24

Searching and Sorting

© 2006 Pearson Education, Inc. All rights reserved.

With sobs and tears
he sorted out
Those of the largest size …

— Lewis Carroll

Attempt the end, and never stand to doubt;
Nothing's so hard, but search will find it out.

— Robert Herrick

It is an immutable law in business that words are words, explanations are explanations, promises are promises — but only performance is reality.

— Harold S. Green

© 2006 Pearson Education, Inc. All rights reserved.

OBJECTIVES

In this chapter you will learn:

- To search for a given value in an array using the linear search and binary search algorithm.
- To sort arrays using the iterative selection and insertion sort algorithms.
- To sort arrays using the recursive merge sort algorithm.
- To determine the efficiency of searching and sorting algorithms.

© 2006 Pearson Education, Inc. All rights reserved.

Outline

24.1 Introduction

24.2 Searching Algorithms

- 24.2.1 Linear Search
- 24.2.2 Binary Search

24.3 Sorting Algorithms

- 24.3.1 Selection Sort
- 24.3.2 Insertion Sort
- 24.3.3 Merge Sort

24.4 Wrap-Up

© 2006 Pearson Education, Inc. All rights reserved.

24.1 Introduction

- **Searching**
  - **Determining whether a search key is present in data**
- **Sorting**
  - **Places data in order based on one or more sort keys**

© 2006 Pearson Education, Inc. All rights reserved.

Chapter	Algorithm	Location
Searching Algorithms:		
24	Linear Search	Section 24.2.1
	Binary Search	Section 24.2.2
	Recursive Linear Search	Exercise 24.8
	Recursive Binary Search	Exercise 24.9
27	`BinarySearch` method of class `Array`	Fig. 27.3
	`Contains` method of classes `ArrayList` and `Stack`	Fig. 27.4
	`ContainsKey` method of class `HashTable`	Fig. 27.7
Sorting Algorithms:		
24	Selection Sort	Section 24.3.1
	Insertion Sort	Section 24.3.2
	Recursive Merge Sort	Section 24.3.3
	Bubble Sort	Exercises 24.5–24.6
	Bucket Sort	Exercise 24.7
	Recursive Quicksort	Exercise 24.10
24, 27	`Sort` method of classes `Array` and `ArrayList`	Fig. 24.4, Figs. 27.3–27.4

Fig. 24.1 | Searching and sorting capabilities in this text.

© 2006 Pearson Education, Inc. All rights reserved.

24.2 Searching Algorithms

- **Linear Search**
  - Searches each element in an array sequentially
  - Has $O(n)$ time
    - The worst case is that every element must be checked to determine whether the search item exists in the array

- **Big O notation**
  - One way to describe the efficiency of a search
    - Measures the worst-case run time for an algorithm
    - $O(1)$ is said to have a constant run time
    - $O(n)$ is said to have a linear run time
    - $O(n^2)$ is said to have a quadratic run time

© 2006 Pearson Education, Inc. All rights reserved.

Outline

LinearArray.cs

(1 of 2)

```
1  // Fig 24.2: LinearArray.cs
2  // Class that contains an array of random integers and a method
3  // that will search that array sequentially.
4  using System;
5
6  public class LinearArray
7  {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11    // create array of given size and fill with random integers
12    public LinearArray( int size )
13    {
14       data = new int[ size ]; // create space for array
15
16       // fill array with random ints in range 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = generator.Next( 10, 100 );
19    } // end LinearArray constructor
20
21    // perform a linear search on the data
22    public int LinearSearch( int searchKey )
23    {
24       // loop through array sequentially
25       for ( int index = 0; index < data.Length; index++ )
26          if ( data[ index ] == searchKey )
27             return index; // return index of integer
28
29       return -1; // integer was not found
30    } // end method LinearSearch
```

