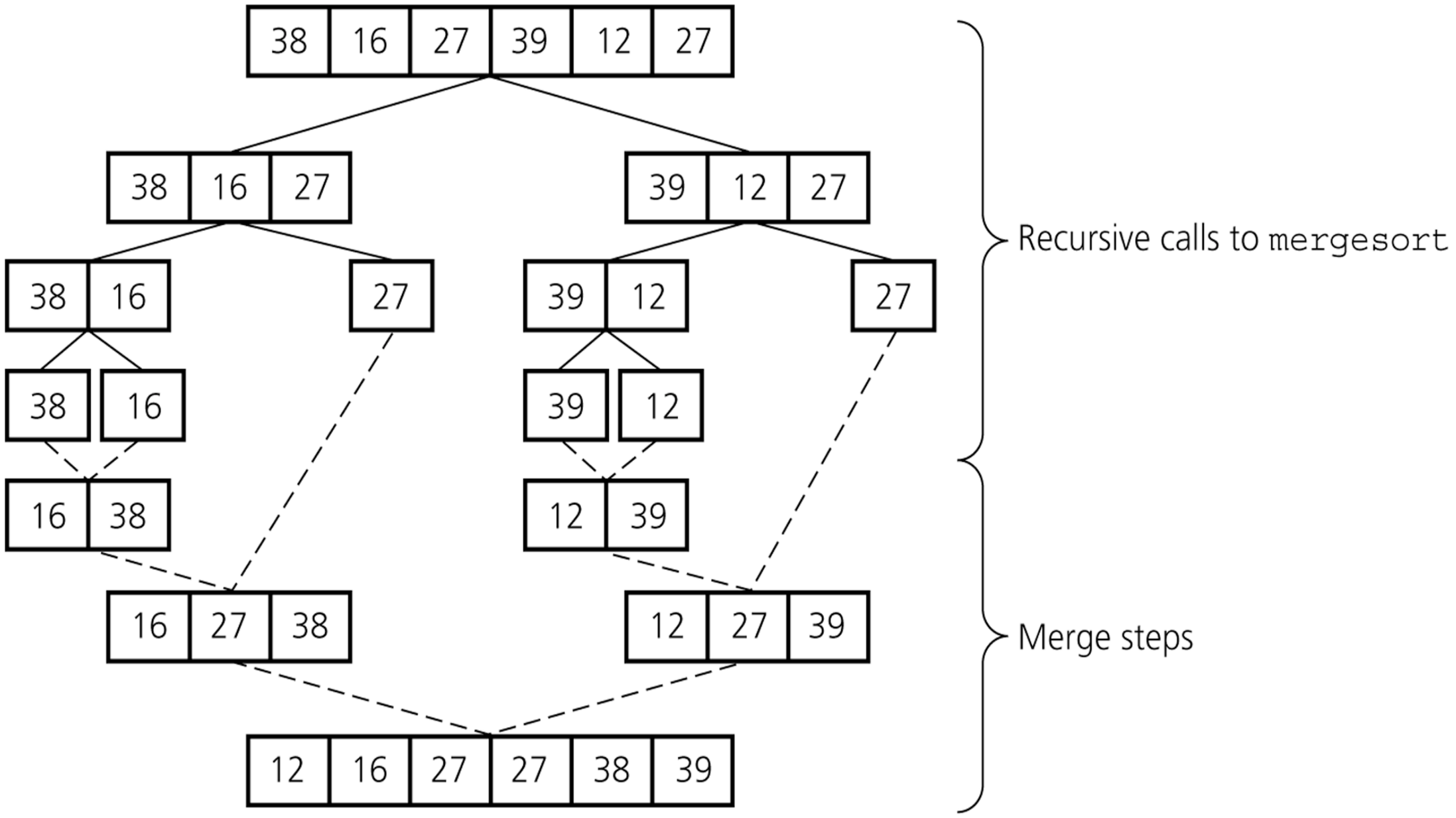
Algoritmo Merge sort

```c
// Função principal de ordenação usando o Merge Sort
void mergesort(int l[], int esq, int dir) {
    if (esq < dir) {
        int centro = (dir + esq) / 2;

        // Ordena as duas metades
        mergesort(l, esq, centro);
        mergesort(l, centro + 1, dir);

        // Intercala as duas metades ordenadas
        intercala(l, esq, centro, dir);
    }
}
```

Exemplo Merge sort



Complexidade Merge sort

- Complexidade de **pior caso** $O(n \log n)$
- Entretanto, mesmo com este bom resultado a ordenação por intercalação não é um dos métodos de ordenação mais empregados
- Sua eficiência depende da cuidada implementação da tabela temporária

Referências

- SZWARCFITER, Jayme Luiz; MARKENZON, Lilian. Estruturas de Dados e seus Algoritmos. Edição: 3a. Editora: LTC. 2010
- ZIVIANI, Nivio. Projeto de Algoritmos com Implementações em Pascal e C (3a. edição). Editora Cengage Learning, 2010.