The Efficiency of Algorithms

Chapter 4

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Contents

- Motivation
- Measuring an Algorithm's Efficiency
  - Counting Basic Operations
  - Best, Worst, and Average Cases
- Big Oh Notation
  - The Complexities of Program Constructs

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Contents

- Picturing Efficiency
- The Efficiency of Implementations of the ADT Bag
  - An Array-Based Implementation
  - A Linked Implementation
  - Comparing the Implementations

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Objectives

- Assess efficiency of given algorithm
- Compare expected execution times of two methods
  - Given efficiencies of algorithms

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Motivation, which is better?

- Contrast algorithms

Algorithm A	Algorithm B	Algorithm C

sum = 0
for i = 1 *to* n
    sum = sum + i
	sum = 0
	for i = 1 *to* n
	{
	    for j = 1 *to* i
	        sum = sum + 1
	}
		sum = n * (n + 1) / 2

Figure 4-1 Three algorithms for computing the sum 1 + 2 + . . . + n for an integer n > 0

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Comparing the arithmetic

FIGURE 4-2 The number of basic operations required by the algorithms in Figure 4-1

	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n(n+1)/2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2+n)/2$	**3**

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 1 For any positive integer n, the identity

$$1 + 2 + . . . + n = n (n + 1) / 2$$

is one that you will encounter while analyzing algorithms. Can you derive it? If you can, you will not need to memorize it. *Hint*: Write $1 + 2 + . . . + n$. Under it write $n + (n - 1) + . . . + 1$. Then add the terms from left to right.

Question 2 Can you derive the values in Figure 4-2? *Hint*: For Algorithm B, use the identity given in Question 1.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

1. If you follow the hint given in the question, you will get the sum of n occurrences of $n + 1$, which is $(n + 1) + (n + 1) + \ldots + (n + 1)$. This sum is simply the product $n(n + 1)$. To get this sum, we added $1 + 2 + \ldots + n$ to itself. Thus, $n(n + 1)$ is $2(1 + 2 + \ldots + n)$. The desired conclusion follows immediately from this fact.

2. Algorithm A: The loop iterates n times, so there are n additions and a total of $n + 1$ assignments. We ignore the assignments.
Algorithm B: For each value of i, the inner loop iterates i times, and so performs i additions and i assignments. The outer loop iterates n times. Together, the loops perform $1 + 2 + \ldots + n$ additions and the same number of assignments. Using the identity given in Question 1, the number of additions is $n(n + 1) / 2$. The additional assignment to set `sum` to zero makes the total number of assignments equal to $1 + n(n + 1) / 2$, which we ignore.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Motivation

- Java code for algorithms

```
// Algorithm A
long sum = 0;
for (long i = 1; i <= n; i++)
   sum = sum + i;
System.out.println(sum);
```

```
// Algorithm B
sum = 0;
for (long i = 1; i <= n; i++)
{
   for (long j = 1; j <= i; j++)
      sum = sum + 1;
} // end for
System.out.println(sum);
```

```
// Algorithm C
sum = n * (n + 1) / 2;
System.out.println(sum);
```

- Even a simple program can be inefficient

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Measuring an Algorithm's Efficiency

- Complexity
  - Space and time requirements
- Other issues for best solution
  - Generality of algorithm
  - Programming effort
  - Problem size – number of items program will handle
  - Growth-rate function

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Counting Basic Operations

	Algorithm A	Algorithm B	Algorithm C
Additions	n	$n(n + 1) / 2$	1
Multiplications			1
Divisions			1
Total basic operations	n	$(n^2 + n) / 2$	**3**

Figure 4-2 The number of basic operations required by the algorithms in Figure 4-1