Fill `int` array with random numbers

Iterate through array sequentially

Compare each array element with search key

Return index if search key is found

Return -1 if search key is not found

© 2006 Pearson Education, Inc. All rights reserved.

```
31
32    // method to output values in array
33    public override string ToString()
34    {
35       string temporary = "";
36
37       // iterate through array
38       foreach ( int element in data )
39          temporary += element + " ";
40
41       temporary += "\n"; // add newline character
42       return temporary;
43    } // end method ToString
44 } // end class LinearArray
```

Outline

`LinearArray.cs`

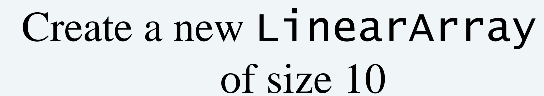
(2 of 2)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

LinearSearch Test.cs

```
1  // Fig 24.3: LinearSearchTest.cs
2  // Sequentially search an array for an item.
3  using System;
4
5  public class LinearSearchTest
6  {
7     public static void Main( string[] args )
8     {
9        int searchInt; // search key
10       int position; // location of search key in array
11
12       // create array and output it
13       LinearArray searchArray = new LinearArray( 10 );
14       Console.WriteLine( searchArray ); // print array
15
16       // input first int from user
17       Console.Write( "Please enter an integer value (-1 to quit): " );
18       searchInt = Convert.ToInt32( Console.ReadLine() );
```



© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
19
20        // repeatedly input an integer; -1 terminates the application
21        while ( searchInt != -1 )
22        {
23           // perform linear search
24           position = searchArray.LinearSearch( searchInt );
25
26           if ( position != -1 ) // integer was not found
27              Console.WriteLine(
28                 "The integer {0} was found in position {1}.\n",
29                 searchInt, position );
30           else // integer was found
31              Console.WriteLine( "The integer {0} was not found.\n",
32                 searchInt );
33
34           // input next int from user
35           Console.Write( "Please enter an integer value (-1 to quit): " );
36           searchInt = Convert.ToInt32( Console.ReadLine() );
37        } // end while
38     } // end Main
39  } // end class LinearSearchTest
```

Execute linear search on the array for the search key

LinearSearch
Test.cs

Output the appropriate message based on the linear search's result

Continue prompting user for another search key

© 2006 Pearson Education, Inc. All rights reserved.

```
64 90 84 62 28 68 55 27 78 73

Please enter an integer value (-1 to quit): 78
The integer 78 was found in position 8.

Please enter an integer value (-1 to quit): 64
The integer 64 was found in position 0.

Please enter an integer value (-1 to quit): 65
The integer 65 was not found.

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

© 2006 Pearson Education, Inc. All rights reserved.

Performance Tip 24.1

Sometimes the simplest algorithms perform poorly. Their virtue is that they are easy to program, test and debug. Sometimes more complex algorithms are required to realize maximum performance.

© 2006 Pearson Education, Inc. All rights reserved.

24.2 Searching Algorithms (Cont.)

- **Binary Search**
  - **Requires that the array be sorted**
    - **For this example, assume the array is sorted in ascending order**
  - **The first iteration of this algorithm tests the middle element**
    - **If this matches the search key, the algorithm ends**
    - **If the search key is less than the middle element, the algorithm continues with only the first half of the array**
      - **The search key cannot match any element in the second half of the array**
    - **If the search key is greater than the middle element, the algorithm continues with only the second half of the array**
      - **The search key cannot match any element in the first half of the array**
  - **Each iteration tests the middle value of the remaining portion of the array**
    - **Called a subarray**
  - **If the search key does not match the element, the algorithm eliminates half of the remaining elements**
  - **The algorithm ends either by finding an element that matches the search key or reducing the subarray to zero size**
  - **Is more efficient than the linear search algorithm, *O(log n)***
    - **Known as logarithmic run time**

