Solutions Manual for Data Structures with Java

John R. Hubbard
Anita Huray
University of Richmond

Chapter 1
Object-Oriented Programming

Exercises

1.1 The requirements stage would generate a user manual like that shown here:

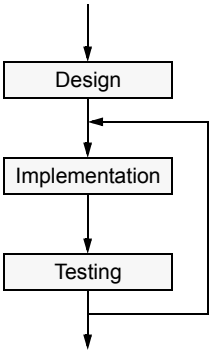
User Manual
Enter the string: **java CelsiusToFahrenheit <*first*> <*last*> <*incr*>**
at the command line, where **<*first*>** is the first Celsius temperature to be converted, **<*first*>** is the last Celsius temperature, and **<*incr*>** is the Celsius increment for the table. The output will be a conversion table with that range and increment.
Example:

Input: **java Convert 0 40 10**
Output: **032**
1050
2068
3086
40104

The design stage could adapt the same class as shown in Listing 1.1.
The implementation stage could adapt the same class as shown in Listing 1.2.
The testing stage could run a test driver like this:

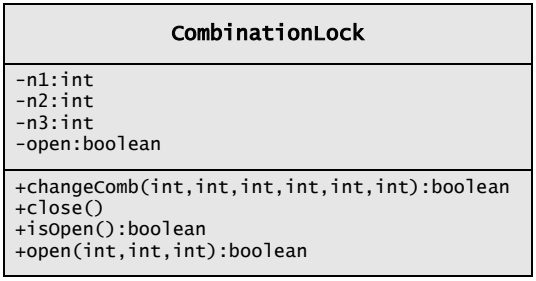
```
public class CelsiusToFahrenheit {
  public static void main(String[] args) {
    if (args.length!=3) exit();
    double first = Double.parseDouble(args[0]);
    double last = Double.parseDouble(args[1]);
    double incr = Double.parseDouble(args[2]);
    for (double i=first; i<=last; i += incr)
      System.out.println(i + "\t" + new MyTemperature(value,'F') );
  }
```

```
    private static void exit() {
      System.out.println(
       "Usage: java CelsiusToFahrenheit <first> <last> <incr>"
        + "\nwhere:"
        + "\t<first> is the first celsius temperature to be listed"
        + "\t<last> is the last celsius temperature to be listed"
        + "\t<incr> is the increment"
        + "\nExample:"
        + "\tjava CelsiusToFahrenheit 0 40 4"
      );
      System.exit(0);
    }
  }
```



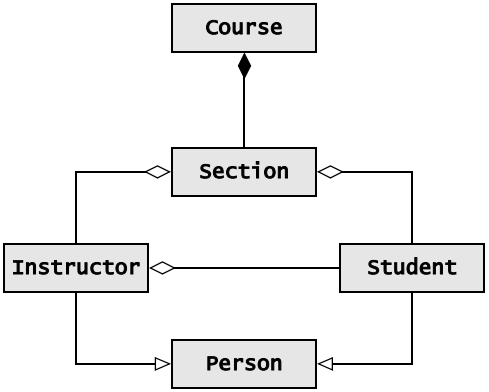
1.2 Another likely cycle is shown in here. This is the common Debug Cycle, consisting of repeated two-part test-and-correct step.

1.3

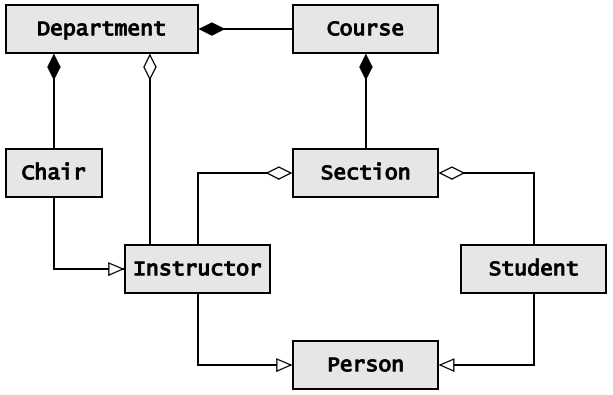


1.4 If d is a divisor of n that is greater than $\sqrt{n}$, then $m = n/d$ must be a whole number (*i.e.*, an integer), and therefore another divisor of n. But since d > $\sqrt{n}$, we have $m = n/d < n/\sqrt{n} = \sqrt{n}$. So by checking for divisors only among those integers that are less than $\sqrt{n}$, the existence of the divisor $d > \sqrt{n}$ will be found indirectly from $m = n/d < \sqrt{n}$.

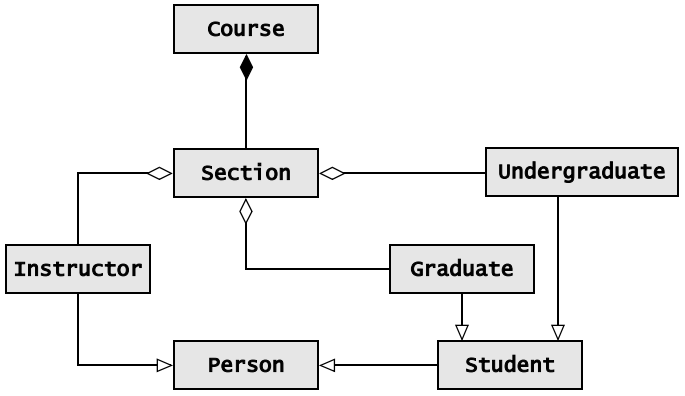
1.5



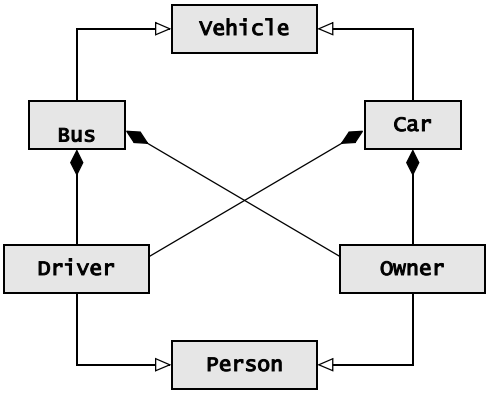
1.6


Department
Course
Chair
Section
Instructor
Student
Person

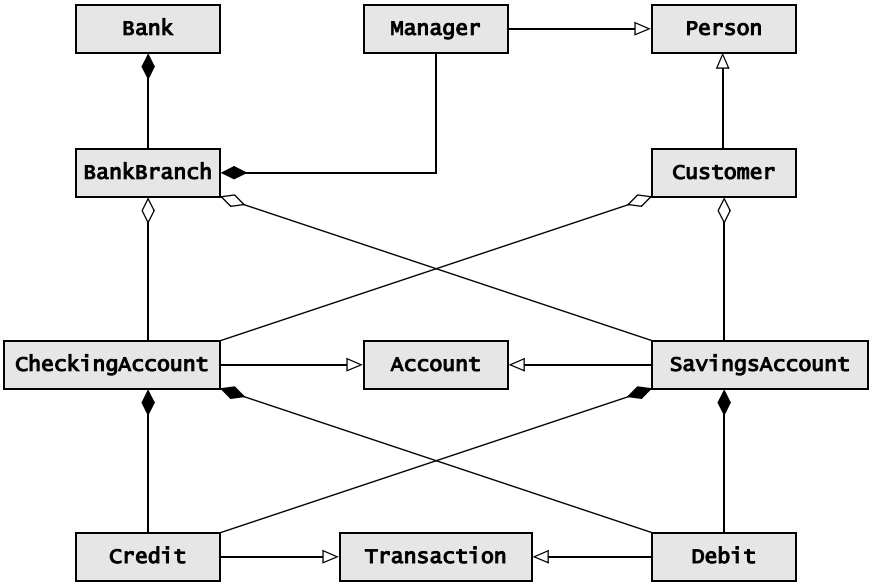
1.7


Course
Section
Undergraduate
Instructor
Graduate
Person
Student

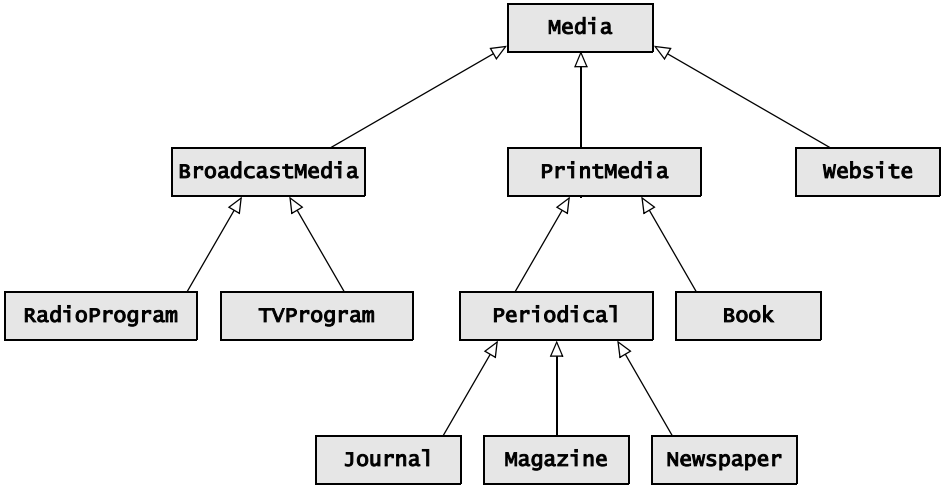
1.8


Vehicle
Bus
Car
Driver
Owner
Person

1.9



1.10



Programming Problems

1.1

```
/**
 *  An interface for representing temperatures, with functionality
 *  for converting their values between Celsius and Fahrenheit.
 *  @author  John R. Hubbard
 *  @see     MyTemperature
 */
```

```
public interface Temperature {
  /**  @return  the Celsius value for this temperature.    */
  public double getCelsius();
  /**  @return  the Fahrenheit value for this temperature.    */
  public double getFahrenheit();
  /**  @return  the Kelvin value for this temperature.    */
  public double getKelvin();
  /**  @param  celsius  the Celsius value for this temperature. */
  public void setCelsius(double celsius);
  /**  @param  fahrenheit  the Fahrenheit value for this temp. */
  public void setFahrenheit(double fahrenheit);
  /**  @param  kelvin  the Kelvin value for this temperature.*/
  public void setKelvin(double kelvin);
}

public class MyTemperature implements Temperature {
  private double celsius;    // stores temperature as a Celsius value
  public MyTemperature(double value, char scale) {
    if (scale=='C') setCelsius(value);
    if (scale=='F') setFahrenheit(value);
    else setKelvin(value);
  }
  public double getCelsius() {
    return celsius;
  }
  public double getFahrenheit() {
    return 9*celsius/5 + 32.0;
  }
  public double getKelvin() {
    return celsius + 273.16;
  }
  public void setCelsius(double celsius) {
    this.celsius = celsius;
  }
  public void setFahrenheit(double fahrenheit) {
    this.celsius = 5*(fahrenheit - 32)/9;
  }
  public void setKelvin(double kelvin) {
    this.celsius = kelvin - 273.16;
  }
  public String toString() {
    // Example: "25.0 C = 77.0 F"
```

```
      return round(getCelsius())+ " C = "
         + round(getFahrenheit())+ " F = "
         + round(getKelvin())+ " K";
    }
    private static double round(double x) {
      // returns x, rounded to one digit on the right of the decimal:
      return Math.round(10*x)/10.0;
    }
  }
```

1.2

```
  public class MyTemperature implements Temperature {
    private double celsius;  // stores temperature as a Celsius value
    private int digits;      // number of digits to right of decimal
    public MyTemperature(double value, char scale, int digits) {
      if (scale=='C') setCelsius(value);
      else setFahrenheit(value);
      this.digits = digits;
    }
    public double getCelsius() {
      return celsius;
    }
    public double getFahrenheit() {
      return 9*celsius/5 + 32.0;
    }
    public void setCelsius(double celsius) {
      this.celsius = celsius;
    }
    public void setDigits(int digits) {
      this.digits = digits;
    }
    public void setFahrenheit(double fahrenheit) {
      this.celsius = 5*(fahrenheit - 32)/9;
    }
    public String toString() {
      // Example: "25.0 C = 77.0 F"
      return round(getCelsius())+" C = "+round(getFahrenheit())+" F";
    }
    private double round(double x) {
      // returns x, rounded to one digit on the right of the decimal:
      double p = Math.pow(10,digits);
      return Math.round(p*x)/p;
    }
  }
```

```
public class Convert {
  public static void main(String[] args) {
    if (args.length!=3) exit();
    double value = Double.parseDouble(args[0]);  // convert string
    char scale = Character.toUpperCase(args[1].charAt(0));
    int digits = Integer.parseInt(args[2]);
    if (scale!='C' && scale!='F') exit();
    Temperature temperature= new MyTemperature(value, scale, digits);
    System.out.println(temperature);
  }
  private static void exit() {
    // prints usage message and then terminates the program:
    System.out.println(
     "Usage: java Convert <temperature> <scale> <digits>"
     + "\nwhere:"
     + "\t<temperature> is the temperature that you want to convert"
     + "\n\t<scale> is either \"C\" or \"F\"."
     + "\n\t<digits> is the number of digits to use right of decimal."
     + "\nExample: java Convert 67 F 2"
    );
    System.exit(0);
  }
}
```

