

25

Searching and Sorting

© 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.
- © 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.

- ### 24.2 Searching Algorithms
- Linear Search**
 - Searches each element in an array sequentially
- © 2006 Pearson Education, Inc. All rights reserved.

```

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

```

Outline

LInearArray.cs
(1 of 2)

- Fill i n t 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.

```

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

```

Outline

LInearSearch Test.cs
(3 of 3)

© 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

© 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
    
```

Outline

BinaryArray.cs
(1 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```

30     do // loop to search for element
31     {
32         // if the element is found at the middle
33         if ( searchElement == data[ middle ] )
34             location = middle; // location is the current middle
35
36         // middle element is too high
37         else if ( searchElement < data[ middle ] )
38             high = middle - 1; // eliminate the higher half
39         else // middle element is too low
40             low = middle + 1; // eliminate the lower half
41
42         middle = ( low + high + 1 ) / 2; // recalculate the middle
43     } while ( ( low <= high ) && ( location == -1 ) );
44
45     return location; // return location of search key
46 } // end method BinarySearch
    
```

Outline

BinaryArray.cs
(2 of 3)

© 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.

```

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
    
```

Outline

SelectionSort.cs
(1 of 3)

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

© 2006 Pearson Education, Inc. All rights reserved.

```

31 // loop to find index of smallest element
32 for ( int index = 1 + 1; index < data.Length; index++ )
33     if ( data[ index ] < data[ smallest ] )
34         smallest = index;
35
36 Swap( 1, smallest ); // swap smallest element into position
37 PrintPass( 1, array, 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.WriteLine( "after pass {0}: ", pass );
53
54     // output elements through the selected item
55     for ( int i = 0; i < index; i++ )
56         Console.WriteLine( data[ i ] + " " );
57
58     Console.WriteLine( data[ index ] + " " ); // indicate swap

```

Determine the index of the remaining smallest element

Place the smallest remaining element in the next spot

(2 of 3)

Swap two elements

© 2006 Pearson Education, Inc. All rights reserved.

```

59 // finish outputting array
60 for ( int i = index + 1; i < data.Length; i++ )
61     Console.WriteLine( data[ i ] + " " );
62
63 Console.WriteLine( "\n" ); // for alignment
64
65 // indicate amount of array that is sorted
66 for ( int j = 0; j < pass; j++ )
67     Console.WriteLine( "-- " );
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 SelectionSort
84 } // end class SelectionSort

```

Outline

SelectionSort.cs

(3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```

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

```

Outline

SelectionSortTest.cs

(1 of 2)

Create an array of random integers

Perform selection sort on the array

© 2006 Pearson Education, Inc. All rights reserved.

```

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

```

Outline

SelectionSortTest.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.

```

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 ];

```

Outline

InsertionSort.cs

(1 of 3)

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

© 2006 Pearson Education, Inc. All rights reserved.

```

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

```

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.

```

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

```

Outline

InsertionSort.cs

(3 of 3)

© 2006 Pearson Education, Inc. All rights reserved.

```

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 array of random integers
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

```

Outline

InsertionSortTest.cs

(1 of 2)

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.