© 2006 Pearson Education, Inc. All rights reserved.

```
1  // Fig 24.4: BinaryArray.cs
2  // Class that contains an array of random integers and a method
3  // that uses binary search to find an integer.
4  using System;
5
6  public class BinaryArray
7  {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11    // create array of given size and fill with random integers
12    public BinaryArray( int size )
13    {
14       data = new int[ size ]; // create space for array
15
16       // fill array with random ints in range 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = generator.Next( 10, 100 );
19
20       Array.Sort( data );
21    } // end BinaryArray constructor
22
23    // perform a binary search on the data
24    public int BinarySearch( int searchElement )
25    {
26       int low = 0; // low end of the search area
27       int high = data.Length - 1; // high end of the search area
28       int middle = ( low + high + 1 ) / 2; // middle element
29       int location = -1; // return value; -1 if not found
```

BinaryArray.cs

Fill `int` array with random integers

Sort the array of random numbers in ascending order (requirement of the binary search)

Declare local variables to store the indexes of the array for the binary search

Calculate the middle, low, and high end index of the array

Initializes `location` to -1 which is the value returned if element is not found

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
30
31        do // loop to search for element
32        {
33           // print remaining elements of array
34           Console.Write( RemainingElements( low, high ) );
35
36           // output spaces for alignment
37           for ( int i = 0; i < middle; i++ )
38              Console.Write( "   " );
39
40           Console.WriteLine( " * " ); // indicate current middle
41
42           // if the element is found at the middle
43           if ( searchElement == data[ middle ] )
44              location = middle; // location is the current middle
45
46           // middle element is too high
47           else if ( searchElement < data[ middle ] )
48              high = middle - 1; // eliminate the higher half
49           else // middle element is too low
50              low = middle + 1; // eliminate the lower half
51
52           middle = ( low + high + 1 ) / 2; // recalculate the middle
53        } while ( ( low <= high ) && ( location == -1 ) );
54
55        return location; // return location of search key
56     } // end method BinarySearch
```

BinaryArray.cs

Tests whether the value in the middle element is equal to searchElement

If found, assigns middle to location

Eliminate the appropriate half of the remaining values in the array

Determine the new middle position of the remaining array

Loop until low is greater than high or searchElement is found

Return result

© 2003 Pearson Education, Inc. All rights reserved.

Outline

BinaryArray.cs

(3 of 3)

```
57
58    // method to output certain values in array
59    public string RemainingElements( int low, int high )
60    {
61       string temporary = "";
62
63       // output spaces for alignment
64       for ( int i = 0; i < low; i++ )
65          temporary += "   ";
66
67       // output elements left in array
68       for ( int i = low; i <= high; i++ )
69          temporary += data[ i ] + " ";
70
71       temporary += "\n";
72       return temporary;
73    } // end method RemainingElements
74
75    // method to output values in array
76    public override string ToString()
77    {
78       return RemainingElements( 0, data.Length - 1 );
79    } // end method ToString
80 } // end class BinaryArray
```

Return a range of numbers in the array

© 2006 Pearson Education, Inc. All rights reserved.

Outline

BinarySearchTest.cs

(1 of 3)

```
1  // Fig 24.5: BinarySearchTest.cs
2  // Use binary search to locate an item in an array.
3  using System;
4
5  public class BinarySearchTest
6  {
7     public static void Main( string[] args )
8     {
9        int searchInt; // search key
10       int position; // location of search key in array
11
12       // create array and output it
13       BinaryArray searchArray = new BinaryArray( 15 );
14       Console.WriteLine( searchArray );
15
16       // prompt and input first int from user
17       Console.Write( "Please enter an integer value (-1 to quit): " );
18       searchInt = Convert.ToInt32( Console.ReadLine() );
19       Console.WriteLine();
```

Create an ordered array of random numbers

Prompt user for a search key

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
20
21       // repeatedly input an integer; -1 terminates the application
22       while ( searchInt != -1 )
23       {
24          // use binary search to try to find integer
25          position = searchArray.BinarySearch( searchInt );
26
27          // return value of -1 indicates integer was not found
28          if ( position == -1 )
29             Console.WriteLine( "The integer {0} was not found.\n",
30                searchInt );
31          else
32             Console.WriteLine(
33                "The integer {0} was found in position {1}.\n",
34                searchInt, position);
35
36          // prompt and input next int from user
37          Console.Write( "Please enter an integer value (-1 to quit): " );
38          searchInt = Convert.ToInt32( Console.ReadLine() );
39          Console.WriteLine();
40       } // end while
41    } // end Main
42 } // end class BinarySearchTest
```