Copyright ©2012 by Pearson Education, Inc. All rights reserved

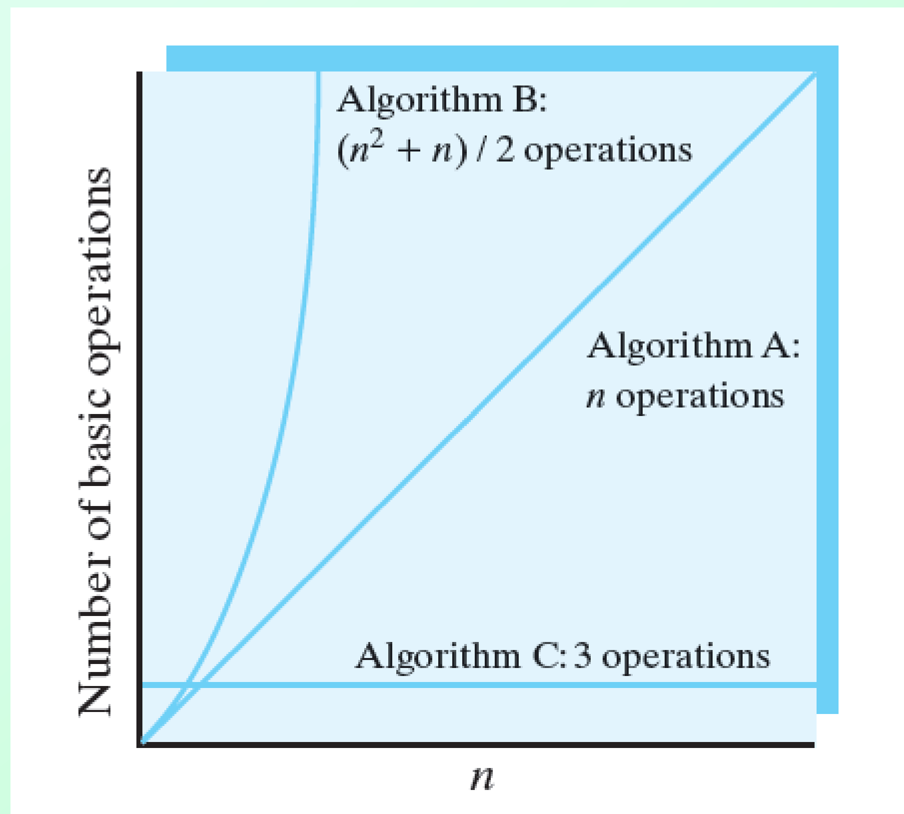

Figure 4-3 The number of basic operations required by the algorithms in Figure 4-1 as a function of n

Copyright ©2012 by Pearson Education, Inc. All rights reserved

n	$\log(\log n)$	$\log n$	$\log^2 n$	n	$n \log n$	n^2	n^3	2^n	$n!$
10	2	3	11	10	33	10^2	10^3	10^3	10^5
10^2	3	7	44	100	664	10^4	10^6	10^{30}	10^{94}
10^3	3	10	99	1000	9966	10^6	10^9	10^{301}	10^{1435}
10^4	4	13	177	10,000	132,877	10^8	10^{12}	10^{3010}	$10^{19,335}$
10^5	4	17	276	100,000	1,660,964	10^{10}	10^{15}	$10^{30,103}$	$10^{243,338}$
10^6	4	20	397	1,000,000	19,931,569	10^{12}	10^{18}	$10^{301,030}$	$10^{2,933,369}$

FIGURE 4-4 Typical growth-rate functions evaluated at increasing values of n

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Best, Worst, and Average Cases

- Some algorithms depend only on size of data set
- Other algorithms depend on nature of the data
  - Best case search when item at beginning
  - Worst case when item at end
  - Average case somewhere between

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Big Oh Notation

- Notation to describe algorithm complexity
- Definition
  A function $f(n)$ is of order at most $g(n)$ that is, $f(n)$ is $O(g(n))$—if :
- A positive real number c and positive integer N exist such that $f(n) \le c \cdot g(n)$ or all $n \geq N$.
- That is, $c \cdot g(n)$ is an upper bound on $f(n)$ when n is sufficiently large.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