1.3

```
public class MyTemperature implements Temperature {
  private static final String s = "###,###,###,###.################";
  private double celsius;  // stores temperature as a Celsius value
  private int digits;      // number of digits to right of decimal
  private DecimalFormat formatter;  // used for formatted output
  public MyTemperature(double value, char scale, int digits) {
    if (scale=='C') setCelsius(value);
    else setFahrenheit(value);
    this.digits = digits;
    setFormatter();
  }
  private void setFormatter() {
    String pattern = new String(s.toCharArray(), 0, 16 + digits);
    this.formatter = new DecimalFormat(pattern);
  }
  public double getCelsius() {
    return celsius;
  }
```

```
  public double getFahrenheit() {
    return 9*celsius/5 + 32.0;
  }
  public void setCelsius(double celsius) {
    this.celsius = celsius;
  }
  public void setDigits(int digits) {
    this.digits = digits;
    setFormatter();
  }
  public void setFahrenheit(double fahrenheit) {
    this.celsius = 5*(fahrenheit - 32)/9;
  }
  public String toString() {
    // Example: "25.0 C = 77.0 F"
    return formatter.format(getCelsius()) + " C = "
       + formatter.format(getFahrenheit()) + " F";
  }
}
public class Convert {
  public static void main(String[] args) {
    if (args.length!=3) exit();
    double value = Double.parseDouble(args[0]);  // convert string
    char scale = Character.toUpperCase(args[1].charAt(0));
    int digits = Integer.parseInt(args[2]);
    if (scale!='C' && scale!='F') exit();
    Temperature temperature= new MyTemperature(value, scale, digits);
    System.out.println(temperature);
  }
  private static void exit() {
    // prints usage message and then terminates the program:
    System.out.println(
     "Usage: java Convert <temperature> <scale> <digits>"
     + "\nwhere:"
     + "\t<temperature> is the temperature that you want to convert"
     + "\n\t<scale> is either \"C\" or \"F\"."
     + "\n\t<digits> is the number of digits to use right of decimal."
     + "\nExample: java Convert 67 F 2"
    );
    System.exit(0);
  }
}
```

1.4
```
public class TestPrimeAlgorithm {
  public static void main(String[] args) {
    Random random = new Random();
    for (int i=0; i<100; i++) {
      int n = random.nextInt(Integer.MAX_VALUE);
      if (isPrime(n)) System.out.print(n + " ");
    }
  }
  public static boolean isPrime(int n) {
    if (n < 2) return false;
    if (n < 4) return true;
    if (n%2 == 0) return false;
    for (int d=3; d*d <= n; d += 2)
      if (n%d == 0) return false;
    return true;
  }
}
```

1.5
```
public class Course {
  private float credit;
  private String dept;
  private String id;
  private String name;
  private Section[] sections = new Section[1000];
  public Course(String dept, String id, String name, float credit) {
    this.credit = credit;
    this.dept = dept;
    this.id = id;
    this.name = name;
  }
  public void add(Section section) {
    int i=0;
    while (sections[i] != null)
      ++i;
    sections[i] = section;
  }
  public String toString() {
    String s = dept + " " + id + " \"" + name + "\", "
                    + credit + " credits";
    for (int i=0; sections[i] != null; i++)
      s += sections[i];
    return s;
  }
```

```
  public static void main(String[] args) {
    Course course = new Course("CMSC", "221", "Data Structures", 4);
    System.out.println(course);
  }
}

public class Section {
  private Course course;
  private String place;
  private String term;
  private String time;
  private Instructor instructor;
  private Student[] students;
  public Section(Course course, String term, String p, String t) {
    this.course = course;
    this.term = term;
    this.place = p;
    this.time = t;
  }
  public Course getCourse() {
    return course;
  }
  public String getPlace() {
    return place;
  }
  public String getTerm() {
    return term;
  }
  public String getTime() {
    return time;
  }
  public Instructor getInstructor() {
    return instructor;
  }
  public Student[] getStudents() {
    return students;
  }
  public void setPlace(String place) {
    this.place = place;
  }
  public void setTerm(String term) {
    this.term = term;
```

```
  }
  public void setTime(String time) {
    this.time = time;
  }
  public void setInstructor(Instructor instructor) {
    this.instructor = instructor;
  }
  public void setStudents(Student[] students) {
    int n = students.length;
    // duplicate the array object:
    this.students = new Student[n];
    // but do not duplicate the Student objects:
    for (int i=0; i<n; i++)
      this.students[i] = students[i];
  }
  public String toString() {
    return course + ": " + term + ", " + place + ", " + time + ", "
                  + instructor;
  }
  public static void main(String[] args) {
    Course course = new Course("CMSC", "221", "Data Structures", 4);
    Section section = new Section(course, "Fall 2004", null, null);
    System.out.println(section);
  }
}

import java.util.*;
public class Person {
  protected int yob;
  protected String email;
  protected String id;
  protected boolean male;
  protected String name;
  public Person(String name, String id, String sex, int yob) {
    this.id = id;
    this.male = (sex.substring(0,1).toUpperCase() == "M");
    this.name = name;
    this.yob = yob;
  }
  public int getYob() {
    return yob;
  }
```

```
  public String getEmail() {
    return email;
  }
  public String getId() {
    return id;
  }
  public boolean isMale() {
    return male;
  }
  public String getName() {
    return name;
  }
  public void setEmail(String email) {
    this.email = email;
  }
  public String toString() {
    String string = name + ", " + id;
    if (male) string += " (M)";
    else string += " (F)";
    if (email != null) string += ", " + email;
    string += " (" + yob + ")";
    return string;
  }
  public static void main(String[] args) {
    Person grandson = new Person("C. Hubbard", "1.2.1", "M", 2002);
    System.out.println(grandson);
    grandson.setEmail("abced@fgh.ijk");
    System.out.println(grandson);
    System.out.println("\t name: " + grandson.name);
    System.out.println("\t   id: " + grandson.id);
    System.out.println("\t  sex: "+ (grandson.male?"male":"female"));
    System.out.println("\temail: " + grandson.email);
    System.out.println("\t  yob: " + grandson.yob);
  }
}

import java.util.*;
public class Student extends Person {
  protected String country;
  protected int credits;
  protected double gpa;
  public Student(String name, String id, String s, int y, String c) {
```

```
    super(name, id, s, y);
    this.country = c;
  }
  public int getCredits() {
    return credits;
  }
  public double getGpa() {
    return gpa;
  }
  public String getCountry() {
    return country;
  }
  public void setCredits(int credits) {
    this.credits = credits;
  }
  public void setGpa(double gpa) {
    this.gpa = gpa;
  }
  public void setCountry(List record) {
    this.country = country;
  }
  public static void main(String[] args) {
    Student student =
      new Student("Anne Miller", "200491", "F", 1985, "US");
    System.out.println(student);
  }
}

import java.util.*;
public class Instructor extends Person {
  protected String dept;
  protected String office;
  protected String tel;
  public Instructor(String name, String id, String sex, int yob) {
    super(name, id, sex, yob);
  }
  public String getDept() {
    return office;
  }
  public String getOffice() {
    return office;
  }
```

```
  public String getTel() {
    return tel;
  }
  public void setDept(String dept) {
    this.dept = dept;
  }
  public void setOffice(String office) {
    this.office = office;
  }
  public void setTel(String tel) {
    this.tel = tel;
  }
  public String toString() {
    String s =  super.toString();
    if (dept != null) s += ", " + dept;
    if (office != null) s += ", " + office;
    if (tel != null) s += " (" + tel + ")";
    return s;
  }
  public static void main(String[] args) {
  Instructor knuth = new Instructor("Don Knuth","8122063","M",1938);
  System.out.println(knuth);
    knuth.setDept("CS");
  System.out.println(knuth);
  }
}
```

1.6

```
public class ComboLock {
  private int n1, n2, n3;
  private boolean open;
  public ComboLock(int n1, int n2, int n3) {
    this.n1 = n1;
    this.n2 = n2;
    this.n3 = n3;
  }
  public boolean changeComb(int n1, int n2, int n3, int n4,
                            int n5, int n6) {
   if (this.n1 != n1 || this.n2 != n2 || this.n3 != n3) return false;
    this.n1 = n4;
    this.n2 = n5;
    this.n3 = n6;
    open = false;
    return true;
```

```
  }
  public void close() {
    open = false;
  }
  public boolean isOpen() {
    return open;
  }
  public boolean open(int n1, int n2, int n3) {
    if (this.n1 == n1 && this.n2 == n2 && this.n3 == n3) open = true;
    else open = false;
    return open;
  }
}

public class TestComboLock {
  public static void main(String[] args) {
    ComboLock lock = new ComboLock(10, 20, 30);
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.open(10,20,30): "+lock.open(10,20,30));
    System.out.println("lock.isOpen(): " + lock.isOpen());
    lock.close();
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.open(11,20,30): "+lock.open(11,20,30));
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.changeComb(11, 20, 30, 11, 22, 33): "
              + lock.changeComb(11, 20, 30, 11, 22, 33));
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.open(11,22,33): "+lock.open(11,22,33));
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.changeComb(10, 20, 30, 15, 25, 35): "
              + lock.changeComb(10, 20, 30, 15, 25, 35));
    System.out.println("lock.isOpen(): " + lock.isOpen());
    System.out.println("lock.open(15,25,35): "+lock.open(15,25,35));
    System.out.println("lock.isOpen(): " + lock.isOpen());
  }
}
```

1.7

```
import java.util.*;
public class Student extends Person {  // Student inherits Person
  private String country;        // Student aggregates String
  private int credits;
  private double gpa;
  private final Transcript transcript = new Transcript();
```

```
  public Student(String name, boolean male, int yob, String c) {
    super(name, male, yob);
    this.country = c;
  }
  public void updateTranscript(Section section, Grade grade) {
    transcript.add(section, grade);
  }
  public void printTranscript() {
    System.out.println(transcript);
  }
  private class Transcript {       // composition
    Map map = new HashMap();
    void add(Section section, Grade grade) {
      map.put(section, grade);
    }
    public String toString() {
      return map.toString();
    }
  }
}

package chap01.prob07;
public class TestStudent {
  public static void main(String[] args) {
    Student joe = new Student("Joe", true, 1983, "IT");
    joe.updateTranscript(new Section("CS211.02"), new Grade("A-"));
    joe.updateTranscript(new Section("EC110.07"), new Grade("B+"));
    joe.printTranscript();
  }
}
```

1.8

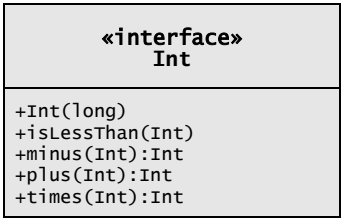
```
class Phone {
  private String areaCode, number;
  public Phone(String areaCode, String number) {
    this.areaCode = areaCode;
    this.number = number;
  }
  public Phone(Phone that) {
    this.areaCode = that.areaCode;
    this.number = that.number;
  }
  public void setAreaCode(String areaCode) {
    this.areaCode = areaCode;
```

```
  }
  public String toString() {
    return "(" + areaCode + ")" + number.substring(0, 3)
         + "-" + number.substring(3);
  }
}

public class Person {
  private final boolean male;
  private final String name;
  private final Phone phone;
  private final int yob;
  public Person(String name, boolean male, int yob, Phone phone) {
    this.name = name;
    this.male = male;
    this.yob = yob;
    this.phone = new Phone(phone);
  }
  public String getName() {
    return name;
  }
  public Phone getPhone() {
    return phone;
  }
  public int getYob() {
    return yob;
  }
  public boolean isMale() {
    return male;
  }
  public String toString() {
    return (male?"Mr. ":"Ms. ") + name+" ("+yob+"), tel. "+phone;
  }
    public static void main(String[] args) {
        Phone tel = new Phone("808", "4561414");
        Person gwb = new Person("G. W. Bush", true, 1946, tel);
        System.out.println(gwb);
        tel.setAreaCode("202");
        System.out.println(gwb);
    }
}
```

Chapter 2
Abstract Data Types

Exercises

2.1 ***ADT: Int***



An ***Int*** is an immutable object that represents a whole number, such as 44 or –7.

Int(*long n*)
Constructs: an *Integer* that is equivalent to *n*.

boolean ***isLessThan***(*Integer x*)
Returns: *true* if this integer is less than *x*; otherwise *false*.

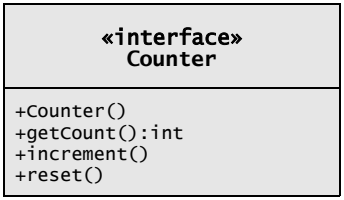
Integer ***minus***(*Integer x*)
Returns: an *Integer* that represents the difference of this integer minus *x*.

Integer ***plus***(*Integer x*)
Returns: an *Integer* that represents the sum of this integer and *x*.

Integer ***times***(*Integer x*)
Returns: an *Integer* that represents the product of this integer and *x*.

```
package chap02.exer01;
public interface Integer {
  public Integer plus(Integer x);
  public Integer minus(Integer x);
  public Integer times(Integer x);
}
```

2.2 *ADT: Counter*



A ***Counter*** is an object that keeps a non-negative integer count.

Counter()
Constructs:a *Counter* that is initialized to 0.

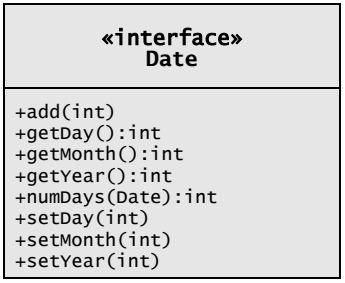
integer ***getCount***()
Postcondition:this date is unchanged.
Returns:the count.

void ***increment***()
Postcondition:the count has been increased by 1.

void ***reset***()
Postcondition:the count is 0.

```
package chap02.exer02;
public interface Counter {
  public void add(Number x);
  public void subtract(Number x);
  public void multiply(Number x);
  public void divide(Number x);
  public Number sum(Number x);
  public Number difference(Number x);
  public Number product(Number x);
  public Number quotient(Number x);
}
```

2.3 *ADT: Date*



A ***Date*** represents a calendar date, such as July 4, 1776.

void ***add***(*integer* days)
Postcondition: this date is advanced by the specified number of days.

integer ***getDay***()
Returns:the day number of this date.
Postcondition:this date is unchanged.

integer ***getMonth***()
Returns:the month number of this date.
Postcondition:this date is unchanged.

integer ***getYear***()
Returns:the year number of this date.
Postcondition:this date is unchanged.

integer ***numDays***(*Date* date)
Postcondition: this date is unchanged.
Returns: the day number days from of this date to the specified date.

void ***setDay***(*integer day*)
Precondition: $1 \le day \le 31$.
Postcondition:this date's day equals the specified *day*.

void ***setMonth***(*integer month*)
Precondition: $1 \le month \le 12$.
Postcondition:this date's month equals the specified *month*.

void ***setYear***(*integer year*)
Precondition: $1700 \le year \le 2100$.
Postcondition:this date's day equals the specified *year*.

```
public interface Date {
  public void add(Number x);
  public void subtract(Number x);
  public void multiply(Number x);
  public void divide(Number x);
  public Number sum(Number x);
  public Number difference(Number x);
  public Number product(Number x);
  public Number quotient(Number x);
}
```

2.4 Algorithm Leap Year Determination

Input: a ***Date*** *date*;
Output: true, if *date* is a leap year; otherwise, *false*.
Postcondition: the ***date*** object is unchanged.