Perform binary search on the search key

Output the appropriate result

Prompt user for another search key

BinarySearchTest.cs

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
12 17 22 25 30 39 40 52 56 72 76 82 84 91 93
Please enter an integer value (-1 to quit): 72
12 17 22 25 30 39 40 52 56 72 76 82 84 91 93
                      *
                        56 72 76 82 84 91 93
                                  *
                        56 72 76
                             *
The integer 72 was found in position 9.
Please enter an integer value (-1 to quit): 13
12 17 22 25 30 39 40 52 56 72 76 82 84 91 93
                      *
12 17 22 25 30 39 40
            *
12 17 22
    *
12
 *
The integer 13 was not found.
Please enter an integer value (-1 to quit): -1
```

BinarySearchTest.cs

© 2006 Pearson Education, Inc. All rights reserved.

24.3 Sorting Algorithms

- **Selection Sort**
  - **The first iteration selects the smallest element in the array and swaps it with the first element**
  - **The second iteration selects the second-smallest item and swaps it with the second element**
  - **Continues until the last iteration selects the second-largest element and swaps it with the second-to-last index**
    - **Leaves the largest element in the last index**
  - **After the *i*th iteration, the smallest *i* items of the array will be sorted in increasing order in the first *i* elements of the array**
  - **Is a simple, but inefficient, sorting algorithm; $O(n^2)$**

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Fig 24.6: SelectionSort.cs
2  // Class that creates an array filled with random integers.
3  // Provides a method to sort the array with selection sort.
4  using System;
5
6  public class SelectionSort
7  {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11    // create array of given size and fill with random integers
12    public SelectionSort( int size )
13    {
14       data = new int[ size ]; // create space for array
15
16       // fill array with random ints in range 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = generator.Next( 10, 100 );
19    } // end SelectionSort constructor
20
21    // sort array using selection sort
22    public void Sort()
23    {
24       int smallest; // index of smallest element
25
26       // loop over data.Length - 1 elements
27       for ( int i = 0; i < data.Length - 1; i++ )
28       {
29          smallest = i; // first index of remaining array
```

Store the index of the smallest element in the remaining array

Iterate through the whole array `data.Length – 1` times

Initializes the index of the smallest element to the current item

SelectionSort.cs

(1 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```
30
31          // loop to find index of smallest element
32          for ( int index = i + 1; index < data.Length; index++ )
33             if ( data[ index ] < data[ smallest ] )
34                smallest = index;
35
36          Swap( i, smallest ); // swap smallest element into position
37          PrintPass( i + 1, smallest ); // output pass of algorithm
38       } // end outer for
39    } // end method Sort
40
41    // helper method to swap values in two elements
42    public void Swap( int first, int second )
43    {
44       int temporary = data[ first ]; // store first in temporary
45       data[ first ] = data[ second ]; // replace first with second
46       data[ second ] = temporary; // put temporary in second
47    } // end method Swap
48
49    // print a pass of the algorithm
50    public void PrintPass( int pass, int index )
51    {
52       Console.Write( "after pass {0}: ", pass );
53
54       // output elements through the selected item
55       for ( int i = 0; i < index; i++ )
56          Console.Write( data[ i ] + "  " );
57
58       Console.Write( data[ index ] + "* " ); // indicate swap
```

Determine the index of the remaining smallest element

Place the smallest remaining element in the next spot

Swap two elements

© 2006 Pearson Education, Inc. All rights reserved.

Outline

SelectionSort.cs

(3 of 3)

