16

Pesquisa e classificação

OBJETIVOS

- Neste capítulo, você aprenderá:
- Como procurar um dado valor em um array utilizando pesquisa linear e pesquisa binária.
- Como classificar arrays utilizando a seleção iterativa e algoritmos de classificação por inserção.
- Como classificar arrays utilizando o algoritmo recursivo de classificação por intercalação.
- Como determinar a eficiência dos algoritmos de pesquisa e de classificação.

Sumário

16.1 Introdução

- **Pesquisar:**
  - **Envolve determinar se uma *chave de pesquisa* está presente nos dados.**
- **Classificar:**
  - **Posiciona os dados na ordem com base em uma ou mais *chaves de classificação*.**

Capítulo	Algoritmo	Localização
Algoritmos de pesquisa:		
16	Pesquisa linear	Seção 16.2.1
	Pesquisa binária	Seção 16.2.2
	Pesquisa linear recursiva	Exercício 16.8
	Pesquisa binária recursiva	Exercício 16.9
17	Pesquisa linear de uma `List`	Exercício 17.21
	Pesquisa em árvore binária	Exercício 17.23
19	Método `binarySearch` da classe `Collections`	Fig. 19.14
Algoritmos de classificação:		
16	Classificação por seleção	Seção 16.3.1
	Classificação por inserção	Seção 16.3.2
	Classificação por intercalação recursiva	Seção 16.3.3
	Classificação por borbulhamento (Bubble Sort)	Exercícios 16.3 e 16.4
	Bucket Sort	Exercícios 16.7
	Quicksort recursivo	Exercícios 16.10
17	Classificação em árvore binária	Seção 17.9
19	Metódo `sort` da classe `Collections`	Fig. 19.8–Fig. 19.11
	Coleção `SortedSet`	Fig. 19.19

Figura 16.1 | Algoritmos de pesquisa e classificação neste texto.

16.2 Algoritmos de pesquisa

- **Exemplos de pesquisa:**
  - **Pesquisando um número de telefone.**
  - **Acessando um site da Web.**
  - **Verificando uma palavra no dicionário.**

16.2.1 Pesquisa linear

- **Pesquisa linear:**
  - **Pesquisa cada elemento seqüencialmente.**
  - **Se a chave de pesquisa não estiver presente:**
    - **Testa cada elemento.**
    - **Quando o algoritmo alcança o final do array, ele informa ao usuário que a chave de pesquisa não está presente.**
  - **Se chave de pesquisa estiver presente:**
    - **Testa cada elemento até encontrar uma correspondência.**

Resumo

LinearArray.java

(1 de 2)

```
1  // Fig 16.2: LinearArray.java
2  // Classe que contém um array de inteiros aleatórios e um método
3  // que pesquisará esse array seqüencialmente
4  import java.util.Random;
5
6  public class LinearArray
7  {
8     private int[] data; // array de valores
9     private static Random generator = new Random();
10
11    // cria um array de um dado tamanho e o preenche com números aleatórios
12    public LinearArray( int size )
13    {
14       data = new int[ size ]; // cria espaço para o array
15
16       // preenche o array com ints aleatórios no intervalo de 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = 10 + generator.nextInt( 90 );
19    } // fim do construtor LinearArray
20
```

```
21    // realiza uma pesquisa linear nos dados
22    public int linearSearch( int searchKey )
23    {
24       // faz loop pelo array seqüencialmente
25       for ( int index = 0; index < data.length; index++ )
26          if ( data[ index ] == searchKey )
27             return index; // retorna o índice do intei
28
29       return -1; // inteiro não foi localizado
30    } // fim do método linearSearch
31
32    // método para gerar saída de valores no array
33    public String toString()
34    {
35       StringBuffer temporary = new StringBuffer();
36
37       // itera pelo array
38       for ( int element : data )
39          temporary.append( element + " " );
40
41       temporary.append( "\n" ); // adiciona um caractere de fim de linha
42       return temporary.toString();
43    } // fim do método toString
44 } // fim da classe LinearArray
```

Itera pelo array

Testa cada elemento seqüencialmente

Retorna o índice que contém a chave de pesquisa

Resumo

LinearSearchTest.java

(1 de 2)

```
1  // Fig 16.3: LinearSearchTest.java
2  // Pesquisa seqüencialmente um item em um array.
3  import java.util.Scanner;
4
5  public class LinearSearchTest
6  {
7     public static void main( String args[] )
8     {
9        // cria o objeto Scanner para inserir dados
10       Scanner input = new Scanner( System.in );
11
12       int searchInt; // chave de pesquisa
13       int position; // localização da chave de pesquisa no array
14
15       // cria o array e gera saída
16       LinearArray searchArray = new LinearArray( 10 );
17       System.out.println( searchArray ); // imprime array
18
19       // obtém entrada do usuário
20       System.out.print(
21          "Please enter an integer value (-1 to quit): " );
22       searchInt = input.nextInt(); // lê primeiro int do usuário
23
24       // insere repetidamente um inteiro; -1 termina o programa
25       while ( searchInt != -1 )
26       {
27          // realiza pesquisa linear
28          position = searchArray.linearSearch( searchInt );
29
```