1. Let ***y*** = ***date.getYear***().
2. If ***y*** is divisible by 400, return *true*.
3. If ***y*** is divisible by 100, return *false*.
4. If ***y*** is divisible by 4, return *true*.
5. Return *false*.

2.5 Algorithm Bag Subset

Input: Two *Bag* objects: *bag1* and *bag2*.
Output: *true*, if element of *bag1* is also an element of *bag2*; otherwise, *false*.

1. Let *x* = *bag1.getFirst*().
2. Repeat steps 3-5:
3. If *x* is *null*, return *true*.
4. If *bag2.contains*(*x*) is *false*, return *false*.
5. Let *x* = *bag1.getNext*().

2.6 **Algorithm Bag Intersection**

Input: Two *Bag* objects: *bag1* and *bag2*.
Output: the number of elements that are in both bags.

1. Let $n = 0$.
2. Let x = *bag1.getFirst*().
3. Repeat steps 4-6:
4. If x is *null*, return n.
5. If *bag2.contains*(x), add 1 to n.
6. Let x = *bag1.getNext*().

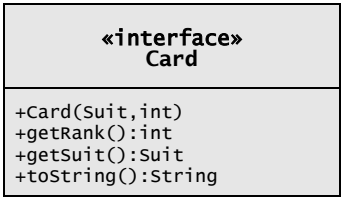
2.7 **Algorithm Bag Remove All**

Input: Two *Bag* objects: *bag1* and *bag2*.
Postcondition: every element of *bag1* is also an element of *bag2*.

1. Let x = *bag1.getFirst*().
2. Repeat steps 3-5:
3. If x is *null*, return.
4. If *bag2.contains*(x) is *false*, *bag1.remove*(x).
5. Let x = *bag1.getNext*().

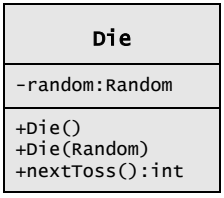
2.8

```
public interface Card {
  public int getRank();
  public Suit getSuit();
  public String toString();
}

final class Suit {
  public static final Suit SPADES = new Suit(0);
  public static final Suit HEARTS = new Suit(1);
  public static final Suit DIAMONDS = new Suit(2);
  public static final Suit CLUBS = new Suit(3);
  private int n;
  private Suit(int n) {
    this.n = n;
  }
}
```



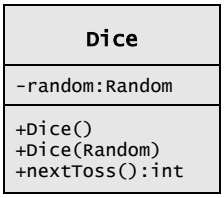
2.9

```
public interface Die {
  public int nextToss();
}
```



2.10

```
public interface Dice {
  public int nextToss()
}
```



2.11 **a.** ***ADT: Number***

A ***Number*** object represents a real number.
void ***add***(*Number* x)
Postcondition: x has been added to this number.
void ***subtract***(*Number* x)
Postcondition: x has been subtracted from this number.
void ***multiply***(*Number* x)
Postcondition: this number has been multiplied by x.
void ***divide***(*Number* x)
Postcondition: this number has been divided by x.
Number ***sum***(*Number* x)
Postcondition: this number is unchanged.
Returns: the sum of x and this number.
Number ***difference***(*Number* x)
Postcondition: this number is unchanged.
Returns: the difference of x from this number.
Number ***product***(*Number* x)
Postcondition: this number is unchanged.
Returns: the sum of x and this number.
Number ***quotient***(*Number* x)
Postcondition: this number is unchanged.
Returns: the quotient of this number divided by x.

b.

```
public interface Number {
  public void add(Number x);
  public void subtract(Number x);
  public void multiply(Number x);
  public void divide(Number x);
  public Number sum(Number x);
  public Number difference(Number x);
  public Number product(Number x);
  public Number quotient(Number x);
}
```

2.12 **a.** ***ADT: Point***

A ***Point*** object represents a point (x, y) in two-dimensional coordinate space.
double ***getX***()
Postcondition: this point is unchanged.
Returns: the *x*-coordinate of this point.
double ***getY***()
Postcondition: this point is unchanged.
Returns: the *y*-coordinate of this point.
void ***setX***(*double* x)
Postcondition: the *x*-coordinate of this point equals x.
void ***setY***(*double* y)
Postcondition: the *y*-coordinate of this point equals y.
void ***moveTo***(*double* x, *double* y)
Postconditions: the *x*-coordinate of this point equals x;
the *y*-coordinate of this point equals y.

b.
```
public interface Point {
  public double getX();
  public double getY();
  public void setX(double x);
  public void setY(double y);
  public void moveTo(double x, double y);
}
```

2.13 **a.** ***ADT: Vector***

A ***Vector*** is an ordered container of objects.
Object ***getAtIndex***(*integer* i)
Postcondition: this vector is unchanged.
Returns: the object at index *i*.
void ***setAtIndex***(*Object* x, *integer* i)
Postcondition: the object *x* is at index *i*.
integer ***size***()
Postcondition: this vector is unchanged.
Returns: the number of objects in this vector.

b.
```
public interface Vector {
public interface Vector {
  public Object getAtIndex(int i);
  public void setAtIndex(Object x, int i);
  public integer size();
}
```

2.14 **a.** ***ADT: Matrix***

A ***Matrix*** is a two-dimensional array of numbers.
int ***getRows***()
Postcondition: this matrix is unchanged.
Returns: the number of rows in this matrix.
int ***getColumns***()
Postcondition: this matrix is unchanged.
Returns: the number of columns in this matrix.
double ***get***(*integer* i, *integer* j)
Postcondition: this matrix is unchanged.
Returns: the number in row i and column j.
void ***set***(*integer* i, *integer* j, *double* x)
Postcondition: the number in row i and column j equals x.
Matrix ***transpose***()
Postcondition: this matrix is unchanged.
Returns: a new matrix that is the transpose of this matrix.

b.
```
public interface Matrix {
public interface Matrix {
  public int getRows();
  public int getColumns();
  public double getY(int i, int j);
  public void set(int i, int j, double x);
  public Matrix transpose();
}
```

2.15 **a.** ***ADT: Polynomial***

A ***Polynomial*** is a mathematical function that is determined by its sequence of coefficients and exponents.
Polynomial ***derivative***()
Postcondition: this polynomial is unchanged.
Returns: a new polynomial that is the derivative of this polynomial.
integer ***getDegree***()
Postcondition: this polynomial is unchanged.
Returns: the highest exponent of this polynomial.
String ***toString***()
Postcondition: this polynomial is unchanged.
Returns: a string representation of this polynomial.
double ***valueAt***(*double* x)
Postcondition: this polynomial is unchanged.
Returns: the *y*-value of this polynomial evaluated at x.

b.
```
public interface Polynomial {
  public Polynomial derivative();
  public int getDegree();
  public String toString();
  public double valueAt(double x);
}
```

2.16 **a.** ***ADT: Purse***

A ***Purse*** is a container of coins.
integer ***getNumCoinsOf***(*integer* c)
Postcondition: this purse is unchanged.
Returns: the number of c-cent coins in this purse.
void ***addNumCoinsOf***(*integer* c, *integer* n)
Postcondition: increases the number of c-cent coins in this purse by n.
integer ***totalNumCoins***()
Postcondition: this purse is unchanged.
Returns: the total number of coins in this purse.
double ***totalValue***()
Postcondition: this polynomial is unchanged.
Returns: the total dollar value of all the coins in this purse.

b.
```
public interface Purse {
  public int getNumCoinsOf(int c);
  public void addNumCoinsOf(int c, int n);
  public int totalNumCoins();
  public double totalValue();
}
```

2.17 **a.** ***ADT: Set***

A ***Set*** is a container of unique objects (no duplicates).

boolean ***add***(*Object* element)
Postcondition: the specified element is in this set.
Returns: *true* if and only if this set is changed.

void ***clear***()
Postcondition: this set is empty.

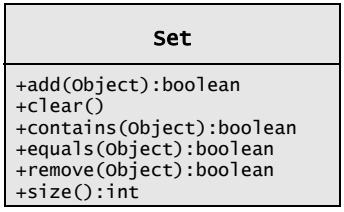
boolean ***contains***(*Object* element)
Postcondition: this set is unchanged.
Returns: *true* if and only if this set contains the specified element.

boolean ***equals***(*Object* object)
Postcondition: this set is unchanged.
Returns: *true* if and only if the specified object is a set and it has the same contents as this set.

boolean ***remove***(*Object* element)
Postcondition: the specified element is not in this set.
Returns: *true* if and only if this set is changed.

integer ***size***()
Postcondition: this set is unchanged.
Returns: the number of elements in this set.

b.



2.18 **a.** ***ADT: Map***

A ***Map*** is a container of key-value pairs, where the keys are unique.

void ***clear***()
Postcondition: this map is empty.

boolean ***containsKey***(*Object* key)
Postcondition: this map is unchanged.
Returns: *true* if and only if this map contains the specified key.

Object ***get***(*Object* key)
Postcondition: this map is unchanged.
Returns: the value for the specified key if it is in this map, otherwise *null* is returned.

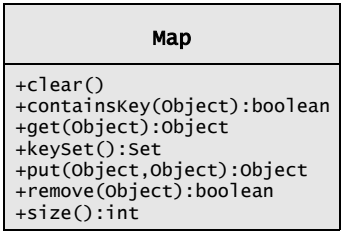
Set ***keySet***()
Postcondition: this map is unchanged.
Returns: a new set whose elements are the keys in this map.

Object ***put***(*Object* key, *Object* value)
Postcondition: the specified key-value pair is in this map.
Returns: the previous value stored in this map for the specified key if that key is in this map; otherwise *null* is returned.

boolean ***remove***(*Object* key)
Postcondition: no key-value pair with the specified key is in this map.
Returns: *true* if and only if this map is changed.

integer ***size***()
Postcondition: this map is unchanged.
Returns: the number of elements in this map.

b.



2.19 **a.** ***ADT: List***

A ***List*** is a sequence of objects.

void ***add***(*integer* index, *Object* element)
Postcondition: the specified element is at the specified index in this list.

void ***clear***()
Postcondition: this list is empty.

boolean ***contains***(*Object* element)
Postcondition: this list is unchanged.
Returns: *true* if and only if this list contains the specified element.

Object ***get***(*integer* index)
Postcondition: this list is unchanged.
Returns: the element at the specified index is returned.

integer ***indexOf***(*Object* element)
Postcondition: this list is unchanged.
Returns: the index of the specified element is returned if it is in this list; otherwise –1 is returned.

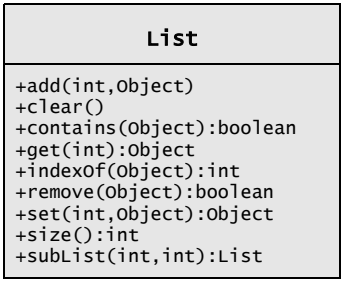
boolean ***remove***(*Object* element)
Postcondition: the first occurrence of the specified element is removed from this list if it is present.
Returns: *true* if and only if this list is changed.

Object ***set***(*integer* index, *Object* element)
Postcondition: the element at the specified index in this list is replaced by the specified element.
Returns: the replaced element.

integer ***size***()
Postcondition: this list is unchanged.
Returns: the number of elements in this list.

List ***subList***(*integer* fromIndex, *integer* toIndex)
Postcondition: this list is unchanged.
Returns: a new list whose elements are the element in this list that are in the specified index range.

b.



Programming Problems

2.1

```
public interface Set {
  public boolean add(Object object);
  public boolean contains(Object object);
  public Object getFirst();
  public Object getNext();
  public boolean remove(Object object);
  public int size()
}
```

2.2

```
public class ArraySet implements Set {
  private Object[] objects = new Object[1000];
  private int size, i;
  public boolean add(Object object) {
    if (contains(object)) return false;  // no duplicates
    objects[size++] = object;
    return true;
  }
  public boolean contains(Object object) {
    for (int i=0; i<size; i++)
      if (objects[i]==object) return true;
    return false;
  }
  public Object getFirst() {
    i = 0;
    return objects[i++];
  }
  public Object getNext() {
    return objects[i++];
  }
  public boolean remove(Object object) {
    for (int i=0; i<size; i++)
      if (objects[i]==object) {
        System.arraycopy(objects, i+1, objects, i, size-i-1);
        objects[--size] = null;
        return true;
      }
    return false;
  }
  public int size() {
    return size;
  }
```

```
}
```

2.3

```
public class TestArraySet {
   public static void main(String[] args) {
   Set set = new ArraySet();
   System.out.println("set.add(\"CA\"):\t" + set.add("CA"));
   System.out.println("set.add(\"US\"):\t" + set.add("US"));
   System.out.println("set.add(\"MX\"):\t" + set.add("MX"));
   System.out.println("set.add(\"CA\"):\t" + set.add("CA"));
   System.out.println("set.add(\"US\"):\t" + set.add("US"));
   System.out.println("set.size():\t" + set.size());
   print(set);
   System.out.println("set.contains(\"CA\"):\t"+set.contains("CA"));
   System.out.println("set.remove(\"CA\"):\t" + set.remove("CA"));
   System.out.println("set.size(): " + set.size());
   print(set);
   System.out.println("set.contains(\"CA\"):\t"+set.contains("CA"));
   System.out.println("set.remove(\"CA\"):\t"+set.remove("CA"));
   System.out.println("set.size(): " + set.size());
   print(set);
   System.out.println("set.contains(\"MX\"):\t"+set.contains("MX"));
   System.out.println("set.contains(\"AR\"):\t"+set.contains("AR"));
 }
  public static void print(Set set) {
    System.out.print(set.getFirst());
    for (int i=1; i<set.size(); i++)
      System.out.print(", " + set.getNext());
    System.out.println();
  }
}
```