```
59
60        // finish outputting array
61        for ( int i = index + 1; i < data.Length; i++ )
62           Console.Write( data[ i ] + "  " );
63
64        Console.Write( "\n              " ); // for alignment
65
66        // indicate amount of array that is sorted
67        for( int j = 0; j < pass; j++ )
68           Console.Write( "--  " );
69        Console.WriteLine( "\n" ); // skip a line in output
70     } // end method PrintPass
71
72     // method to output values in array
73     public override string ToString()
74     {
75        string temporary = "";
76
77        // iterate through array
78        foreach ( int element in data )
79           temporary += element + "  ";
80
81        temporary += "\n"; // add newline character
82        return temporary;
83     } // end method ToString
84  } // end class SelectionSort
```

© 2006 Pearson Education, Inc. All rights reserved.

Outline

SelectionSortTest.cs

(1 of 2)

```
1  // Fig 24.7: SelectionSortTest.cs
2  // Test the selection sort class.
3  using System;
4
5  public class SelectionSortTest
6  {
7     public static void Main( string[] args )
8     {
9        // create object to perform selection sort
10       SelectionSort sortArray = new SelectionSort( 10 );
11
12       Console.WriteLine( "Unsorted array:" );
13       Console.WriteLine( sortArray ); // print unsorted array
14
15       sortArray.Sort(); // sort array
16
17       Console.WriteLine( "Sorted array:" );
18       Console.WriteLine( sortArray ); // print sorted array
19    } // end Main
20 } // end class SelectionSortTest
```

Create an array of random integers

Perform selection sort on the array

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
Unsorted array:
86   97   83   45   19   31   86   13   57   61

after pass 1: 13   97   83   45   19   31   86   86*  57   61
              --

after pass 2: 13   19   83   45   97*  31   86   86   57   61
              --   --

after pass 3: 13   19   31   45   97   83*  86   86   57   61
              --   --   --

after pass 4: 13   19   31   45*  97   83   86   86   57   61
              --   --   --   --

after pass 5: 13   19   31   45   57   83   86   86   97*  61
              --   --   --   --   --

after pass 6: 13   19   31   45   57   61   86   86   97   83*
              --   --   --   --   --   --

after pass 7: 13   19   31   45   57   61   83   86   97   86*
              --   --   --   --   --   --   --

after pass 8: 13   19   31   45   57   61   83   86*  97   86
              --   --   --   --   --   --   --   --

after pass 9: 13   19   31   45   57   61   83   86   86   97*
              --   --   --   --   --   --   --   --   --

Sorted array:
13   19   31   45   57   61   83   86   86   97
```

`SelectionSort Test.cs`

(2 of 2)

© 2006 Pearson Education, Inc. All rights reserved.

24.3 Sorting Algorithms (Cont.)

- **Insertion Sort**
  - **The first iteration takes the second element in the array and swaps it with the first element if it is less than the first element**
  - **The second iteration looks at the third element and inserts it in the correct position with respect to the first two elements**
    - **All three elements will be in order**
  - **At the *i*th iteration of this algorithm, the first *i* elements in the original array will be sorted**
  - **Another simple, but inefficient, sorting algorithm; $O(n^2)$**

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Fig 24.8: InsertionSort.cs
2  // Class that creates an array filled with random integers.
3  // Provides a method to sort the array with insertion sort.
4  using System;
5
6  public class InsertionSort
7  {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11    // create array of given size and fill with random integers
12    public InsertionSort( int size )
13    {
14       data = new int[ size ]; // create space for array
15
16       // fill array with random ints in range 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = generator.Next( 10, 100 );
19    } // end InsertionSort constructor
20
21    // sort array using insertion sort
22    public void Sort()
23    {
24       int insert; // temporary variable to hold element to insert
25
26       // loop over data.Length - 1 elements
27       for ( int next = 1; next < data.Length; next++ )
28       {
29          // store value in current element
30          insert = data[ next ];
```

Holds the element to be inserted while the order elements are moved

Iterate through `data.Length - 1` items in the array

Stores the value of the element that will be inserted in the sorted portion of the array

`InsertionSort.cs`

(1 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```
31
32          // initialize location to place element
33          int moveItem = next;
34
35          // search for place to put current element
36          while ( moveItem > 0 && data[ moveItem - 1 ] > insert )
37          {
38             // shift element right one slot
39             data[ moveItem ] = data[ moveItem - 1 ];
40             moveItem--;
41          } // end while
42
43          data[ moveItem ] = insert; // place inserted element
44          PrintPass( next, moveItem ); // output pass of algorithm
45       } // end for
46    } // end method Sort
47
48    // print a pass of the algorithm
49    public void PrintPass( int pass, int index )
50    {
51       Console.Write( "after pass {0}: ", pass );
52
53       // output elements till swapped item
54       for ( int i = 0; i < index; i++ )
55          Console.Write( data[ i ] + "  " );
56
57       Console.Write( data[ index ] + "* " ); // indicate swap
```