Resumo

LinearSearchTest.java

(2 de 2)

```
30          if ( position == -1 ) // inteiro não foi localizado
31             System.out.println( "The integer " + searchInt +
32                " was not found.\n" );
33          else // integer foi localizado
34             System.out.println( "The integer " + searchInt +
35                " was found in position " + position + ".\n" );
36
37          // obtém entrada a partir do usuário
38          System.out.print(
39             "Please enter an integer value (-1 to quit): " );
40          searchInt = input.nextInt(); // lê o próximo int do usuário
41       } // fim do while
42    } // fim do main
43 } // fim da classe LinearSearchTest
```

```
16 35 68 10 48 36 81 60 84 21

Please enter an integer value (-1 to quit): 48
The integer 48 was found in position 4.

Please enter an integer value (-1 to quit): 60
The integer 60 was found in position 7.

Please enter an integer value (-1 to quit): 33
The integer 33 was not found.

Please enter an integer value (-1 to quit): -1
```

Eficiência da pesquisa linear

- **Notação big O:**
  - **Indica o pior cenário em tempo de execução para um algoritmo.**
  - **Em outras palavras, a dificuldade de um algoritmo em resolver um problema.**
  - **Tempo de execução constante:**
    - ***O*(1).**
    - **Não aumenta à medida que o tamanho do array aumenta.**
  - **Tempo de execução linear:**
    - ***O(n).***
    - **Aumenta proporcionalmente ao tamanho do array.**

Eficiência da pesquisa linear (*Cont.*)

- **Notação big O:**
  - **Tempo de execução constante:**
    - ***O*(1).**
    - **Não aumenta à medida que o tamanho do array aumenta.**
  - **Tempo de execução linear:**
    - ***O*(*n*).**
    - **Aumenta proporcionalmente ao tamanho do array.**
  - **Tempo de execução quadrático:**
    - ***O*(n^2).**
    - **Aumenta proporcionalmente ao quadrado do tamanho do array.**

Eficiência da pesquisa linear (*Cont.*)

- **Algoritmo de pesquisa linear:**
  - ***O(n)*.**
  - **Pior caso: O algoritmo verifica cada elemento antes de ser capaz de determinar se a chave de pesquisa não está presente.**
  - **Aumenta proporcionalmente ao tamanho do array.**

Dica de desempenho 16.1

Às vezes, os algoritmos mais simples demonstram um pobre desempenho. Sua virtude é que eles são fáceis de programar, testar e depurar. Às vezes, algoritmos mais complexos são necessários para conseguir desempenho máximo.

16.2.2 Pesquisa binária

- **Pesquisa binária:**
  - **Mais eficiente que a pesquisa linear.**
  - **Requer que os elementos sejam classificados.**
  - **Testa o elemento intermediário em um array.**
    - **Se for a chave de pesquisa, o algoritmo retorna.**
    - **Do contrário, se a chave de pesquisa for menor, elimina a maior metade do array.**
    - **Se a chave de pesquisa for maior, elimina a menor metade do array.**
  - **Cada iteração elimina metade dos elementos restantes.**

Resumo

BinaryArray.java

(1 de 3)

```
1  // Fig 16.4: BinaryArray.java
2  // Classe que contém um array de inteiros aleatórios e um método
3  // que utiliza a pesquisa binária para localizar um inteiro.
4  import java.util.Random;
5  import java.util.Arrays;
6
7  public class BinaryArray
8  {
9     private int[] data; // array de valores
10    private static Random generator = new Random();
11
12    // cria um array de um dado tamanho e o preenche com inteiros aleatórios
13    public BinaryArray( int size )
14    {
15       data = new int[ size ]; // cria espaço para o array
16
17       // preenche o array com ints aleatórios no intervalo 10-99
18       for ( int i = 0; i < size; i++ )
19          data[ i ] = 10 + generator.nextInt( 90 );
20
21       Arrays.sort( data );
22    } // fim do construtor BinaryArray
23
```

```
24    // realiza uma pesquisa binária nos dados
25    public int binarySearch( int searchElement )
26    {
27       int low = 0; // extremidade baixa da área de pes
28       int high = data.length - 1; // extremidade alta da área de pesquisa
29       int middle = ( low + high + 1 ) / 2; // elemento intermediário
30       int location = -1; // retorna valor; -1 se não l
31
32       do // faz loop jpara pesquisar elemento
33       {
34          // imprime os elementos remanescentes do array
35          System.out.print( remainingElements( low, high ) );
36
37          // gera espaços para alinhamento
38          for ( int i = 0; i < middle; i++ )
39             System.out.print( "   " );
40          System.out.println( " * " ); // indica o meio
41
42          // se o elemento for localizado no meio
43          if ( searchElement == data[ middle ] )
44             location = middle; // localização é o meio
45
46          // elemento do meio é muito alto
47          else if ( searchElement < data[ middle ] )
48             high = middle - 1; // elimina a metade sup
49          else  // elemento é muito baixo
50             low = middle + 1; // elimina a metade inferior
51
```