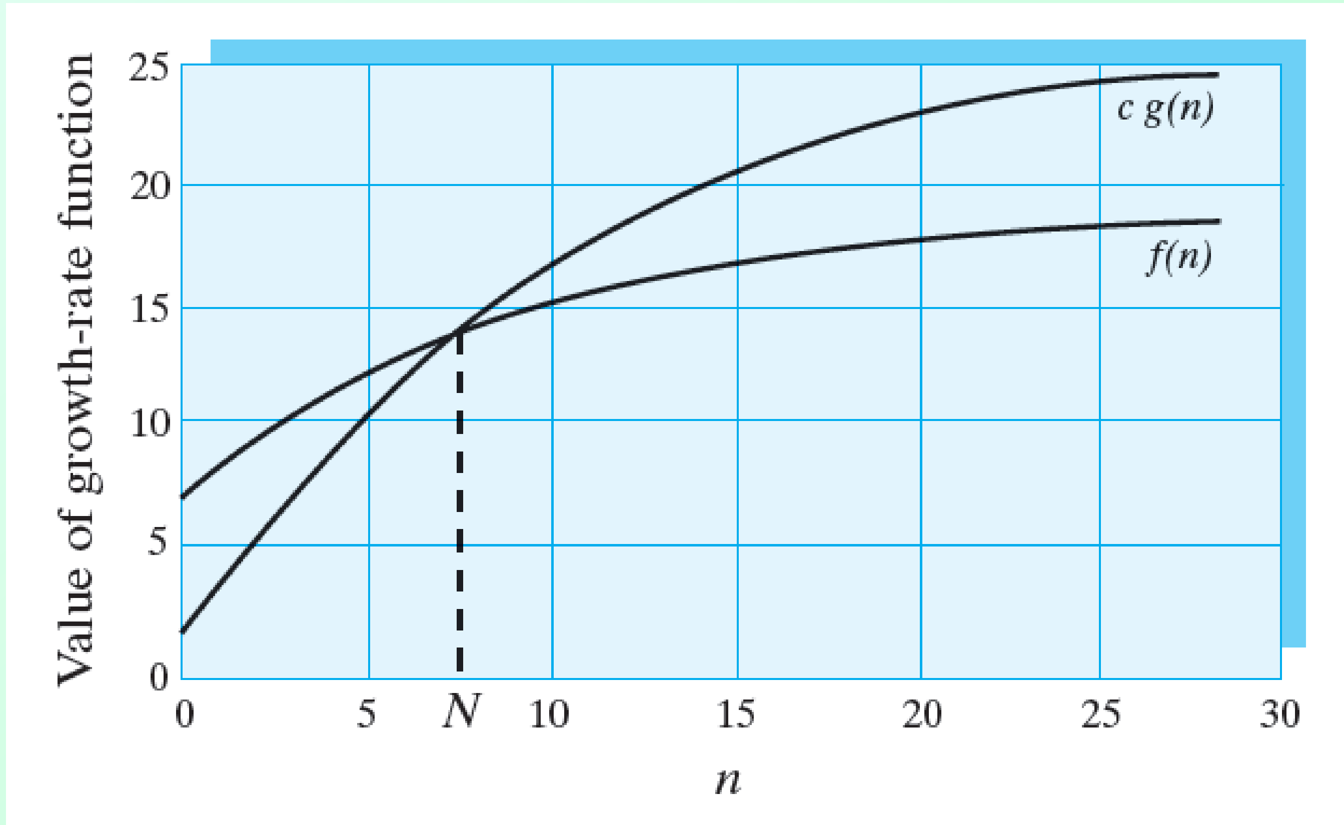

Figure 4-5 An illustration of the definition of Big Oh

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 3 Show that $3n^2 + 2^n$ is $O(2^n)$. What values of c and N did you use?

Copyright ©2012 by Pearson Education, Inc. All rights reserved

3. $3n^2 + 2^n < 2^n + 2^n = 2 \times 2^n$ when $n \geq 8$. So $3n^2 + 2^n = O(2^n)$, using $c = 2$ and $N = 8$.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Big Oh Identities

- $O(k\,g(n)) = O(g(n))$ for a constant k
- $O(g1(n)) + O(g2(n)) = O(g1(n) + g2(n))$
- $O(g1(n)) \times O(g2(n)) = O(g1(n) \times g2(n))$
- $O(g1(n) + g2(n) + \ldots + gm(n)) = O(\max(g1(n), g2(n), \ldots, gm(n))$
- $O(\max(g1(n), g2(n), \ldots, gm(n)) = \max(O(g1(n)), O(g2(n)), \ldots, O(gm(n)))$

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 4 If $P_k(n) = a_0 n^k + a_1 n^{k-1} + \ldots + a_k$ for $k > 0$ and $n > 0$, what is $O(P_k(n))$?