Keep track of where to insert the element

Loop to locate the correct position to insert the element

Moves an element to the right and decrement the position at which to insert the next element

Inserts the element in place

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
58
59         // finish outputting array
60         for ( int i = index + 1; i < data.Length; i++ )
61            Console.Write( data[ i ] + "  " );
62
63         Console.Write( "\n              " ); // for alignment
64
65         // indicate amount of array that is sorted
66         for( int i = 0; i <= pass; i++ )
67            Console.Write( "--  " );
68         Console.WriteLine( "\n" ); // skip a line in output
69      } // end method PrintPass
70
71      // method to output values in array
72      public override string ToString()
73      {
74         string temporary = "";
75
76         // iterate through array
77         foreach ( int element in data )
78            temporary += element + "  ";
79
80         temporary += "\n"; // add newline character
81         return temporary;
82      } // end method ToString
83   } // end class InsertionSort
```

InsertionSort.cs

(3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

InsertionSortTest.cs

(1 of 2)

```
1  // Fig 24.9: InsertionSortTest.cs
2  // Test the insertion sort class.
3  using System;
4
5  public class InsertionSortTest
6  {
7     public static void Main( string[] args )
8     {
9        // create object to perform insertion sort
10       InsertionSort sortArray = new InsertionSort( 10 );
11
12       Console.WriteLine( "Unsorted array:" );
13       Console.WriteLine( sortArray ); // print unsorted array
14
15       sortArray.Sort(); // sort array
16
17       Console.WriteLine( "Sorted array:" );
18       Console.WriteLine( sortArray ); // print sorted array
19    } // end Main
20 } // end class InsertionSortTest
```

Create an array of random integers

Perform insertion sort on the array

© 2006 Pearson Education, Inc. All rights reserved.

```
Unsorted array:
12   27   36   28   33   92   11   93   59   62

after pass 1: 12   27*  36   28   33   92   11   93   59   62
              --   --
after pass 2: 12   27   36*  28   33   92   11   93   59   62
              --   --   --
after pass 3: 12   27   28*  36   33   92   11   93   59   62
              --   --   --   --
after pass 4: 12   27   28   33*  36   92   11   93   59   62
              --   --   --   --   --
after pass 5: 12   27   28   33   36   92*  11   93   59   62
              --   --   --   --   --   --
after pass 6: 11*  12   27   28   33   36   92   93   59   62
              --   --   --   --   --   --   --
after pass 7: 11   12   27   28   33   36   92   93*  59   62
              --   --   --   --   --   --   --   --
after pass 8: 11   12   27   28   33   36   59*  92   93   62
              --   --   --   --   --   --   --   --   --
after pass 9: 11   12   27   28   33   36   59   62*  92   93
              --   --   --   --   --   --   --   --   --   --