mo

ray.java

Armazena a parte superior, inferior e intermediária do array remanescente para pesquisar

Faz um loop até que a chave seja localizada ou não houver elementos a pesquisar

Se o elemento de pesquisa for o elemento intermediário

Retorna o elemento intermediário

Se elemento de pesquisa for menor que o elemento intermediário

Elimina a metade superior

Do contrário, elimina a metade inferior

```
52          middle = ( low + high + 1 ) / 2; // recalcula o meio
53       } while ( ( low <= high ) && ( location == -1 ) );
54
55       return location; // retorna localização da chave de pesquisa
56    } // fim do método binarySearch
57
58    // método para gerar saída de certos valores no array
59    public String remainingElements( int low, int high )
60    {
61       StringBuffer temporary = new StringBuffer();
62
63       // gera espaços para alinhamento
64       for ( int i = 0; i < low; i++ )
65          temporary.append( "   " );
66
67       // gera saída de elementos deixados no array
68       for ( int i = low; i <= high; i++ )
69          temporary.append( data[ i ] + " " );
70
71       temporary.append( "\n" );
72       return temporary.toString();
73    } // fim do método remainingElements
74
75    // método para gerar saída de valores no array
76    public String toString()
77    {
78       return remainingElements( 0, data.length - 1 );
79    } // fim do método toString
80 } // fim da classe BinaryArray
```

Atualiza o meio do array

Retorna a localização do elemento

Resumo

BinarySearchTest.java

(1 de 3)

```
1  // Fig 16.5: BinarySearchTest.java
2  // Utiliza a pesquisa binária para localizar um item em um array.
3  import java.util.Scanner;
4
5  public class BinarySearchTest
6  {
7     public static void main( String args[] )
8     {
9        // cria objeto Scanner para inserir dados
10       Scanner input = new Scanner( System.in );
11
12       int searchInt; // chava de pesquisa
13       int position; // localização de chave de pesquisa no array
14
15       // cria array e gera saída
16       BinaryArray searchArray = new BinaryArray( 15 );
17       System.out.println( searchArray );
18
```

```
19       // obtém dados do usuário
20       System.out.print(
21          "Please enter an integer value (-1 to quit): " );
22       searchInt = input.nextInt(); // lê um int do usuário
23       System.out.println();
24
25       // insere repetidamente um inteiro; -1 termina o programa
26       while ( searchInt != -1 )
27       {
28          // usa pesquisa binária para tentar localizar o inteiro
29          position = searchArray.binarySearch( searchInt );
30
31          // valor de retorno de -1 indica que o inteiro não foi localizado
32          if ( position == -1 )
33             System.out.println( "The integer " + searchInt +
34                " was not found.\n" );
35          else
36             System.out.println( "The integer " + searchInt +
37                " was found in position " + position + ".\n" );
38
39          // obtém a entrada do usuário
40          System.out.print(
41             "Please enter an integer value (-1 to quit): " );
42          searchInt = input.nextInt(); // lê um int do usuário
43          System.out.println();
44       } // fim do while
45    } // fim do main
46 } // fim da classe BinarySearchTest
```

Resumo

BinarySearchTest.java

(2 de 3)

```
 13 23 24 34 35 36 38 42 47 51 68 74 75 85 97

Please enter an integer value (-1 to quit): 23

13 23 24 34 35 36 38 42 47 51 68 74 75 85 97
                        *
13 23 24 34 35 36 38
          *
13 23 24
    *
The integer 23 was found in position 1.

Please enter an integer value (-1 to quit): 75

13 23 24 34 35 36 38 42 47 51 68 74 75 85 97
                        *
                          47 51 68 74 75 85 97
                                    *
                                      75 85 97
                                         *
                                      75
                                       *
The integer 75 was found in position 12.

Please enter an integer value (-1 to quit): 52

13 23 24 34 35 36 38 42 47 51 68 74 75 85 97
                        *
                          47 51 68 74 75 85 97
                                    *
                          47 51 68
                              *
                                 68
                                  *
The integer 52 was not found.

Please enter an integer value (-1 to quit): -1
```