Copyright ©2012 by Pearson Education, Inc. All rights reserved

4. n^k.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

```
for i = 1 to n
    sum = sum + i
```

Figure 4-6 An O(n) algorithm

Copyright ©2012 by Pearson Education, Inc. All rights reserved

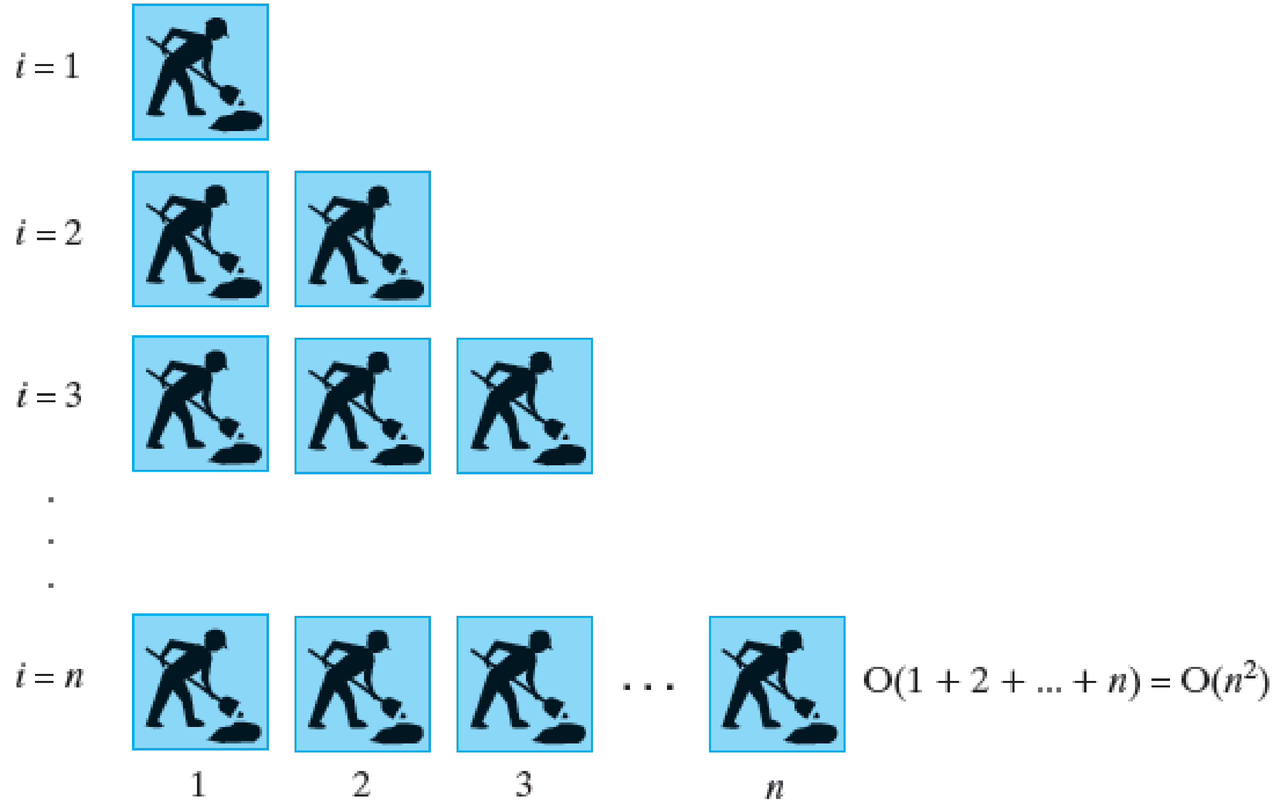

Figure 4-7 An $O(n^2)$ algorithm

Copyright ©2012 by Pearson Education, Inc. All rights reserved

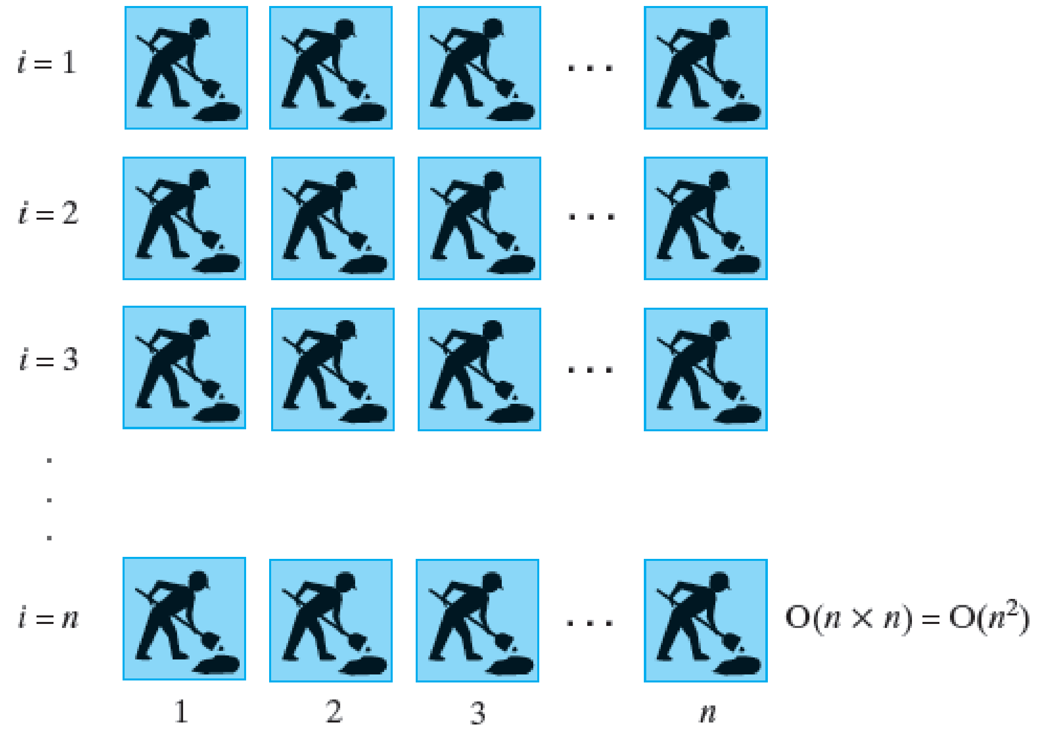

Figure 4-8 Another $O(n^2)$ algorithm

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 5 Using Big Oh notation, what is the order of the following computation's time requirement?

```
for i = 1 to n
{
    for j = 1 to 5
        sum = sum + 1
}
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

5. The inner loop requires a constant amount of time, and so it is $O(1)$. The outer loop is $O(n)$, and so the entire computation is $O(n)$.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Growth-Rate Function for Size n Problems	Growth-Rate Function for Size $2n$ Problems	Effect on Time Requirement
1	1	None
$\log n$	$1 + \log n$	Negligible
n	$2n$	Doubles
$n \log n$	$2n \log n + 2n$	Doubles and then adds $2n$
n^2	$(2n)^2$	Quadruples
n^3	$(2n)^3$	Multiplies by 8
2^n	2^{2n}	Squares

Figure 4-9 The effect of doubling the problem size on an algorithm's time requirement

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 6 Suppose that you can solve a problem of a certain size on a given computer in time t by using an $O(n)$ algorithm. If you double the size of the problem, how fast must your computer be to solve the problem in the same time?

Question 7 Repeat the previous question, but instead use an $O(n^2)$ algorithm.

Question 8 The following algorithm discovers whether an array contains duplicate entries within its first n elements. What is the Big Oh of this algorithm in the worst case?