Sorted array:
11   12   27   28   33   36   59   62   92   93
```

Outline

InsertionSortTest.cs

(2 of 2)

© 2006 Pearson Education, Inc. All rights reserved.

24.3 Sorting Algorithms (Cont.)

- **Merge Sort**
  - **Sorts an array by splitting it into two equal-sized subarrays**
    - **Sort each subarray and merge them into one larger array**
  - **With an odd number of elements, the algorithm still creates two subarrays**
    - **One subarray will have one more element than the other**
  - **Merge sort is an efficient sorting algorithm: *O(n log n)***
    - **Conceptually more complex than selection sort and insertion sort**

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Figure 24.10: MergeSort.cs
2  // Class that creates an array filled with random integers.
3  // Provides a method to sort the array with merge sort.
4  using System;
5
6  public class MergeSort
7  {
8     private int[] data; // array of values
9     private static Random generator = new Random();
10
11    // create array of given size and fill with random integers
12    public MergeSort( int size )
13    {
14       data = new int[ size ]; // create space for array
15
16       // fill array with random ints in range 10-99
17       for ( int i = 0; i < size; i++ )
18          data[ i ] = generator.Next( 10, 100 );
19    } // end MergeSort constructor
20
21    // calls recursive SortArray method to begin merge sorting
22    public void Sort()
23    {
24       SortArray( 0, data.Length - 1 ); // sort entire array
25    } // end method Sort
```

The arguments are the beginning and ending indices of the array to be sorted

`MergeSort.cs`

(1 of 5)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

Test base case: if size of the array is 1, the array is already sorted

MergeSort.cs

Calculate the middle elements

Recursively call `SortArray` on the two halves of the array

Combine the two sorted arrays into one larger sorted array

```
26
27    // splits array, sorts subarrays and merges subarrays into sorted array
28    private void SortArray( int low, int high )
29    {
30       // test base case; size of array equals 1
31       if ( ( high - low ) >= 1 ) // if not base case
32       {
33          int middle1 = ( low + high ) / 2; // calculate middle of array
34          int middle2 = middle1 + 1; // calculate next element over
35
36          // output split step
37          Console.WriteLine( "split:    " + Subarray( low, high ) );
38          Console.WriteLine( "          " + Subarray( low, middle1 ) );
39          Console.WriteLine( "          " + Subarray( middle2, high ) );
40          Console.WriteLine();
41
42          // split array in half; sort each half (recursive calls)
43          SortArray( low, middle1 ); // first half of array
44          SortArray( middle2, high ); // second half of array
45
46          // merge two sorted arrays after split calls return
47          Merge( low, middle1, middle2, high );
48       } // end if
49    } // end method SortArray
```

© 2006 Pearson Education, Inc. All rights reserved.

```
50
51    // merge two sorted subarrays into one sorted subarray
52    private void Merge( int left, int middle1, int middle2, int right )
53    {
54       int leftIndex = left; // index into left subarray
55       int rightIndex = middle2; // index into right subarray
56       int combinedIndex = left; // index into temporary working array
57       int[] combined = new int[ data.Length ]; // working array
58
59       // output two subarrays before merging
60       Console.WriteLine( "merge:    " + Subarray( left, middle1 ) );
61       Console.WriteLine( "          " + Subarray( middle2, right ) );
62
63       // merge arrays until reaching end of either
64       while ( leftIndex <= middle1 && rightIndex <= right )
65       {
66          // place smaller of two current elements into result
67          // and move to next space in arrays
68          if ( data[ leftIndex ] <= data[ rightIndex ] )
69             combined[ combinedIndex++ ] = data[ leftIndex++ ];
70          else
71             combined[ combinedIndex++ ] = data[ rightIndex++ ];
72       } // end while
```

Outline

Temporary local variables to store the indices of the array for merging

(3 of 5)

Loop until the application reaches the end of either subarray

Test which element at the beginning of the arrays is smaller

Places the smaller element into position in the combined array

© 2006 Pearson Education, Inc. All rights reserved.

MergeSort.cs

```
73
74          // if left array is empty
75          if ( leftIndex == middle2 )
76             // copy in rest of right array
77             while ( rightIndex <= right )
78                combined[ combinedIndex++ ] = data[ rightIndex++ ];
79          else // right array is empty
80             // copy in rest of left array
81             while ( leftIndex <= middle1 )
82                combined[ combinedIndex++ ] = data[ leftIndex++ ];
83
84          // copy values back into original array
85          for ( int i = left; i <= right; i++ )
86             data[ i ] = combined[ i ];
87
88          // output merged array
89          Console.WriteLine( "           " + Subarray( left, right ) );
90          Console.WriteLine();
91       } // end method Merge
92
93       // method to output certain values in array
94       public string Subarray( int low, int high )
95       {
96          string temporary = "";
97
98          // output spaces for alignment
99          for ( int i = 0; i < low; i++ )
100            temporary += "   ";
```