Resumo

`BinarySearchTest.java`

(3 de 3)

Eficiência da pesquisa binária

- **Pesquisa binária:**
  - **Cada comparação divide o tamanho do array restante.**
  - **Resulta em $O(\log n)$.**
  - **Também conhecida como *tempo de execução logarítmico*.**

16.3 Algoritmos de classificação

- **Classificando dados:**
  - **Adicionando dados em alguma ordem particular:**
    - **Um banco classifica os cheques pelo número da conta.**
    - **Empresas de telefonia classificam as contas pelo nome.**
  - **O resultado final sempre é o mesmo — um array classificado.**
  - **A escolha do algoritmo afeta a maneira como você alcança o resultado e, acima de tudo, a rapidez com que você alcança o resultado.**

16.3.1 Classificação por seleção

- **Classificação por seleção:**
  - **Um algoritmo de classificação simples, porém ineficiente.**
  - **Primeira iteração seleciona o menor elemento no array e o troca pelo primeiro elemento.**
  - **Cada iteração seleciona o menor elemento remanescente não-classificado e o troca pelo próximo elemento na frente do array.**
  - **Após *i* iterações, os menores *i* elementos serão classificados nos primeiros *i* elementos do array.**

Resumo

SelectionSort.java

(1 de 3)

```
1  // Fig 16.6: SelectionSort.java
2  // Classe que cria um array preenchido com inteiros aleatórios.
3  // Fornece um método para classificar o array com a classificação por seleção.
4  import java.util.Random;
5
6  public class SelectionSort
7  {
8     private int[] data; // array de valores
9     private static Random generator = new Random();
10
11    // cria um array de um dado tamanho e o preenche com inteiros aleatórios
12    public SelectionSort( int size )
13    {
14       data = new int[ size ]; // cria espaço para array
15
16       // preenche o array com ints aleatórios no intervalo 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = 10 + generator.nextInt( 90 );
19    } // fim do construtor SelectionSort
20
21    // classifica array usando classificação por seleção
22    public void sort()
23    {
24       int smallest; // índice do menor elemento
25
26       // faz um loop sobre data.length - 1 elemento
27       for ( int i = 0; i < data.length - 1; i++ )
28       {
29          smallest = i; // primeiro índice do array remanescente
30
```

Variável para armazenar o índice do menor elemento

Loop itera `length` – 1 vezes

Resumo

```
31          // faz um loop para localizar o índice do menor elemento
32          for ( int index = i + 1; index < data.length; index++ )
33             if ( data[ index ] < data[ smallest ] )
34                smallest = index;
35
36          swap( i, smallest ); // permuta o menor eleme
37          printPass( i + 1, smallest ); // gera saída da passagem do algoritmo
38       } // fim do for externo
39    } // fim do método sort
40
41    // método auxiliar para permutar valores em dois el
42    public void swap( int first, int second )
43    {
44       int temporary = data[ first ]; // armazena o primeiro em temporário
45       data[ first ] = data[ second ]; // substitui o primeiro pelo segundo
46       data[ second ] = temporary; // coloca o temporário no segundo
47    } // fim do método swap
48
49    // imprime uma passagem do algoritmo
50    public void printPass( int pass, int index )
51    {
52       System.out.print( String.format( "after pass %2d: ", pass ) );
53
54       // gera saída de elementos até o item selecionado
55       for ( int i = 0; i < index; i++ )
56          System.out.print( data[ i ] + "  " );
57
58       System.out.print( data[ index ] + "* " ); // indica swap
59
```

.java

(2 de 3)

Itera sobre os elementos remanescentes

Localiza o menor elemento remanescente

Troca o menor elemento pelo primeiro elemento não-classificado

```
60       // termina de gerar a saída do array
61       for ( int i = index + 1; i < data.length; i++ )
62          System.out.print( data[ i ] + "  " );
63
64       System.out.print( "\n               " ); // para alinhamento
65
66       // indica quantidade do array que é classificado
67       for ( int j = 0; j < pass; j++ )
68          System.out.print( "--  " );
69       System.out.println( "\n" ); // adiciona fim de linha
70    } // fim do método indicateSelection
71
72    // método para gerar saída de valores no array
73    public String toString()
74    {
75       StringBuffer temporary = new StringBuffer();
76
77       // itera pelo array
78       for ( int element : data )
79          temporary.append( element + "  " );
80
81       temporary.append( "\n" ); // adiciona caractere de final de linha
82       return temporary.toString();
83    } // fim do método toString
84 } // fim da classe SelectionSort
```