```
Algorithm hasDuplicates(array, n)
for index = 0 to n - 2
    for rest = index + 1 to n - 1
        if (array[index] equals array[rest])
            return true
return false
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

6. Twice as fast.

7. Four times as fast.

8. Let's tabulate the maximum number of times the inner loop executes for various values of `index`:

`index`	Inner Loop Iterations
0	$n - 1$
1	$n - 2$
2	$n - 3$
. . .	. . .
$n - 2$	1

As you can see, the maximum number of times the inner loop executes is $1 + 2 + . . . + n - 1$, which is $n (n - 1) / 2$. Thus, the algorithm is $O(n^2)$ in the worst case.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Growth-Rate Function g	$g(10^6) / 10^6$
$\log n$	0.0000199 seconds
n	1 second
$n \log n$	19.9 seconds
n^2	11.6 days
n^3	31,709.8 years
2^n	$10^{301,016}$ years

Figure 4-10 The time required to process one million items by algorithms of various orders at the rate of one million operations per second

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Array Based Implementation

- Adding an entry to a bag, an O(1) method,

```
public boolean add(T newEntry)
{
   boolean result = true;
   if (isFull())
   {
      result = false;
   }
   else
   { // assertion: result is true here
      bag[numberOfEntries] = newEntry;
      numberOfEntries++;
   } // end if

   return result;
} // end add
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Array Based Implementation

- Searching for an entry, O(1) best case, O(n) worst or average case
- Thus an O(n) method overall

```
private int getIndexOf(T anEntry)
{
    int where = -1;
    boolean found = false;

    for (int index = 0; !found && (index < numberOfEntries); index++)
    {
        if (anEntry.equals(bag[index]))
        {
            found = true;
            where = index;
        } // end if
    } // end for
    return where;
} // end getIndexOf
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 9 What is the Big Oh of the bag's `remove` methods? Assume that a fixed-size array represents the bag, and use an argument similar to the one we just made for `contains`.

Question 10 Repeat Question 9, but instead analyze the method `getFrequencyOf`.

Question 11 Repeat Question 9, but instead analyze the method `toArray`.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

9. Removing an unspecified entry is $O(1)$. Removing a particular entry is $O(1)$ in the best case and $O(n)$ in the worst and average cases.

10. $O(n)$.

11. $O(n)$.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

A *Linked* Implementation

- Adding an entry to a bag, an O(1) method,

```
public boolean add(T newEntry) // OutOfMemoryError possible
{
   // add to beginning of chain:
   Node newNode = new Node(newEntry);
   newNode.next = firstNode;    // make new node reference rest of chain
                                // (firstNode is null if chain is empty)
   firstNode = newNode;         // new node is at beginning of chain
   numberOfEntries++;

   return true;
} // end add
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

A *Linked* Implementation

- Searching a bag for a given entry, O(1) best case, O(n) worst case
- O(n) overall

```
public boolean contains(T anEntry)
{
   boolean found = false;
   Node currentNode = firstNode;

   while (!found && (currentNode != null))
   {
      if (anEntry.equals(currentNode.data))
         found = true;
      else
         currentNode = currentNode.next;
   } // end while

   return found;
} // end contains
```

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Question 12 What is the Big Oh of the method `contains` when it searches for an entry that is not in the bag? Assume that a chain of linked nodes represents the bag.

Question 13 What is the Big Oh of the bag's `remove` methods? Assume that a chain of linked nodes represents the bag, and use an argument similar to the one you just made for `contains`.

Question 14 Repeat Question 13, but instead analyze the method `getFrequencyOf`.

Question 15 Repeat Question 13, but instead analyze the method `toArray`.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

12. $O(n)$.

13. Removing an unspecified entry is $O(1)$. Removing a particular entry is $O(1)$ in the best case and $O(n)$ in the worst and average cases.

14. $O(n)$.

15. $O(n)$.

Copyright ©2012 by Pearson Education, Inc. All rights reserved

Operation	Fixed-Size Array	Linked
`add(newEntry)`	O(1)	O(1)
`remove()`	O(1)	O(1)
`remove(anEntry)`	O(1), O(n), O(n)	O(1), O(n), O(n)
`clear()`	O(n)	O(n)
`getFrequencyOf(anEntry)`	O(n)	O(n)
`contains(anEntry)`	O(1), O(n), O(n)	O(1), O(n), O(n)
`toArray()`	O(n)	O(n)
`getCurrentSize(), isEmpty(), isFull()`	O(1)	O(1)

Figure 4-11 The time efficiencies of the ADT bag operations for two implementations, expressed in Big Oh notation

Copyright ©2012 by Pearson Education, Inc. All rights reserved

End

Chapter 4

Copyright ©2012 by Pearson Education, Inc. All rights reserved