Test whether left array has reached its end

Fill the combined array with the remaining elements of the right array

If the right array has reached its end, fill the combined array with the remaining elements of the left array

Copy the combined array into the original array

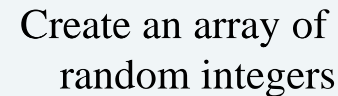
© 2006 Pearson Education, Inc. All rights reserved.

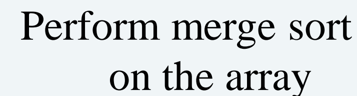
```
101
102       // output elements left in array
103       for ( int i = low; i <= high; i++ )
104          temporary += " " + data[ i ];
105
106       return temporary;
107    } // end method Subarray
108
109    // method to output values in array
110    public override string ToString()
111    {
112       return Subarray( 0, data.Length - 1 );
113    } // end method ToString
114 } // end class MergeSort
```

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
1  // Figure 24.11: MergeSortTest.cs
2  // Test the merge sort class.
3  using System;
4
5  public class MergeSortTest
6  {
7     public static void Main( string[] args )
8     {
9        // create object to perform merge sort
10       MergeSort sortArray = new MergeSort( 10 );
11
12       // print unsorted array
13       Console.WriteLine( "Unsorted: {0}\n", sortArray );
14
15       sortArray.Sort(); // sort array
16
17       // print sorted array
18       Console.WriteLine( "Sorted: {0}", sortArray );
19    } // end Main
20 } // end class MergeSortTest
```





MergeSortTest.cs

(1 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

MergeSortTest.cs

(2 of 3)

```
Unsorted:  36 38 81 93 85 72 31 11 33 74
split:    36 38 81 93 85 72 31 11 33 74
          36 38 81 93 85
                         72 31 11 33 74
split:    36 38 81 93 85
          36 38 81
                   93 85
split:    36 38 81
          36 38
                81
split:    36 38
          36
             38

merge:    36
             38
          36 38
merge:    36 38
                81
          36 38 81
split:             93 85
                   93
                      85
merge:             93
                      85
                   85 93
merge:    36 38 81
                   85 93
          36 38 81 85 93
```

(continued)

© 2006 Pearson Education, Inc. All rights reserved.

Outline

```
split:                    72 31 11 33 74
                          72 31 11
                                   33 74

split:                    72 31 11
                          72 31
                                11

split:                    72 31
                          72
                             31

merge:                    72
                             31
                          31 72

merge:                    31 72
                                11
                          11 31 72

split:                             33 74
                                   33
                                      74

merge:                             33
                                      74
                                   33 74

merge:                    11 31 72
                                   33 74
                          11 31 33 72 74

merge:    36 38 81 85 93
                          11 31 33 72 74
          11 31 33 36 38 72 74 81 85 93

Sorted:  11 31 33 36 38 72 74 81 85 93
```

MergeSortTest.cs

(3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

Algorithm	Location	Big O
Searching Algorithms:		
Linear Search	Section 24.2.1	$O(n)$
Binary Search	Section 24.2.2	$O(\log n)$
Recursive Linear Search	Exercise 24.8	$O(n)$
Recursive Binary Search	Exercise 24.9	$O(\log n)$
Sorting Algorithms:		
Selection Sort	Section 24.3.1	$O(n^2)$
Insertion Sort	Section 24.3.2	$O(n^2)$
Merge Sort	Section 24.3.3	$O(n \log n)$
Bubble Sort	Exercises 24.5–24.6	$O(n^2)$

Fig. 24.12 | Searching and sorting algorithms with Big O values.

© 2006 Pearson Education, Inc. All rights reserved.

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}

Fig. 24.13 | Number of comparisons for common Big O notations.

© 2006 Pearson Education, Inc. All rights reserved.