2.4

```
public class TestArraySet {
  public static void main(String[] args) {
    Date bang = new IntsDate(1776, 07, 04);
    System.out.println(bang.getMonth() + "/" + bang.getDay() +"/"
                       + bang.getYear());
    bang.setYear(2004);
    System.out.println(bang.getMonth() + "/" + bang.getDay() +"/"
                       + bang.getYear());
    bang.setYear(-4);
    System.out.println(bang.getMonth() + "/" + bang.getDay() +"/"
                       + bang.getYear());
  }
}
```

2.5

```
public class IntsDate implements Date {
    private int day, month, year;
    public IntsDate(int year, int month, int day) {
    setDay(day);
    setMonth(month);
    setYear(year);
  }
  public int getDay()  {
    return day;
  }
  public int getMonth()  {
    return month;
  }
  public int getYear()  {
    return year;
  }
  public void setDay(int day) {
    if (day<1 || day>31) throw new IllegalArgumentException();
    if ((month == 4 || month == 6 || month == 9 || month == 11)
      && (day > 30)) throw new IllegalArgumentException();
    if (month == 2 && (day > 29 || !isLeap(year) && day > 28))
      throw new IllegalArgumentException();
    this.day = day;
  }
  public void setMonth(int month) {
    if (month<1 || month>12) throw new IllegalArgumentException();
    this.month = month;
  }
  public void setYear(int year) {
    if (year<1700 || year>2100) throw new IllegalArgumentException();
    this.year = year;
  }
  private boolean isLeap(int year) {
    if (year%400 == 0) return true;
    if (year%100 == 0) return false;
    if (year%4 == 0) return true;
    return false;
  }
}
```

2.6

```
public class ComboLock {
  private int n1, n2, n3;
  private boolean open;
```

```
  public final long id;
  private static long nextId=1000;
  { id = nextId++; }  // initialization block
  public ComboLock(int n1, int n2, int n3) {
    this.n1 = n1;
    this.n2 = n2;
    this.n3 = n3;
  }
  public boolean changeComb(int n1, int n2, int n3, int n4,
                            int n5, int n6) {
    if (this.n1 != n1 || this.n2 != n2 || this.n3 != n3) return false;
    this.n1 = n4;
    this.n2 = n5;
    this.n3 = n6;
    open = false;
    return true;
  }
  public void close() {
    open = false;
  }
  public boolean isOpen() {
    return open;
  }
  public boolean open(int n1, int n2, int n3) {
    if (this.n1 == n1 && this.n2 == n2 && this.n3 == n3) open = true;
    else open = false;
    return open;
  }
}
public class TestComboLock {
 public static void main(String[] args) {
  ComboLock lock0 = new ComboLock(10, 20, 30);
  System.out.println("lock0.id): " + lock0.id);
  ComboLock lock1 = new ComboLock(11, 21, 31);
  System.out.println("lock1.id): " + lock1.id);
  ComboLock lock2 = new ComboLock(12, 22, 32);
  System.out.println("lock2.id): " + lock2.id);
  System.out.println("lock0.open(11,21,31): "+lock0.open(11,21,31));
  System.out.println("lock1.open(11,21,31): "+lock1.open(11,21,31));
  System.out.println("lock2.open(11,21,31): "+lock2.open(11,21,31));
 }
}
```

Chapter 3
Arrays

Exercises

3.1 **Algorithm Minimum Element**

Input: a sequence $\{a_0, a_1, a_2, ..., a_{n-1}\}$ of comparable elements.

Output: an index value m.

Postconditions: $a_m \le a_i$, for all i.

1. Let $m = 0$.
2. Repeat Step 3 for $i = 1$ to $n - 1$.
3. If $a_i < a_m$, set $m = i$.
4. Return m.

3.2 After step 1, $m = 0$.

After the first iteration of the loop in steps 2-3, m is either 0 or 1; if $a_1 > a_0$, then m got reset to 1; otherwise, $a_1 \le a_0$ and m is still 0. In either case, $a_m \ge a_0$ and $a_m \ge a_1$; *i.e.*, $a_m = \max\{a_0, a_1\}$.

After the second iteration of the loop in steps 2-3, m is either 0, 1, or 2. If $a_2 > a_m$, then m got reset to 2; otherwise, $a_2 \le a_m$ and m is still 0 or 1. In the first case, $a_2 > a_m$ and $a_m \ge a_0$ and $a_m \ge a_1$, so $a_2 \ge a_0$ and $a_2 \ge a_1$, so $a_2 = \max\{a_0, a_1, a_2\}$; thus setting $m = 2$ makes $a_m = \max\{a_0, a_1, a_2\}$. In the second case, $a_2 \le a_m$ and $a_m \ge a_0$ and $a_m \ge a_1$, so $a_m = \max\{a_0, a_1, a_2\}$ without resetting m. Thus, in either case, $a_m = \max\{a_0, a_1, a_2\}$.

Before iteration i begins, $a_m = \max\{a_0, a_1, ..., a_{i-1}\}$. On that ith iteration, either $a_i > a_m$ or $a_i \le a_m$. In the first case, $a_i > \max\{a_0, a_1, ..., a_{i-1}\}$, so $a_i = \max\{a_0, a_1, ..., a_i\}$; thus, after m is set to i, $a_m = \max\{a_0, a_1, ..., a_i\}$. In the second case, $a_m \ge a_i$, so $a_m = \max\{a_0, a_1, ..., a_i\}$ without resetting m. Thus, in either case, after the iteration has finished step 3, $a_m = \max\{a_0, a_1, ..., a_i\}$. This is the *loop invariant*.

The argument in the previous paragraph applies to every index i, all the way up to $i = n - 1$. Therefore, after that last iteration, $a_m = \max\{a_0, a_1, ..., a_{n-1}\}$. Thus, the correct value of m is returned at step 4.

3.3 Algorithm Sequential Search

Input: a sequence $\{a_0, a_1, a_2, ..., a_{n-1}\}$ and a target value x.

Output: an index value i.

Postconditions: Either $a_i = x$, or $i = -n$ and each $a_j \neq x$.

If $a_i = x$, then $a_j \neq x$ for all $j > i$.

1. For each i from $n-1$ down to 0, do step 2:
2. If $a_i = x$, return i.
3. Return $-n$.

3.4 If the algorithm returns i at step 2, then ai = x, as required by the first alternative of the postcondition.

If the algorithm returns $-n$ at step 3, then the condition $a_i = x$ at step 2 was never true for any of the values of i in the range 0 to $n-1$. That is the second alternative of the post-condition.

3.5 Algorithm Maximum Element

Input: a sequence $\{a_0, a_1, a_2, ..., a_{n-1}\}$ of comparable elements.

Output: an element a_m.

Postconditions: $a_m \geq a_i$, for all i.

1. Repeat Step 2 for $i = n - 1$ down to 1:
2. If $a_i > a_{i-1}$, swap a_i with a_{i-1}.
3. Return a_0.

3.6 On the first iteration of the loop in steps 1-2, $i = n - 1$. *If* $a_{n-1} \leq a_{n-2}$, then no swap occurs in step 2, and $a_{n-2} = \max\{a_{n-2}, a_{n-1}\}$. But if $a_{n-1} > a_{n-2}$, then the swap occurs, thereby making a_{n-2} the larger. Thus, in either case, after the first iteration has finished, $a_{n-2} = \max\{a_{n-2}, a_{n-1}\}$.

On the second iteration, $i = n - 2$. *If* $a_{n-2} \leq a_{n-3}$, then no swap occurs, and therefore $a_{n-3} \geq a_{n-2} = \max\{a_{n-2}, a_{n-1}\}$. But if $a_{n-1} > a_{n-2}$, then the swap occurs, thereby making a_{n-3} the larger of those two, and thus again $a_{n-3} \geq a_{n-2} = \max\{a_{n-2}, a_{n-1}\}$. Therefore, in either case, after the second iteration, $a_{n-3} = \max\{a_{n-3}, a_{n-2}, a_{n-1}\}$.

Similarly, on the jth iteration, $a_{n-j-1} = \max\{a_{n-j-1}, ..., a_{n-2}, a_{n-1}\}$. This is a *loop invariant* for the algorithm. Note that $j = n - i$, so $i = n - j$ and the loop invariant can be expressed as $a_{i-1} = \max\{a_{i-1}, ..., a_{n-2}, a_{n-1}\}$. Since this is true for each i from $n - 1$ down to 1, it is true when $i = 1$; *i.e.*, $a_0 = \max\{a_0, ..., a_{n-2}, a_{n-1}\}$. Thus the correct value is returned at step 3.

3.7 The assignment `b = a` merely makes `b` an alias for the array named `a`; *i.e.*, there is still only one array.

In order to return a separate array, we must allocate new storage for that array

```
int[] b = new int[a.length];
```

and then copy all elements from `a` into `b`

```
System.arraycopy(a, 0, b, 0, a.length);
```

3.8 The method statements have this effect:

```
int temp = i;  // assignes the value of i to temp
i = j;         // assigns the value of j to i
j = temp;      // assigns the value of temp to j
```

This has no effect upon the array `a[]` (or even on the variables passed to `i` and `j`).

Here is a correct version:

```
public static void swap(double[] a, int i, int j) {
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

3.9 This doesn't work because everythin gets swapped twice, thereby putting it back where it started. The only change required is to run the `for` loop only half-way through the array:

```
for (int i = 0; i < n/2; i++)
  swap(a, i, n-1-i);
```

3.10 The first two output statements merely show that `x` and `y` are arrays and that they are stored at different locations in memory. Consequently, the `boolean` value `eqeq` is `false` because the == operator tests for identity. But the `java.util.Arrays.equals()` method tests for equality of content, so its value is `true`.

The assignment `y = x` changes `y` to an alias for the x array. So then `x` and `y` are identically equal, and thus the last output is `true`.

3.11 The Binary Search algorithm requires the array to be in ascending order. So, if the target is found, all other elements that contain it must be adjacent to the element where it is found. So to find the smallest index among those elements that contain the same value, we can simply decrement the index until its element is no longer equal to the target:

Algorithm Modified Binary Search

Input: a sequence $\{a_0, a_1, a_2, ..., a_{n-1}\}$ and a target value x.

Output: an index value i.

Precondition: The sequence is sorted: $a_0 \le a_1 \le a_2 \le \cdots \le a_{n-1}$.

Postcondition: Either $a_i = x$;

or $i = -p-1$, where $a_j < x$ for all $j < p$, and $a_j > x$ for all $j \ge p$.

1. Let $p = 0$ and $q = n - 1$.
2. Repeat steps 3-5 while $p \le q$:
3. Let $i = (p + q)/2$.
4. If $a_i = x$, then decrement i until $i < 0$ or $a_i \ne x$, and then return $i + 1$.
5. If $a_i < x$, let $p = i + 1$; otherwise let $q = i - 1$.
6. Return $-p - 1$.

3.12 To change the Binary Search to the Interpolation Search, replace step 3 with:

3. Let $i = p + r(q - p)$, where $r = (x - a_p)/(a_q - a_p)$.

3.13 $f \in \Theta(g) \Leftrightarrow f(n)/g(n)$ is bounded and $g(n)/f(n)$ is bounded $\Leftrightarrow f \in O(g)$ and $g \in O(f)$.

3.14 $f \in o(g) \Leftrightarrow f(n)/g(n) \to 0 \Rightarrow f(n)/g(n)$ is bounded and $g(n)/f(n)$ is not bounded
$\Rightarrow f \in O(g)$ and $f \notin \Theta(g) \Leftrightarrow f \in O(g) - \Theta(g)$.

3.15 $f \in \omega(g) \Leftrightarrow g(n)/f(n) \to 0 \Rightarrow g(n)/f(n)$ is bounded and $f(n)/g(n)$ is not bounded
$\Rightarrow f \in \Omega(g)$ and $f \notin \Theta(g) \Leftrightarrow f \in o(g) - \Theta(g)$.

3.16 The functions $f(n)$, $g(n)$, $f(n)/g(n)$, and $g(n)/f(n)$ have the values shon in this table:

n	$f(n)$	$g(n)$	$f(n)/g(n)$	$g(n)/f(n)$
1	1	1	1	1
2	1	1	1	1
3	1	2	1/2	2
4	2	2	1	1
5	2	8	1/4	4
6	8	8	1	1
7	8	64	1/8	8
8	64	64	1	1
9	64	1024	1/16	16

When n is even, both ratios equal 1; thus, $f(n)/g(n)$ does not approach 0, and so $f \notin o(g)$.
When n is odd, $g(n)/f(n)$ grows exponentially; thus, $g(n)/f(n)$ is not bounded, so $f \notin \Theta(g)$.
But $f(n)/g(n)$ never exceeds 1; thus $f(n)/g(n)$ is bounded, and so $f \in O(g)$.
Thus, $f \in O(g) - \Theta(g) - o(g)$.

3.17 $f \in O(g) \Leftrightarrow f(n)/g(n)$ is bounded $\Leftrightarrow g \in \Omega(f)$.

3.18 $f \in o(g) \Leftrightarrow f(n)/g(n) \to 0 \Leftrightarrow g \in \omega(f)$.

Programming Problems

3.1

```
public class Main {
  private java.util.Random random = new java.util.Random();

  public Main(int n) {
    double[] a = randomDoubles(n);
    double[] aa = (double[])a.clone();
    print(a);
    print(aa);
    a[0] = a[1] = a[2] = 0.888;
    print(a);
    print(aa);
  }
```

```
  private double[] randomDoubles(int n) {
    double[] a = new double[n];
    for (int i=0; i<n; i++)
      a[i] = random.nextDouble();
    return a;
  }

  private void print(double[] a) {
    printf(a[0], "{#.###");
    for (int i=1; i<a.length; i++)
      printf(a[i], "','#.###");
    System.out.println("}");
  }

  private void printf(double x, String fs) {
    System.out.print(new java.text.DecimalFormat(fs).format(x));
  }

  public static void main(String[] args) {
    new Main(Integer.parseInt(args[0]));
  }
}
```

3.2

```
public class Main {
  boolean[] z = new boolean[8];
  char[] c = new char[8];
  byte[] b = new byte[8];
  short[] s = new short[8];
  long[] j = new long[8];
  float[] f = new float[8];
  double[] d = new double[8];
  Object[] x = new Object[8];

  public Main() {
    int[] a = { 33,44,55,66};
    try {a[8]=0;} catch(Exception e){}
    System.out.println("z:\t" + z);
    System.out.println("c:\t" + c);
    System.out.println("b:\t" + b);
    System.out.println("s:\t" + s);
    System.out.println("j:\t" + j);
    System.out.println("f:\t" + f);
    System.out.println("d:\t" + d);
```