Resumo

SelectionSort.java

(3 de 3)

Resumo

SelectionSortTest.java

(1 de 2)

```
1  // Fig 16.7: SelectionSortTest.java
2  // Testa a classe de classificação por seleção.
3
4  public class SelectionSortTest
5  {
6     public static void main( String[] args )
7     {
8        // cria objeto para realizar classificação por seleção
9        SelectionSort sortArray = new SelectionSort( 10 );
10
11       System.out.println( "Unsorted array:" );
12       System.out.println( sortArray ); // imprime array não classificado
13
14       sortArray.sort(); // classifica array
15
16       System.out.println( "Sorted array:" );
17       System.out.println( sortArray ); // imprime array classificado
18    } // fim de main
19 } // fim da classe SelectionSortTest
```

```
Unsorted array:
61  87  80  58  40  50  20  13  71  45

after pass  1: 13  87  80  58  40  50  20  61* 71  45
               --

after pass  2: 13  20  80  58  40  50  87* 61  71  45
               --  --

after pass  3: 13  20  40  58  80* 50  87  61  71  45
               --  --  --

after pass  4: 13  20  40  45  80  50  87  61  71  58*
               --  --  --  --

after pass  5: 13  20  40  45  50  80* 87  61  71  58
               --  --  --  --  --

after pass  6: 13  20  40  45  50  58  87  61  71  80*
               --  --  --  --  --  --

after pass  7: 13  20  40  45  50  58  61  87* 71  80
               --  --  --  --  --  --  --

after pass  8: 13  20  40  45  50  58  61  71  87* 80
               --  --  --  --  --  --  --  --

after pass  9: 13  20  40  45  50  58  61  71  80  87*
               --  --  --  --  --  --  --  --  --

Sorted array:
13  20  40  45  50  58  61  71  80  87
```

Resumo

`SelectionSortTest.java`

(2 de 2)

Eficiência da classificação por seleção

- **Classificação por seleção:**
  - **Loop `for` externo itera por $n - 1$ elementos.**
  - **Loop for interno itera pelos elementos remanescentes no array.**
  - **Resulta em $O(n^2)$.**

16.3.2 Classificação por inserção

- **Classificação por inserção:**
  - **Um outro algoritmo de classificação simples, mas ineficiente.**
  - **A primeira passagem recebe o segundo elemento e o insere na ordem correta em relação ao primeiro elemento.**
  - **Cada iteração recebe o próximo elemento no array e o insere nos elementos classificados no começo do array.**
  - **Após *i* iterações, os primeiros *i* elementos do array estarão na ordem classificada.**

Resumo

InsertionSort.java

(1 de 4)

```
1  // Fig 16.8: InsertionSort.java
2  // Classe que cria um array preenchido com inteiros aleatórios.
3  // fornece um método para classificar o array com a classificação por inserção.
4  import java.util.Random;
5
6  public class InsertionSort
7  {
8     private int[] data; // array de valores
9     private static Random generator = new Random();
10
11    // cria um array de um dado tamanho e o preenche com inteiros aleatórios
12    public InsertionSort( int size )
13    {
14       data = new int[ size ]; // cria espaço para o array
15
16       // preenche o array com ints aleatórios no intervalo de 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = 10 + generator.nextInt( 90 );
19    } // fim do construtor InsertionSort
20
```

```
21    // classifica o array utilizando a classificação po
22    public void sort()
23    {
24       int insert; // variável temporária para armazenar o elemento a inserir
25
26       // faz um loop sobre data.length - 1 elementos
27       for ( int next = 1; next < data.length; next++ )
28       {
29          // armazena o valor no elemento atual
30          insert = data[ next ];
31
32          // inicializa a localização para colocar elem
33          int moveItem = next;
34
35          // procura o local para colocar o elemento atual
36          while ( moveItem > 0 && data[ moveItem - 1 ] >
37          {
38             // desloca o elemento direito um slot
39             data[ moveItem ] = data[ moveItem - 1 ];
40             moveItem--;
41          } // end while
42
43          data[ moveItem ] = insert; // coloca o elemento inserido
44          printPass( next, moveItem ); // passagem de saída do algoritmo
45       } // fim de for
46    } // fim do método sort
47
```

Declara a variável para armazenar o elemento a ser inserido

Itera por `length` – 1 elementos

Armazena o valor a inserir

Procura o local em que posicionar o elemento inserido

Move um dos elementos para a direita

Decrementa o local a inserir o elemento

Insere o elemento no local classificado

Resumo

InsertionSort.java

(3 de 4)

```
48    // imprime uma passagem do algoritmo
49    public void printPass( int pass, int index )
50    {
51       System.out.print( String.format( "after pass %2d: ", pass ) );
52
53       // gera saída dos elementos até o item permutado
54       for ( int i = 0; i < index; i++ )
55          System.out.print( data[ i ] + "  " );
56
57       System.out.print( data[ index ] + "* " ); // indica permuta
58
59       // termina de gerar a saída do array
60       for ( int i = index + 1; i < data.length; i++ )
61          System.out.print( data[ i ] + "  " );
62
63       System.out.print( "\n               " ); // para alinhamento
64
65       // indica quantidade do array que é classificado
66       for( int i = 0; i <= pass; i++ )
67          System.out.print( "--  " );
68       System.out.println( "\n" ); // adiciona fim de linha
69    } // fim do método printPass
70
```

```
71    // método de gerar valores no array
72    public String toString()
73    {
74       StringBuffer temporary = new StringBuffer();
75
76       // itera pelo array
77       for ( int element : data )
78          temporary.append( element + "   " );
79
80       temporary.append( "\n" ); // adiciona caractere de final de linha
81       return temporary.toString();
82    } // fim do método toString
83 } // fim da classe InsertionSort
```

Resumo

InsertionSort.java

(4 de 4)

Resumo

InsertionSortTest.java

(1 de 2)

```
1  // Fig 16.9: InsertionSortTest.java
2  // Testa a classe de classificação por inserção.
3
4  public class InsertionSortTest
5  {
6     public static void main( String[] args )
7     {
8        // cria um objeto para realizar a classificação por seleção
9        InsertionSort sortArray = new InsertionSort( 10 );
10
11       System.out.println( "Unsorted array:" );
12       System.out.println( sortArray ); // imprime um array não-classificado
13
14       sortArray.sort(); // classifica o array
15
16       System.out.println( "Sorted array:" );
17       System.out.println( sortArray ); // imprime o array classificado
18    } // fim de main
19 } // fim da classe InsertionSortTest
```

```
Unsorted array:
40  17  45  82  62  32  30  44  93  10

after pass  1: 17* 40  45  82  62  32  30  44  93  10
               --  --

after pass  2: 17  40  45* 82  62  32  30  44  93  10
               --  --  --

after pass  3: 17  40  45  82* 62  32  30  44  93  10
               --  --  --  --

after pass  4: 17  40  45  62* 82  32  30  44  93  10
               --  --  --  --  --

after pass  5: 17  32* 40  45  62  82  30  44  93  10
               --  --  --  --  --  --

after pass  6: 17  30* 32  40  45  62  82  44  93  10
               --  --  --  --  --  --  --

after pass  7: 17  30  32  40  44* 45  62  82  93  10
               --  --  --  --  --  --  --  --

after pass  8: 17  30  32  40  44  45  62  82  93* 10
               --  --  --  --  --  --  --  --  --

after pass  9: 10* 17  30  32  40  44  45  62  82  93
               --  --  --  --  --  --  --  --  --  --

Sorted array:
10  17  30  32  40  44  45  62  82  93
```

Resumo

`InsertionSortTest.java`

(2 de 2)

Eficiência da classificação por inserção

- **Classificação por inserção:**
  - **Loop `for` externo itera por $n - 1$ elementos.**
  - **Loop `while` interno itera sobre os elementos precedentes do array.**
  - **Resulta em $O(n^2)$.**

16.3.3 Classificação por intercalação

- **Classificação por intercalação:**
  - **Um *algoritmo de classificação* mais eficiente, porém também mais complexo.**
  - **Divide o array em dois subarrays com aproximadamente o mesmo tamanho; classifica cada subarray e então intercala os subarrays.**
  - **A implementação a seguir é recursiva:**
    - **O caso básico é um array de um elemento que não pode ser não classificado.**
    - **O passo de recursão divide um array em duas partes, classifica cada parte e então intercala as partes classificadas.**

Resumo

MergeSort.java

(1 de 5)

```
1  // Figure 16.10: MergeSort.java
2  // Classe que cria um array preenchido com inteiros aleatórios.
3  // Fornece um método p/ classificar o array c/ a classificação por intercalação.
4  import java.util.Random;
5
6  public class MergeSort
7  {
8     private int[] data; // array de valores
9     private static Random generator = new Random();
10
11    // cria um array de um dado tamanho e o preenche com inteiros aleatórios
12    public MergeSort( int size )
13    {
14       data = new int[ size ]; // cria espaço para o array
15
16       // preenche o array com ints aleatórios no intervalo de 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = 10 + generator.nextInt( 90 );
19    } // fim do construtor MergeSort
20
21    // chama o método de divisão recursiva p/ iniciar a cla
22    public void sort()
23    {
24       sortArray( 0, data.length - 1 ); // divide o array inteiro
25    } // fim do método sort
26
```