```
    System.out.println("x:\t" + x);
  }

  public static void main(String[] args) {
    new Main();
  }
}
```

3.3

```
private String letters(int n) {
  // Returns: an object s of type String;
  // Postconditions: s.length() == n;
  //                 each character of s is a calital letter;
  StringBuffer buf = new StringBuffer(n);
  for (int i=0; i<n; i++) {
    int j = random.nextInt(26);
    char ch = (char)('A'+j);
    buf.append(ch);
  }
  return buf.toString();
}
```

3.4

```
private int[] permutation(int n) {
  // Returns: an array a[] of type int[];
  // Postconditions: a.length == n;
  //                 0 <= a[i] < n, for all i;
  //                 all n elements of a[] are different;
  int[] a = new int[n];
  for (int i=0; i<n; i++)
    a[i] = i;
  for (int i=0; i<n; i++)
    swap(a,i,random.nextInt(n));
  return a;
}
```

3.5

```
int search(int[] a, int target) {
  for (int i = a.length - 1; i >= 0; i--)
    if (a[i]==target) return i;
  return -a.length;
}
```

3.6

```
public class TestArrays {
  public static void main(String[] args) {
    int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
    IntArrays.print(a);
    System.out.println("IntArrays.isSorted(a): "
```

```
      + IntArrays.isSorted(a));
    a = IntArrays.randomInts(8, 100);
    IntArrays.print(a);
    System.out.println("IntArrays.isSorted(a): "
      + IntArrays.isSorted(a));
    a = IntArrays.resize(a, 12);
    IntArrays.print(a);
    IntArrays.swap(a, 2, 6);
    IntArrays.print(a);
  }
}
```

3.7

```
public static int[] truncate(int[] a, int n) {
  // returns a new array with the first n elements of a[];
  if (n > a.length) throw new IllegalArgumentException();
  int[] aa = new int[n];
  System.arraycopy(a, 0, aa, 0, n);
  return aa;
}
```

3.8

```
public static int[] reverse(int[] a) {
  // returns a new array with the elements of a[] in reverse order;
  int[] aa = new int[n];
  int n = a.length;
  int[] aa = new int[n];
  for (int i=0; i<n; i++)
    aa[i] = a[n-i-1];
  return aa;
}
```

3.9

```
public static int[] uniqueRandomInts(int n, int range) {
  // returns a new array of length n with unique nonnegative random
  // integers that are less than the specified range;
  if (n > range) throw new IllegalArgumentException();
  int[] a = new int[n];
  for (int i=0; i<n; i++) {  // insert a unique integer:
    next:
      for (;;) {  // repeat until another unique one is found
        int x = random.nextInt(range);
        for (int j=0; j<i; j++) {
          if (a[j] == x)   // a[i] is not unique
            continue next;  // try again
        }
        a[i] = x;  // insert it
```

```
          break;  // a[i] is unique among {a[0]..a[i-1]}
        }
    }
    return a;
  }
```

3.10

```
  public static int[] permutation(int n) {
    // returns a new array of length n whose elements are the n
    // nonnegative integers that are less than n, in random order;
    int[] a = new int[n];
    for (int i=0; i<n; i++)
      a[i] = i;
    for (int i=0; i<n; i++)
      swap(a, i, random.nextInt(n));
    return a;
  }
```

3.11

```
  /**
   * Determines whether the specified array is in descending order.
   *
   * @param  a the array.
   * @return true if a[] is in descending order, otherwise false.
   */
  public static boolean isDescending(int[] a) {
    for (int i=1; i<a.length; i++)
      if (a[i] > a[i-1]) return false;
    return true;
  }

  /**
   * Determines whether the specified array is sorted, either
   * in ascending order or in descending order.
   *
   * @param  a the array.
   * @return true if a[] is sorted, otherwise false.
   */
  public static boolean isSorted(int[] a) {
    return isAscending(a) || isDescending(a);
  }
```

3.12

```
  public class TestBinarySearch {
    public static void main(String[] args) {
      int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
      int k = search(a, 50);
```

```
    System.out.println("search(a," + 50 + "): " + k);
    int i = -k-1;
    System.out.println("i: " + i);
    System.out.println("a[" + (i-1) + "]: " + a[i-1]);
    System.out.println("a[" + i + "]: " + a[i]);
  }

  static int search(int[] a, int x) {
    int p = 0, q = a.length-1;
    while (p <= q) {           // search the segment a[p..q]
      int i = (p+q)/2;         // index of element in the middle
      if (a[i] == x) return i;
      if (a[i] < x) p = i+1;   // search upper half
      else q = i-1;            // search lower half
    }
    return -p-1;               // not found
  }
}

/*  Output:
search(a,50): -4
i: 3
a[2]: 44
a[3]: 55
*/
}
```

3.13

```
void bubble(int[] a) {
  for (int i=1; i<a.length; i++)
    if (a[i] < a[i-1]) IntArrays.swap(a, i-1, i);
}
```

3.14

```
void select(int[] a) {
  int m=0;
  for (int i=1; i<a.length; i++)
    if (a[i] > a[m]) m = i;
  IntArrays.swap(a, m, a.length-1);
}
```

3.15

```
void insert(int[] a) {
  int i = a.length - 1;
  int ai = a[i], j = i;
  for (j=i; j>0 && a[j-1]>ai; j--)
    a[j] =a[j-1];
```

```
    a[j] = ai;
  }
```

3.16

```
  void merge(int[] a) {
    int n=a.length, m=n/2;
    int i=0, j=m, k=0;
    int[] aa = new int[n];
    while (i < m && j < n)
      if (a[i] < a[j]) aa[k++] = a[i++];
      else aa[k++] = a[j++];
    if (i < m) System.arraycopy(a, i, a, k, m-i);
    System.arraycopy(aa, 0, a, 0, k);
  }
```

3.17

```
  int frequency(int[] a, int value) {
    int f = 0;
    for (int i=0; i<a.length; i++)
      if (a[i] == value) ++f;
    return f;
  }
```

3.18

```
  int numRuns(int[] a, int length) {
    int num = 0;
    int value = a[0], lengthThisRun = 1;
    for (int i=1; i<a.length; i++) {
      if (a[i] == value) ++lengthThisRun;
      else {
        value = a[i];
        if (lengthThisRun == length) ++num;
        lengthThisRun = 1;
      }
    }
    return num;
  }
```

3.19

```
  int partition(int[] a) {
    int a0 = a[0], i = 0, j = a.length;
    while (i < j) {
      while (i < j && a[--j] >= a0)
        ;
      if (i < j) a[i] = a[j];
      while (i < j && a[++i] <= a0)
        ;
      if (i < j) a[j] = a[i];
    }
```

```
    a[j] = a0;
    return j;
  }
```

3.20

```
  public class Main {
    public static void main(String[] args) {
      int len = 1, count = 0;
      while (count<args.length) {
        for (int j=0; j<args.length; j++) {
          if (args[j].length() == len) {
            System.out.println(args[j]);
            ++count;
          }
        }
        ++len;
      }
    }
  }
```

3.21

```
  public void copyInto(Object[] a) {
    // Replaces the first n elements of a[] with the n elements of
    // this vector, where n is the size of this vector.
    if (size > a.length) throw new IllegalArgumentException();
    for (int i=0; i<size; i++)
      a[i] = objects[i];
  }
```

3.22

```
  public void trimToSize() {
    // Resizes the backing array this vector, making its length
    // equal to the size of this vector.
    Object[] temp = new Object[size];
    System.arraycopy(objects, 0, temp, 0, size);
    objects = temp;
  }
```

3.23

```
  public void ensureCapacity(int n) {
    // Resizes the backing array this vector, making its length
    // equal to n.
    Object[] temp = new Object[n];
    System.arraycopy(objects, 0, temp, 0, size);
    objects = temp;
  }
```

3.24

```
  public int indexOf(Object x) {
    // Returns the smallest index of any component of this vector
    // that equals(x). If not found, -1 is returned.
```

```
    for (int i=0; i<size; i++)
      if (objects[i].equals(x)) return i;
    return -1;
  }
```

3.25

```
  public boolean contains(Object x) {
    // Returns true if and only if some component of this vector
    // equals(x).
    for (int i=0; i<size; i++)
      if (objects[i] == x) return true;
    return false;
  }
```

3.26

a.

```
  public Object elementAt(int i) {
    // Returns the element at index i in this vector.
    return objects[i];
  }
```

b.

```
  public Object firstElement() {
    // Returns the first element in this vector.
    return objects[0];
  }
```

c.

```
  public Object lastElement() {
    // Returns the last element in this vector.
    return objects[size-1];
  }
```

d.

```
  public void setElementAt(Object x, int i) {
    // Replaces the element at index i with x.
    objects[i] = x;
  }
```

3.27

a.

```
  public Object get(int i) {
    // Returns the element at index i in this vector.
    return objects[i];
  }
```

b.

```
  public void set(int i, Object x) {
    // Replaces the element at index i with x.
    objects[i] = x;
  }
```

c.

```
  public boolean add(Object x) {
    // Appends x to the end of this vector and returns true.
    if (size == objects.length) resize();
    objects[size++] = x;
    return true;
```

```
  }
```

3.28

```
public boolean remove(Object x) {
  // Deletes the first occurrence of x from this vector, shifting all
  // the elements that follow it back one position. Returns true if
  // x is found and removed; otherwise returns false.
  int i = indexOf(x);
  if (i < 0) return false;
  --size;
  System.arraycopy(objects, i+1, objects, i, size-i);
  objects[size] = null;
  return true;
}
```

3.29

```
public void add(int i, Object x) {
  // Inserts x at index i, shifting all the elements from that
  // position to the end forward one position.
  if (i<0 || i>size) throw new IndexOutOfBoundsException();
  System.arraycopy(objects, i, objects, i+1, size-i);
  objects[i] = x;
  ++size;
}
```

3.30

```
public Object remove(int i) {
  // Removes the element at index i, shifting all the elements
  // that follow it back one position. Returns the removed object.
  if (i<0 || i>=size) throw new IndexOutOfBoundsException();
  Object x = objects[i];
  --size;
  System.arraycopy(objects, i+1, objects, i, size-i);
  objects[size] = null;
  return x;
}
```

3.31

```
public boolean isHomogeneous(Object[] a) {
  // Returns true iff every non-null element has the same type.
  int n=a.length;
  if (n<2) return true;
  if (a[0] == null) {
    for (int i=1; i<n; i++) {
      if (a[i] != null)
        return false;
    }
  } else {
    Class c = a[0].getClass();
```

```
    for (int i=1; i<n; i++) {
      if (a[i] == null || !a[i].getClass().equals(c))
        return false;
    }
  }
  return true;
}
```

3.32

```
public void print(int[][] a) {
  // Prints the two-dimensional array a[][], one row per line.
  int rows=a.length, cols=a[0].length;
  for (int i=0; i<rows; i++) {
    for (int j=0; j<cols; j++) {
      System.out.print(a[i][j] + " ");
    }
    System.out.println();
  }
}
```

3.33

```
public class PascalsTriangle {
  public static void main(String[] args) {
    int rows = Integer.parseInt(args[0]);
    int[][] a = init(rows);
    print(a);
  }

  static int[][] init(int m) {
    int[][] a = new int[m][];
    for (int i = 0; i < m; i++)
      a[i] = new int[i+1];
    for (int i = 0; i < m; i++)
      for (int j = 0; j <= i; j++)
        if (j == 0 || j == i) a[i][j] = 1;
        else a[i][j] = a[i-1][j-1] + a[i-1][j];
    return a;
  }

  static void print(int[][] a) {
    for (int i = 0; i < a.length; i++) {
      for (int j = 0; j <= i; j++)
        print(a[i][j],5);
      System.out.println();
    }
```

```
  }

  static void print(int n, int w) {
    // prints n right-justified in a field on width w:
    String s = "" + n, blanks = "                    ";
    int len = s.length();
    System.out.print(blanks.substring(0, w-len) + s);
  }
}
```

3.34

```
public class PascalsTriangle {
  public static void main(String[] args) {
    int rows = Integer.parseInt(args[0]);
    int[][] a = init(rows);
    print(a);
  }

  static int fact(int n) {
    if (n<2) return 1;
    int f=n;
    for (int i=2; i<n; i++)
      f *= i;
    return f;
  }

  static int[][] init(int n) {
    int[][] a = new int[n][n];
    for (int i = 0; i < n; i++)
      for (int j = 0; j <= i; j++)
        if (j == 0 || j == i) a[i][j] = 1;
        else a[i][j] = fact(i)/(fact(j)*fact(i-j));
    return a;
  }

  static void print(int[][] a) {
    for (int i = 0; i < a.length; i++) {
      for (int j = 0; j <= i; j++)
        print(a[i][j],5);
      System.out.println();
    }
  }
```

```
  static void print(int n, int w) {
    // prints n right-justified in a field on width w:
    String s = "" + n, blanks = "                    ";
    int len = s.length();
    System.out.print(blanks.substring(0, w-len) + s);
  }
}
```

3.35