Chama o método auxiliar recursivo

Resumo

```
27    // divide o array, classifica subarrays e intercala subarrays no array classificado
28    private void sortArray( int low, int high )
29    {
30       // testa caso básico; tamanho do array é igual a 1
31       if ( ( high - low ) >= 1 ) // se não for o caso b
32       {
33          int middle1 = ( low + high ) / 2; // calcula meio do array
34          int middle2 = middle1 + 1; // calcula o próximo elemento over
35
36          // gera uma saída do passo de divisão
37          System.out.println( "split:    " + subarray( l
38          System.out.println( "          " + subarray( low, middle1 );
39          System.out.println( "          " + subarray( middle2, high ) );
40          System.out.println();
41
42          // divide o array pela metade; classifica cada metade (chamadas recursivas)
43          sortArray( low, middle1 ); // primeira metade
44          sortArray( middle2, high ); // segunda metade
45
46          // intercala dois arrays classificados depois que as chamadas de divisão retornam
47          merge ( low, middle1, middle2, high );
48       } // fim do if
49    } // fim do método split
50
```

Testa para o caso básico

Calcula o meio do array

Calcula o elemento uma casa à direita do meio

Classifica recursivamente a primeira metade do array

Classifica recursivamente a segunda metade do array

Intercala os dois subarrays classificados

MergeSort.java

(2 de 5)

```
51    // intercala dois subarrays classificados em um subarray classificado
52    private void merge( int left, int middle1, int midd
53    {
54       int leftIndex = left; // índice no subarray esquerdo
55       int rightIndex = middle2; // índice no subarray
56       int combinedIndex = left; // índice no array tem
57       int[] combined = new int[ data.length ]; // arra
58
59       // gera saída de dois subarrays antes de interca
60       System.out.println( "merge:    " + subarray( left
61       System.out.println( "          " + subarray( midd
62
63       // intercala arrays até alcançar o final de um d
64       while ( leftIndex <= middle1 && rightIndex <= ri
65       {
66          // coloca o menor dos dois elementos atuais no resultado
67          // e o move para o próximo espaço nos arrays
68          if ( data[ leftIndex ] <= data[ rightIndex ]
69             combined[ combinedIndex++ ] = data[ leftIn
70          else
71             combined[ combinedIndex++ ] = data[ rightIndex++ ];
72       } // fim de while
73
```

Índice do elemento no array esquerdo

Índice do elemento no array direito

Índice par colocar o elemento no array combinado

Array para conter os elementos classificados

Faz um loop até alcançar o final de um dos arrays

Determina o menor dos dois elementos

Posiciona o menor elemento no array combinado

```
74       // se o array esquerdo estiver vazio
75       if ( leftIndex == middle2 )
76          // copia o restante do array direito
77          while ( rightIndex <= right )
78             combined[ combinedIndex++ ] = data[ rightIndex++ ];
79       else // array direito está vazio
80          // copia o restante do array direito
81          while ( leftIndex <= middle1 )
82             combined[ combinedIndex++ ] = data[ leftIndex++ ];
83
84       // copia os valores de volta ao array original
85       for ( int i = left; i <= right; i++ )
86          data[ i ] = combined[ i ];
87
88       // gera saída do array intercalado
89       System.out.println( "          " + subarray( left, right ) );
90       System.out.println();
91    } // fim do método merge
92
```

Se o array esquerdo estiver vazio

Preenche com elementos do array direito

Se o array direito estiver vazio

Preenche com elementos do array esquerdo

Copia os valores de volta ao array original

(4 de 5)

Resumo

MergeSort.java

(5 de 5)

```
93    // método para gerar saída de certos valores no array
94    public String subarray( int low, int high )
95    {
96       StringBuffer temporary = new StringBuffer();
97
98       // gera saída de espaços para alinhamento
99       for ( int i = 0; i < low; i++ )
100         temporary.append( "   " );
101
102      // gera saída de elementos deixados no array
103      for ( int i = low; i <= high; i++ )
104         temporary.append( " " + data[ i ] );
105
106      return temporary.toString();
107   } // fim do método subarray
108
109   // método para gerar saída de valores no array
110   public String toString()
111   {
112      return subarray( 0, data.length - 1 );
113   } // fim do método toString
114} // fim da classe MergeSort
```

Resumo

MergeSortTest.java

(1 de 4)

```
1  // Figure 16.11: MergeSortTest.java
2  // Testa a classe de classificação por intercalação.
3
4  public class MergeSortTest
5  {
6     public static void main( String[] args )
7     {
8        // cria objeto para realizar classificação por intercalação
9        MergeSort sortArray = new MergeSort( 10 );
10
11       // imprime array não classificado
12       System.out.println( "Unsorted:" + sortArray + "\n" );
13
14       sortArray.sort(); // classifica array
15
16       // imprime array classificado
17       System.out.println( "Sorted:   " + sortArray );
18    } // fim do main
19 } // fim da classe MergeSortTest
```

```
Unsorted: 75 56 85 90 49 26 12 48 40 47

split:    75 56 85 90 49 26 12 48 40 47
          75 56 85 90 49
                         26 12 48 40 47

split:    75 56 85 90 49
          75 56 85
                   90 49

split:    75 56 85
          75 56
                85

split:    75 56
          75
             56
```

Resumo

MergeSortTest.java

(2 de 4)

```
merge:    75
             56
          56 75

merge:    56 75
                85
          56 75 85

split:               90 49
                     90
                        49

merge:               90
                        49
                     49 90

merge:    56 75 85
                     49 90
          49 56 75 85 90

split:                        26 12 48 40 47
                              26 12 48
                                       40 47

split:                        26 12 48
                              26 12
                                    48

split:                        26 12
                              26
                                 12
```

Resumo

MergeSortTest.java

(3 de 4)

```
merge:                    26
                             12
                          12 26

merge:                    12 26
                                48
                          12 26 48

split:                              40 47
                                    40
                                       47

merge:                              40
                                       47
                                    40 47

merge:                    12 26 48
                                    40 47
                          12 26 40 47 48

merge:    49 56 75 85 90

          12 26 40 47 48 49 56 75 85 90

Sorted:   12 26 40 47 48 49 56 75 85 90
```

Resumo

`MergeSortTest.java`

(4 de 4)

Eficiência da classificação por intercalação

- **Classificação por intercalação:**
  - **Muito mais eficiente que a classificação por seleção ou classificação por inserção.**
  - **A última intercalação requer $n - 1$ comparações para intercalar todo o array.**
  - **Cada nível mais baixo tem o dobro de chamadas ao método merge, com cada chamada operando em um array com metade do tamanho, o que resulta em $O(n)$ comparações totais.**
  - **Deverá haver $O(\log n)$ níveis.**
  - **Resulta em $O(n \log n)$.**

Algoritmo	Localização	Big O
Algoritmos de pesquisa:		
Pesquisa linear	**Seção 16.2.1**	***O(n)***
Pesquisa binária	**Seção 16.2.2**	***O(log n)***
Pesquisa linear recursiva	**Exercício 16.8**	***O(n)***
Pesquisa binária recursiva	**Exercício 16.9**	***O(log n)***
Algoritmos de classificação:		
Classificação por seleção	**Seção 16.3.1**	$O(n^2)$
Classificação por inserção	**Seção 16.3.2**	$O(n^2)$
Classificação por intercalação	**Seção 16.3.3**	$O(n \log n)$
Classificação por borbulhamento (Bubble Sort)	**Exercício 16.3 e 16.4**	$O(n^2)$

Figura 16.12 | Algoritmos de pesquisa e classificação com valores na notação Big O.

n =	*O*(log *n*)	*O*(*n*)	*O*(*n* log *n*)	*O*(n^2)
1	0	1	0	1
2	1	2	2	4
3	1	3	3	9
4	1	4	4	16
5	1	5	5	25
10	1	10	10	100
100	2	100	200	10000
1,000	3	1000	3000	10^6
1,000,000	6	1000000	6000000	10^{12}
1,000,000,000	9	1000000000	9000000000	10^{18}

Figura 16.13 | Número de comparações para notações Big O comuns.

16.4 Invariantes

- **Invariante:**
  - **É uma assertiva que é verdadeira antes e depois de uma parte do seu código ser executada.**
  - **O tipo mais comum é uma invariante de loop.**

16.4 Invariantes

- **A invariante do loop permanece verdadeira:**
  - **antes da execução do loop;**
  - **após cada iteração do corpo do loop; e**
  - **quando o loop termina.**

16.4 Invariantes

- **Quatro passos para desenvolver um loop a partir de uma invariante de loop.**
  - **Configure os valores iniciais para quaisquer variáveis de controle do loop.**
  - **Determine a condição que faz com que o loop termine.**
  - **Modifique a(s) variável(is) de controle de modo que o loop avance em direção ao término.**
  - **Verifique se a invariante permanece verdadeira no final de cada iteração.**