```
public class MultiplicationTable {
  public static void main(String[] args) {
    int rows = Integer.parseInt(args[0]);
    int[][] a = init(rows);
    print(a);
  }

  static int[][] init(int n) {
    int[][] a = new int[n][n];
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        a[i][j] = (i+1)*(j+1);
    return a;
  }

  static void print(int[][] a) {
    for (int i = 0; i < a.length; i++) {
      for (int j = 0; j < a.length; j++)
        print(a[i][j],5);
      System.out.println();
    }
  }

  static void print(int n, int w) {
    // prints n right-justified in a field on width w:
    String s = "" + n, blanks = "                    ";
    int len = s.length();
    System.out.print(blanks.substring(0, w-len) + s);
  }
}
```

3.36

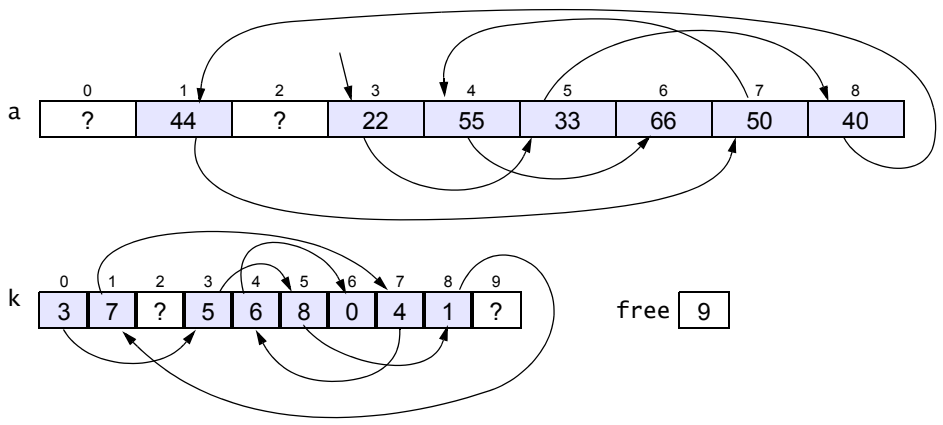
```
public class Main {
  public Main(String file) {
    int words = 0, chars = 0;
```

```
    try {
      BufferedReader in = new BufferedReader(new FileReader(file));
      String line = in.readLine();
      while(line!=null) {
        StringTokenizer parser
          = new StringTokenizer(line," ,:;-.?!");
        while (parser.hasMoreTokens()) {
          ++words;
          String word = parser.nextToken().toUpperCase();
          chars += word.length();
        }
        line = in.readLine();
      }
      in.close();
    } catch(IOException e) { System.out.println(e); }
    System.out.println("words: " + words);
    System.out.println("characters: " + chars);
  }

  public static void main(String[] args) {
    new Main(args[0]);
  }
}
```

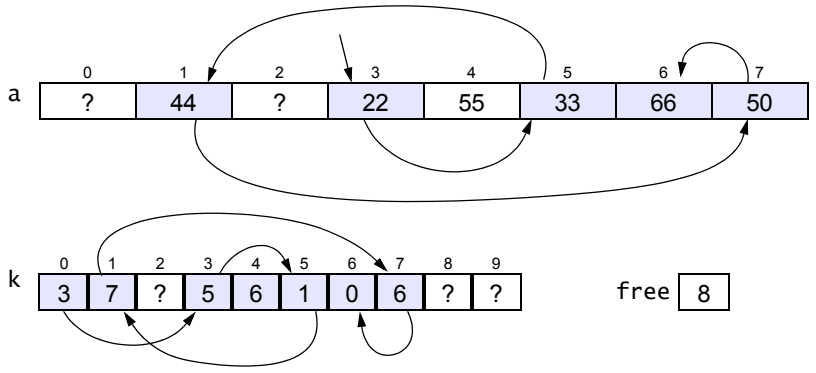
Chapter 4
Linked Structures

Exercises

4.1



4.2



4.3 The Fibonacci number F_{1000} has 209 digits.

Programming Problems

4.1
```
void delete(int[] a, int n, int x) {
  // preconditions: [0] <= ... <= a[n-1], and n <= a.length;
  // postconditions: a[0] <= ... <= a[n-2], and x is deleted;
  int i = 0;  // find the first index i for which a[i] > x:
  while (i < n && a[i] <= x)
    ++i;
  // shift {a[i],...,a[n-1]} into {a[i-1],...,a[n-2]}:
  if (i<n-1) System.arraycopy(a, i, a, i-1, n-i);
  a[n-1] = 0;
}
```

4.2
```
int size(Node list) {
  // returns: the number of nodes in the specified list;
  int count = 0;
  while (list != null) {
    ++count;
    list = list.next;
  }
  return count;
}
```

4.3
```
int sum(Node list) {
  // returns: the sum of the integers in the specified list;
  int sum = 0;
  while (list != null) {
    sum += list.data;
    list = list.next;
  }
  return sum;
}
```

4.4
```
void deleteLast(Node list) {
  // precondition: the specified list has at least two nodes;
  // postcondition: the last node in the list has been deleted;
  if (list == null || list.next == null)
    throw new IllegalStateException();
  while (list.next.next != null) {
    list = list.next;
  }
  list.next = null;
}
```

4.5

```
Node copy(Node list) {
  // returns: the sum of the integers in the specified list;
  if (list == null) return null;
  Node clone = new Node(list.data);
  for (Node p=list, q=clone; p.next != null; p=p.next, q=q.next)
    q.next = new Node(p.next.data);
  return clone;
}
```

4.6

```
Node sublist(Node list, int m, int n) {
  // returns: a new list that contains copies of the q-p nodes of the
  //   specified list, starting with node number p (starting with 0);
  if (m < 0 || n < m) throw new IllegalArgumentException();
  if (n == m) return null;
  for (int i=0; i<m; i++)
    list = list.next;
  Node clone = new Node(list.data);
  Node p=list, q=clone;
  for (int i=m+1; i<n; i++) {
    if (p.next == null) throw new IllegalArgumentException();
    q.next = new Node(p.next.data);
    p = p.next;
    q = q.next;
  }
  return clone;
}
```

4.7

```
void append(Node list1, Node list2)
  // precondition: list1 has at least one node;
  // postcondition: list1 has list2 appended to it;
  if (list1 == null)
    throw new IllegalArgumentException("list1 == null");
  while (list.next != null) {
    list1 = list1.next;
  }
  list1.next = list2;
}
```

4.8

```
Node concat(Node list1, Node list2) {
  // returns: a new list that contains a copy of list1, followed by
  //   a copy of list2;
  Node list3 = new Node(0), p=list1, q=list3;
  while (p != null) {
    q.next = new Node(p.data);
    p = p.next;
```

```
    q = q.next;
  }
  p=list2;
  while (p != null) {
    q.next = new Node(p.data);
    p = p.next;
    q = q.next;
  }
  return list3.next;  // discard dummy head node
}
```

4.9

```
void replace(Node list, int i, int x)
  // replaces the value of element number i with x;
  String msg = "The list must have at least " + i + " elements.";
  if (list == null) throw new IllegalArgumentException(msg);
  for (int j = 0; j < i; j++) {
    if (list.next == null)
      throw new IllegalArgumentException(msg);
    list = list.next;
  }
  list.data = x;
}
```

4.10

```
void swap(Node list, int i, int j) {
  // swaps the ith element with the jth element;
  if (i < 0 || j < 0) throw new IllegalArgumentException();
  if (i == j) return;
  Node p=list, q=list;
  for (int ii=0; ii<i; ii++) {
    if (p == null) throw new IllegalStateException();
    p = p.next;
  }
  for (int jj=0; jj<j; jj++) {
    if (q == null) throw new IllegalStateException();
    q = q.next;
  }
  int pdata=p.data, qdata=q.data;
  p.data = qdata;
  q.data = pdata;
  return;
}
```

4.11

```
Node merged(Node list1, Node list2)
  // precondition: list1 and list2 are both in ascending order;
  // returns: a new list that contains all the elements of list1 and
```

```
    //   list2 in ascending order;
    Node list = new Node(0), p=list, p1=list1, p2=list2;
    while (p1 != null && p2 != null) {
      if (p1.data < p2.data) {
        p = p.next = new Node(p1.data);
        p1 = p1.next;
      } else {
        p = p.next = new Node(p2.data);
        p2 = p2.next;
      }
    }
    while (p1 != null) {
      p = p.next = new Node(p1.data);
      p1 = p1.next;
    }
    while (p2 != null) {
      p = p.next = new Node(p2.data);
      p2 = p2.next;
    }
    return list.next;
  }
```

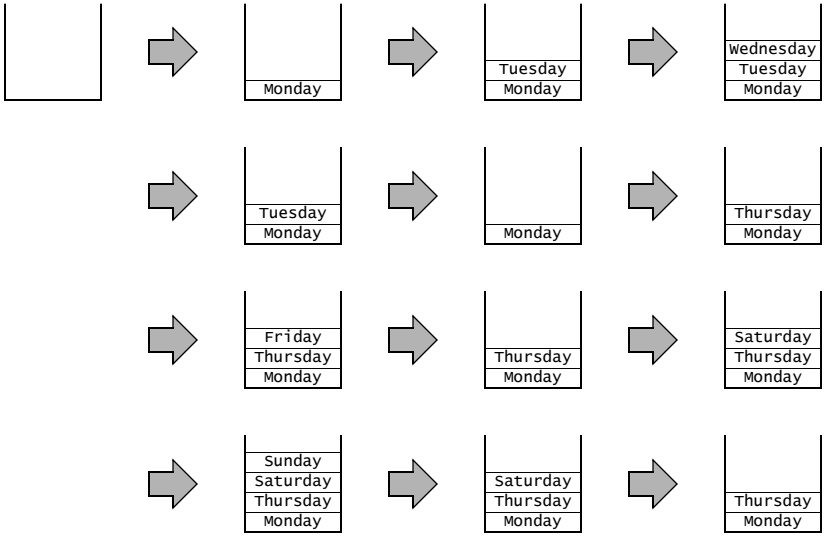
4.12

```
  private void shuffle(Node list) {
    // performs a perfect shuffle on the specified list;
    Node p=list, q=list;
    while (p != null) {
      p = p.next;
      if (p != null) p = p.next;
      q = q.next;
    }
    // now q = middle node:
    Node m = q;
    p = list;
    Node t=p.next, tt=m.next;
    while (m.next != null) {
      tt = m.next;
      p.next = m;
      p = m.next = t;
      t = p.next;
      m = tt;
    }
    p.next = m;
  }
```

Chapter 5
Stacks

Exercises

5.1



5.2

a. $x\ 4\ +\ y\ 8\ z\ +\ /\ -$

b. $x\ 6\ y\ /\ 5\ z\ -\ *\ /$

c. $*\ /\ x\ /\ 2\ y\ -\ 7\ z$

5.3

a. $x\ +\ 4\ /\ y\ -\ 3\ +\ z$; *i.e.*, $(x+4)/(y-3)+z$

b. $x\ -\ 9\ +\ y\ *\ 6\ -\ z$; *i.e.*, $x-(9+y)*(6-z)$

c. $x\ /\ 3\ /\ y\ /\ 2\ /\ z$; *i.e.*, $x/(3/(y/(2/z)))$

Programming Problems

5.1

```
public class IntArrayStack {
  private int[] a;
  private int size;

  public IntArrayStack(int capacity) {
    a = new int[capacity];
  }

  public boolean isEmpty() {
    return (size == 0);
  }

  public int peek() {
    if (size == 0) throw new IllegalStateException("stack is empty");
    return a[size-1];
  }

  public int pop() {
    if (size == 0) throw new IllegalStateException("stack is empty");
    int n = a[--size];
    return n;
  }

  public void push(int n) {
    if (size == a.length) resize();
    a[size++] = n;
  }

  public int size() {
    return size;
  }

  private void resize() {
    int[] aa = a;
    a = new int[2*aa.length];
    System.arraycopy(aa, 0, a, 0, size);
  }
}
```

5.2

```
import chap05.list01.Stack;
import java.util.ArrayList;

public class ArrayListStack implements Stack {
  private ArrayList a;

  public ArrayListStack(int capacity) {
    a = new ArrayList(capacity);
  }

  public boolean isEmpty() {
    return (a.isEmpty());
  }

  public Object peek() {
    int n = a.size();
    if (n == 0) throw new IllegalStateException("stack is empty");
    return a.get(n-1);
  }

  public Object pop() {
    int n = a.size();
    if (n == 0) throw new IllegalStateException("stack is empty");
    return a.remove(n-1);
  }

  public void push(Object object) {
    a.add(object);
  }

  public int size() {
    return a.size();
  }
}
```

5.3

```
public String toString() {
  if (size == 0) return "()";
  int i = size;
  StringBuffer buf = new StringBuffer("(" + a[--i]);
  while (i > 0)
    buf.append("," + a[--i]);
  return buf + ")";
}
```

5.4
```
public String toString() {
  if (size == 0) return "()";
  StringBuffer buf = new StringBuffer("(" + top.object);
  for (Node p=top.next; p!=null; p = p.next)
    buf.append("," + p.object);
  return buf + ")";
}
```

5.5
```
public boolean equals(Object obj) {
  if (obj == this) return true;
  if (!(obj instanceof ArrayStack)) return false;
  ArrayStack that = (ArrayStack)obj;
  if (that.size != this.size) return false;
  for (int i=0; i<size; i++)
    if (!that.a[i].equals(this.a[i])) return false;
  return true;
}
```

5.6
```
public boolean equals(Object obj) {
  if (obj == this) return true;
  if (!(obj instanceof LinkedStack)) return false;
  LinkedStack that = (LinkedStack)obj;
  if (that.size != this.size) return false;
  Node p=this.top, q = that.top;
  while (p != null && q != null) {
    if (!p.object.equals(q.object)) return false;
    p = p.next;
    q = q.next;
  }
  return true;
}
```

5.7
```
public Object[] toArray() {
  Object[] a = new Object[size];
  int i = 0;
  for (Node p=top; p!=null; p = p.next)
    a[i++] = p.object;
  return a;
}
```

5.8
```
public ArrayStack toArrayStack() {
  ArrayStack temp = new ArrayStack(2*size);
  for (Node p=top; p!=null; p = p.next)
    temp.push(p.object);
  ArrayStack s = new ArrayStack(2*size);
  for (Node p=top; p!=null; p = p.next)
```

```
      s.push(temp.pop());
    return s;
  }
```

5.9

```
  public LinkedStack toLinkedStack() {
    LinkedStack s = new LinkedStack();
    for (int i=0; i<size; i++)
      s.push(a[i]);
    return s;
  }
```

5.10

```
  public Object peekSecond() {
    if (size < 2) throw new java.util.NoSuchElementException();
    return top.next.object;
  }
```

5.11

```
  public Object popSecond() {
    if (size < 2) throw new java.util.NoSuchElementException();
    --size;
    Object second = a[size-1];
    a[size-1] = a[size];
    a[size] = null;
    return second;
  }
```

5.12

```
  public Object bottom() {
    if (size == 0) throw new java.util.NoSuchElementException();
    Node p = top;
    while (p.next != null)
        p =p.next;
    return p.object;
  }
```

5.13

```
  public Object popBottom() {
    if (size == 2) throw new java.util.NoSuchElementException();
    Object bottom = a[0];
    System.arraycopy(a, 1, a, 0, --size);
    a[size] = null;
    return bottom;
  }
```

5.14

```
  public void reverse() {
    if (size < 2) return;
    LinkedStack stack1 = new LinkedStack();
    LinkedStack stack2 = new LinkedStack();
    while (this.size > 0)
      stack1.push(this.pop());
    while (stack1.size > 0)
```

```
    stack2.push(stack1.pop());
  while (stack2.size > 0)
    this.push(stack2.pop());
}
```

5.15

```
Object removeSecondFromTop(Stack stack) {
  int n = stack.size();
  if (n<2) throw new IllegalArgumentException("stack is empty");
  Object top1 = stack.pop();
  Object top2 = stack.pop();
  stack.push(top1);
  return top2;
}
```

5.16

```
Object removeBottom(Stack stack) {
  int n = stack.size();
  if (n<1) throw new IllegalArgumentException("stack is empty");
  if (n<2) return stack.pop();
  Stack auxStack = new LinkedStack();
  for (int i=0; i<n-1; i++)
    auxStack.push(stack.pop());
  Object bottom = stack.pop();
  for (int i=0; i<n-1; i++)
    stack.push(auxStack.pop());
  return bottom;
}
```

5.17

```
void reverse(Stack stack) {
  if (stack.size() < 2) return;
  Stack stack1 = new LinkedStack();
  Stack stack2 = new LinkedStack();
  while (stack.size() > 0)
    stack1.push(stack.pop());
  while (stack1.size() > 0)
    stack2.push(stack1.pop());
  while (stack2.size() > 0)
    stack.push(stack2.pop());
}
```

5.18

```
Stack reversed(Stack stack) {
  Stack stack1 = new LinkedStack();
  Stack stack2 = new LinkedStack();
  while (stack.size() > 0) {
    stack1.push(stack.peek());
    stack2.push(stack.pop());
  }
```

```
    while (stack2.size() > 0)
      stack.push(stack2.pop());
    return stack1;
  }
```

5.19

```
  public void push(Object object) {
    if (contains(object)) return;
    if (size == a.length) resize();
    a[size++] = object;
  }
  private boolean contains(Object x) {
    for (int i=0; i<size; i++)
      if (a[i]==x) return true;
    return false;
  }
```

5.20

```
  public void push(Object object) {
    if (object == null)
      throw new IllegalArgumentException("object is null");
    if (size == a.length) resize();
    a[size++] = object;
  }
```

5.21

```
  /**
   * The <code>ArrayStack</code> class implements the
   * <code>Stack</code> interface defined in the
   * <code>chap05.list01</code> package. Its instances are
   * last-in-first-out (LIFO) containers that store their
   * elements in a backing array.
   */
  public class ArrayStack implements Stack { ...

    /**
     * Creates an empty stack with a specified capacity.
     *
     * @param  capacity  the length of the backing array.
     */
    public ArrayStack(int capacity) { ...

    /**
     * Reports whether this stack is empty.
     *
     * @return  <code>true</code> if and only if this stack contains
     *          no elements.
     */
```

```
    public boolean isEmpty() { ...

   /**
    * Returns a reference to the top element on this stack, leaving
    * the stack unchanged.
    *
    * @return     the top element on this stack.
    * @exception  IllegalStateException  if this stack is empty.
    */
   public Object peek() { ...

   /**
    * Removes the top element on this stack and returns it.
    *
    * @return     the top element on this stack.
    * @exception  IllegalStateException  if this stack is empty.
    */
   public Object pop() { ...

   /**
    * Inserts the specified object onto the top of this stack.
    *
    * @param  object  the element to be pushed onto this stack.
    */
   public void push(Object object) { ...

   /**
    * Returns the number of elements in this stack.
    *
    * @return     the number of elements in this stack.
    */
   public int size() { ...
```

5.22

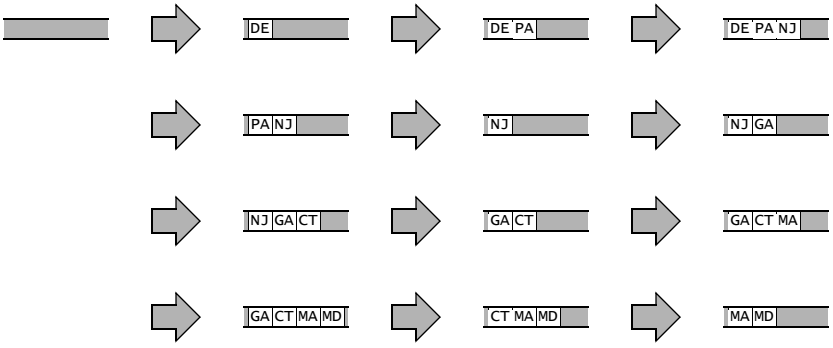
```
/**
 * The <code>LinkedStack</code> class implements the
 *  <code>Stack</code> interface defined in the
 *  <code>chap05.list01</code> package. Its instances are
 * last-in-first-out (LIFO) containers that store their
 * elements in a backing array.
 */
public class LinkedStack implements Stack { ...
```

```
/**
 * Reports whether this stack is empty.
 *
 * @return  <code>true</code> if and only if this stack contains
 *          no elements.
 */
 public boolean isEmpty() { ...

/**
 * Returns a reference to the top element on this stack, leaving
 * the stack unchanged.
 *
 * @return     the top element on this stack.
 * @exception  IllegalStateException  if this stack is empty.
 */
public Object peek() { ...

/**
 * Removes the top element on this stack and returns it.
 *
 * @return     the top element on this stack.
 * @exception  IllegalStateException  if this stack is empty.
 */
public Object pop() { ...

/**
 * Inserts the specified object onto the top of this stack.
 *
 * @param  object  the element to be pushed onto this stack.
 */
public void push(Object object) { ...

/**
 * Returns the number of elements in this stack.
 *
 * @return  the number of elements in this stack.
 */
public int size() { ...
```

Chapter 6
Queues

Exercises

6.1



6.2 The expression `new Node(object,head.prev,head)` in permuted statement

```
head.prev.next = head.prev = new Node(object,head.prev,head);
```

creates the specified new node. The value of that expression is a reference to that new node. The permuted statement first assigns that reference to `head.prev` . That changes the value of `head.prev` . So when the second (chained) assignment is performed, the new node reference gets assigned to the `next` field of that new node, since `head.prev` refers to it. Thus, the new node's `next` field ends up refering to itself.

6.3 The total capital gains on these transactions are $3250:

Date	Bought	Sold	Gain
Feb 20	200@ $20		
Apr 15		100@ $22	100@ $2 = $200
Jun 25	150@ $25		
Aug 10	250@ $24		
Sep 20		100@ $28	100@ $8 = $800
		150@ $28	150@ $3 = $450
		50@ $28	50@ $4 = $200
Nov 15		200@ $32	200@ $8 = $1600

6.4 With the following implementation, the worst case would be alternating calls to `add()` and `remove()`. In that case, both operations run in linear time:

```
void add(Object x) {
  while (!stack2.isEmpty())
    stack1.push(stack2.pop());
  stack1.push(x);
}
Object remove() {
  while (!stack1.isEmpty())
    stack2.push(stack1.pop());
  return stack2.pop();
}
int size() {
  return stack1.size() + syack2.size();
}
```

Programming Problems

6.1

```
import chap06.list01.Queue;
public class ArrayQueue implements Queue {
  private Object[] a;
  private int front, back;

  public ArrayQueue(int capacity) {
    a = new Object[capacity];
  }

  public void add(Object object) {
    if (back == a.length) resize();
    a[back++] = object;
  }

  public Object first() {
    if (size()==0) throw new IllegalStateException("queue is empty");
    return a[front];
  }

  public boolean isEmpty() {
    return (size() == 0);
  }

  public Object remove() {
    if (size()==0) throw new IllegalStateException("queue is empty");
    Object object=a[front];
    a[front++] = null;
    return object;
  }

  public int size() {
    return back - front;
  }

  private void resize() {
    Object[] aa = a;
    a = new Object[2*aa.length];
    System.arraycopy(aa, front, a, 0, size());
  }
}
```

6.2

```
/**
 * The <code>ArrayQueue</code> class implements the
 * <code>Queue</code> interface defined in the
 * <code>chap06.list01</code> package. Its instances are
 * first-in-first-out (FIFO) containers that store their
 * elements in a backing array.
 */
public class ArrayQueue implements Queue { ...

    /**
     * Creates an empty queue with a specified capacity.
     *
     * @param  capacity  the length of the backing array.
     */
    public ArrayQueue(int capacity) { ...

    /**
     * Adds the specified element to the back of this queue.
     *
     * @param  object  the element to be added to this queue.
     */
    public void add(Object object) { ...

    /**
     * Returns the element at the front of this queue.
     *
     * @return  the element at the front of this queue.
     * @throws  IllegalStateException  if this queue is empty
     */
    public Object first() { ...

    /**
     * Reports whether this queue is empty.
     *
     * @return  <code>true</code> if and only if this queue contains
     *          no elements.
     */
    public boolean isEmpty() { ...
```

```
    /**
     * Removes and returns the element at the front of this queue.
     *
     * @return  the element at the front of this queue.
     * @throws  IllegalStateException  if this queue is empty
     */
    public Object remove() { ...

    /**
     * Returns the number of elements in this queue.
     *
     * @return  the number of elements in this queue.
     */
    public int size() { ...
```

6.3

```
/**
 * The <code>LinkedQueue</code> class implements the
 * <code>Queue</code> interface defined in the
 * <code>chap06.list01</code> package. Its instances are
 * first-infirst-out (FIFO) containers that store their
 * elements in a linked list.
 */
public class LinkedQueue implements Queue { ...

The rest is the same as in Problem 6.2 above.
```

6.4

```
import chap06.list01.Queue;
import java.util.ArrayList;
public class ArrayListQueue implements Queue {
  private ArrayList a;

  public ArrayListQueue(int capacity) {
    a = new ArrayList(capacity);
  }

  public void add(Object object) {
    a.add(object);
  }

  public Object first() {
    if (size()==0) throw new IllegalStateException("queue is empty");
    return a.get(0);
  }
```

```
  public boolean isEmpty() {
    return a.isEmpty();
  }

  public Object remove() {
    if (size()==0) throw new IllegalStateException("queue is empty");
    return a.remove(0);
  }

  public int size() {
    return a.size();
  }
}
```

6.7

```
public String toString() {
  if (size == 0) return "()";
  StringBuffer buf = new StringBuffer("(" + head.object);
  for (Node p=head.next; p!=null; p = p.next)
    buf.append("," + p.object);
  return buf + ")";
}
```

6.8

```
public boolean equals(Object obj) {
  if (obj == this) return true;
  if (!(obj instanceof LinkedQueue)) return false;
  LinkedQueue that = (LinkedQueue)obj;
  if (that.size != this.size) return false;
  Node p=this.head, q = that.head;
  while (p != null && q != null) {
    if (!p.object.equals(q.object)) return false;
    p = p.next;
    q = q.next;
  }
  return true;
}
```

6.10

```
public Object[] toArray() {
  Object[] a = new Object[size];
  int i = 0;
  for (Node p=head; p!=null; p = p.next)
    a[i++] = p.object;
  return a;
}
```

6.11
```
public ArrayQueue toArrayQueue() {
  ArrayQueue q = new ArrayQueue(2*size);
  for (Node p=head; p!=null; p = p.next)
    q.add(p.object);
  return q;
}
```

6.12
```
public Object getSecond() {
  if (size < 2) throw new java.util.NoSuchElementException();
  return head.next.object;
}
```

6.13
```
public Object removeSecond() {
  if (size < 2) throw new java.util.NoSuchElementException();
  Object object=head.prev.object;
  head.next.next = head.next.next.next;
  head.next.next.prev = head;
  --size;
  return object;
}
```

6.14
```
public Object last() {
  if (size == 0) throw new java.util.NoSuchElementException();
  return head.prev.object;
}
```

6.15
```
public void reverse() {
  Node p = head.next;
  while (p != head) {
    Node temp = p.next;
    p.next = p.prev;
    p = p.prev = temp;
  }
  head.next = p.prev;
}
```

6.16
```
Object removeSecond(Queue q) {
  int n = q.size();
  if (n<2) throw new IllegalArgumentException();
  q.add(q.remove());
  Object x = q.remove();
  for (int i=2; i<n; i++)
    q.add(q.remove());
  return x;
}
}
```

6.17

```
public static Object lastElement(Queue queue) {
  Queue aux = new LinkedQueue();
  while (queue.size() > 1)
    aux.add(queue.remove());
  Object last = queue.remove();
  while (aux.size() > 0)
    queue.add(aux.remove());
  return last;
}
```

6.18

```
void reverse(Queue q) {
void reverse(Queue q) {
  int n = q.size();
  if (n<2) return;
  Stack s = new ArrayStack(n);
  for (int i=0; i<n; i++)
    s.push(q.remove());
  for (int i=0; i<n; i++)
    q.add(s.pop());
}
```

6.19

```
public static Queue reversed(Queue queue) {
  Stack auxStack = new LinkedStack();
  Queue auxQueue = new LinkedQueue();
  while (queue.size() > 0) {
    Object object = queue.remove();
    auxStack.push(object);
    auxQueue.add(object);
  }
  while (auxQueue.size() > 0) {
    queue.add(auxQueue.remove());
    auxQueue.add(auxStack.pop());
  }
  return auxQueue;
}
```

Chapter 7
Collections

Exercises

7.1
```
[CH, JP, IN, ID, AU, NZ]
[NZ, JP, IN, ID, AU]
[NZ, AU, IN, ID]
[NZ, AU, IN]
```

7.2
```
[CA, US, MX, AR, BR, CH]
[MX, AR, BR]
[CA, US, MX, CO, BR, CH]
[MX, CO, BR]
[CA, US, MX, CO, VE, CH]
[MX, CO, VE]
```

7.3
```
[CA, US, MX, AR, BR, CH]
AR
[CA, US, MX, BR, CH]
[US, MX, BR, CH]
[US, BR]
```

7.4
```
[CA, US, MX, AR, BR, CH]
[CA, US, MX, CO, BR, CH]
[CA, US, null, CO, BR, CH]
```

7.5
```
[CA, US, MX, AR, BR, CH]
[CA, MX, AR, BR, CH]
[CA, AR, BR, CH]
[CA, AR, BR]
```

7.6

a. The list's size increases from 0 to 5; content are: 0, 10, 20, 30, 40 in ascending order.

b. `"[0]0, [1]10, [2]20, [3]30, [4]40"`

c. Three more items are inserted so that `size` increases from 5 to 8, and –80 precedes 0, –70 precedes 10, –60 precedes 20.

d. Three items are removed so that `size` increases from 8 to 5; line 16 removes 0, line 17 removes 20, and line 18 removes 40.

e.
```
initial size is 0
empty
**current size is 5
[0]0, [1]10, [2]20, [3]30, [4]40
****current size is 8
[0]-80, [1]0, [2]-70, [3]10, [4]-60, [5]20, [6]30, [7]40
removing 0
removing 20
removing 40
******current size is 5
[0]-80, [1]-70, [2]10, [3]-60, [4]30
```

7.7 **a.** Each time line 9 executes `itr.nextIndex()` returns a higher index. So the values 6.6 5.5 –2.2 –4.4 9.9 are inserted at indexes 0, 1, 2, 3, 4. Output from line 10 will be this:

```
current size is 5
```

b. Method `printA()` prints the values in order of their indexes. Method `printB()` prints the values in reverse order of their indexes.

c. The `ListIterator itr` is moved from the end back to beginning of `dList`. All of the values in the list are negated, changed to these: –6.6, –5.5, 2.2, 4.4, and –9.9.

d. Since `itr` is at the beginning of `dList`, the item at index 0 with value –6.6 is removed, *i.e.* values are changed to these: –5.5, 2.2, 4.4, –9.9.

e.
```
testing itr.add()
current size is 5
calling printB( dList )
[4]9.9
[3]-4.4
[2]-2.2
[1]5.5
[0]6.6
testing itr.set()
changing node at index 4 to -9.9
changing node at index 3 to 4.4
changing node at index 2 to 2.2
changing node at index 1 to -5.5
changing node at index 0 to -6.6
**current size is 5
calling printA( dList )
[0]-6.6
[1]-5.5
[2]2.2
[3]4.4
[4]-9.9
testing itr.remove()
current size is 4
calling printA( dList )
[0]-5.5
[1]2.2
[2]4.4
[3]-9.9
```

Programming Problems

7.1

```
public ArrayCollection() {
}
public ArrayCollection(int capacity) {
    a = new Object[capacity];
}
public ArrayCollection(Object[] objects) {
    a = new Object[objects.length];
    for (int i=0; i<objects.length; i++)
      if (objects[i] != null) a[size++] = objects[i];
}
```

7.2

```
public Object getLast() {
  // returns a reference to the last element of this collection;
  // or returns null if this collection is empty;
  for (int i=a.length-1; i>=0; i--)
    if (a[i] != null) return a[i];
  return null;
}
```

7.3

```
public Object getLast(Collection collection) {
  // returns a reference to the last element of the collection;
  // or returns null if the given collection is empty;
  Iterator it = collection.iterator();
  Object object = null;
  while (it.hasNext())
    object = it.next();
  return object;
}
```

7.4

```
public boolean removeLast() {
  // removes the last element if this collection is not empty;
  // returns true if and only if it changes this collection;    for
(int i=a.length-1; i>=0; i--)
  for (int i=a.length-1; i>=0; i--)
    if (a[i] != null) {
      a[i] = null;
      --size;
      return true;
    }
  return false;
}
```

7.5

```
public boolean removeLast(Collection collection) {
   // removes the last element if the collection is not empty;
   // returns true if and only it changes the given collection;
   Iterator it = collection.iterator();
   Object object = null;
   if (!it.hasNext()) return false;
   while (it.hasNext())
     object = it.next();
   it.remove();
   return true;
 }
```

7.6

```
public int frequency(Object object) {
   // returns the number of occurrences of the given object
   // in this collection;
   int f=0;
   for (Node p=head.next; p!=head; p = p.next)
     if (p.object.equals(object)) ++f;
   return f;
 }
```

7.7

```
 public int frequency(Collection collection, Object object) {
   // returns the number of occurrences of the given object
   // in the given collection;
   Iterator it = collection.iterator();
   int f=0;
   while (it.hasNext())
     if (it.next().equals(object)) ++f;
   return f;
 }
```

7.8

```
public boolean removeOdd() {
   // removes every other element (the first, the third, etc.) from
   // this collection; returns true iff this collection is changed;
   if (size < 2) return false;
   Node p=head.next;
   while (p.next!=head) {
     p = p.next = p.next.next;
     --size;
   }
   return true;
 }
```

7.9

```
public boolean removeOdd(Collection collection) {
  // removes every other element (the first, the third, etc.)
  // from the given collection; returns true iff it is changed;
  if (collection.size() < 2) return false;
  Iterator it = collection.iterator();
  while (it.hasNext()) {
    it.next();
    if (it.hasNext()) {
      it.next();
      it.remove();
    }
  }
  return true;
}
```

7.10

```
public class LinkedCollection extends AbstractCollection {
  private static class Node {
    Object object;
    Node prev, next;
    Node() { prev = next = this; }
    Node(Object o, Node p, Node n) { object=o; prev=p; next=n; }
  }
  private int size;
  private Node head = new Node();  // dummy head node
  private Node tail = new Node();  // dummy tail node

  public LinkedCollection() {
    head.next = head.prev = tail;
    tail.next = tail.prev = head;
  }

  public boolean add(Object object) {
    tail.prev = tail.prev.next = new Node(object, tail.prev, tail);
    ++size;
    return true;  // no object is rejected
  }

  public Iterator iterator() {
    return new Iterator() {    // anonymous inner class
      private Node cursor=head.next;  // current element node
      private boolean okToRemove=false;
      public boolean hasNext() { return cursor != tail; }
```

```
        public Object next() {
          if (cursor == head) throw new RuntimeException();
          okToRemove = true;
          Object object=cursor.object;
          cursor = cursor.next;
          return object;
        }
        public void remove() {
          if (!okToRemove) throw new IllegalStateException();
          cursor.prev = cursor.prev.prev;
          cursor.prev.next = cursor;
          --size;
          okToRemove = false;  // must call next() again before remove()
        }
      };
    }

    public int size() {
      return size;
    }
  }
```

7.11

```
  public boolean addAll(Collection collection) {
    for (Iterator it=collection.iterator(); it.hasNext(); )
      add(it.next());
    return true;
  }
```

7.12

```
  public void clear() {
    for (Iterator it=iterator(); it.hasNext(); )
      it.remove();
  }
```

7.13

```
  public boolean containsAll(Collection collection) {
    for (Iterator it=collection.iterator(); it.hasNext(); )
      if(!contains(it.next())) return false;
    return true;
  }
```

7.14

```
  public boolean retainAll(Collection collection) {
    boolean changed=false;
    for (Iterator it=iterator(); it.hasNext(); )
      if(!collection.contains(it.next())) {
        it.remove();
        changed = false;
```

```
      }
    return changed;
  }
```

7.15

```
  public Object[] toArray() {
    int n=size();
    Object[] objects = new Object[n];
    Iterator it=iterator();
    for (int i=0; i<n; i++)
      objects[i] = it.next();
    if (objects.length>n) objects[n] = null;
    return objects;
  }
```

Chapter 8
Lists

Exercises

8.1 One.

8.2 It depends upon the initial contents of the list. If it has fewer than six elements, then the call `set(6, "BE")` will throw an `IndexOutOfBoundsException`. Otherwise, the end result would be to insert `"BE"` at position 6 while shifting the existing elements that are in positions 2–6 down to positions 1–5. The existing element at position 2 will be returned.

8.3 **a.** See Figure 8.1 on the next page.

b. See Figure 8.2 on the next page.

8.4 **a.** There are $n + 1$ terms. The average exponent for the n non-constant terms is $n/2$. So the total number of operations averages $n(n/2) = \Theta(n^2)$.

b. With Horner's method, there are only n multiplications and n additions. So the total number of operations is $2n = \Theta(n)$.

Programming Problems

8.1

```
int frequency(List list, Object object) {
// returns the number of occurrences of object in list
  int f = 0;
  for (Iterator it = list.iterator(); it.hasNext(); )
    if (it.next().equals(object)) ++f;
  return f;
}
```

8.2

```
public class TestSequence {
  public TestSequence() {
    String[] countries = {"CA","DE","FR","GB","IT","JP","RU","US"};
    Sequence g8 = new Sequence(Arrays.asList(countries));
```

```
    print(g8);
    g8.remove(6);
    print(g8);
  }

  void print(Sequence s) {
    for (Iterator it =  s.iterator(); it.hasNext(); )
      System.out.print(it.next() + " ");
    System.out.println();
  }

  public static void main(String[] args) {
    new TestSequence();
  }
}
```

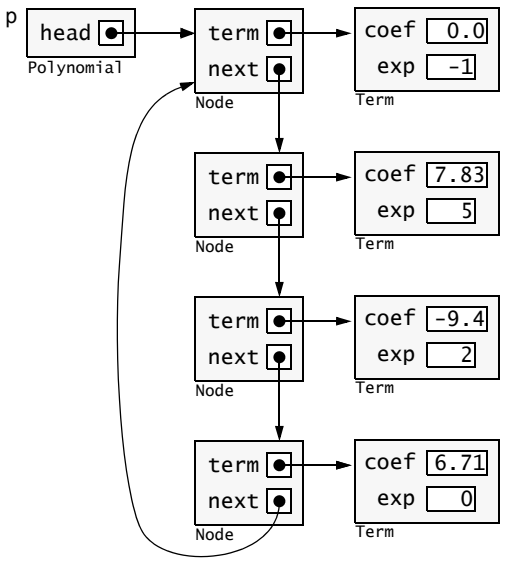


Figure 8.1

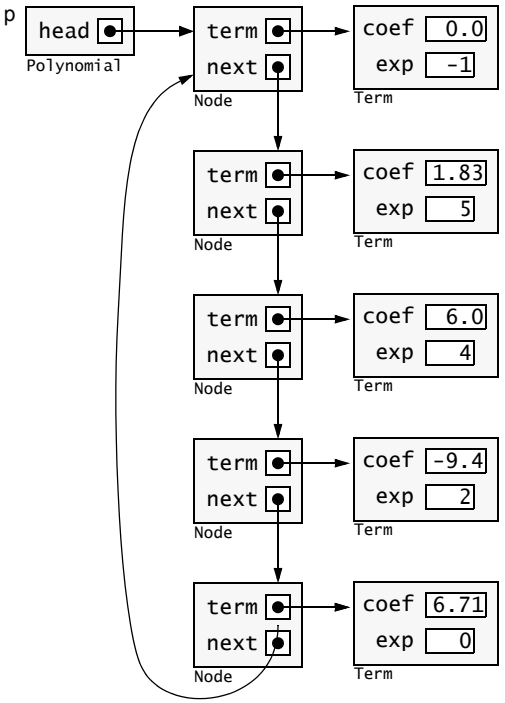


Figure 8.2

8.3

```
public class TestTerm {
  public TestTerm() {
    Term t1 = new Term(7.0, 4);
    System.out.println(t1);
    System.out.println("t1.getCoef(): " + t1.getCoef());
    System.out.println("t1.getExp(): " + t1.getExp());
    Term t2 = new Term(t1);
    System.out.println(t2);
    System.out.println("t2.equals(t1): " + t2.equals(t1));
    Term t3 = t1.times(-2.0);
    System.out.println(t3);
    System.out.println("t3.equals(t1): " + t3.equals(t1));
    System.out.println(t3.abs());
    System.out.println("t1.plus(t3): " + t1.plus(t3));
    System.out.println("t3.plus(t1).plus(t1): "
      + t3.plus(t1).plus(t1));
    System.out.println("t1.times(t3): " + t1.times(t3));
  }
  public static void main(String[] args) {
    new TestTerm();
  }
}
```

8.4

```
public class TestPolynomial {
  public TestPolynomial() {
    Polynomial p1 = new Polynomial(2.0, 3);
    System.out.println(p1);
    System.out.println("p1.degree(): " + p1.degree());
    Polynomial p2 = new Polynomial(p1);
    System.out.println("p2.equals(p1): " + p2.equals(p1));
    p2 = p1.plus(p2);
    System.out.println(p2);
    System.out.println("p2.equals(p1): " + p2.equals(p1));
    p1 = p1.plus(new Term(-9.0, 2));
    System.out.println(p1);
    System.out.println("p1.degree(): " + p1.degree());
    p1 = p1.plus(new Term(5.0, 8));
    System.out.println(p1);
    System.out.println("p1.degree(): " + p1.degree());
    p1 = p1.plus(p2);
    System.out.println(p1);
    System.out.println("p1.degree(): " + p1.degree());
    Polynomial p3 = p1.times(-2.0);
```

```
    System.out.println("p3: " + p3);
    System.out.println("p3.degree(): " + p3.degree());
    p1 = p1.times(new Polynomial(-5.0, 3));
    System.out.println(p1);
    System.out.println("p1.degree(): " + p1.degree());
    System.out.println("p1.valueAt(1.0): " + p1.valueAt(1.0));
    System.out.println("p1.valueAt(-1.0): " + p1.valueAt(-1.0));
    System.out.println("p1.valueAt(2.0): " + p1.valueAt(2.0));
    p1 = p1.times(new Polynomial(5.0, -3));
  }
  public static void main(String[] args) {
    new TestPolynomial();
  }
}
```

8.5

```
public Polynomial(double[] c) {
  Node p = head;
  for (int i = c.length - 1; i >= 0; i--)
    if (c[i] != 0.0) p = p.next = new Node(new Term(c[i], i), head);
}
```

8.6

```
public double[] toArray() {
  int n = degree() + 1;
  double[] a = new double[n];
  for (int i=0; i<n; i++)
      a[i] = 0.0;
  for (Node p = head.next; p != head; p = p.next)
      a[p.exp] = p.coef;
  return a;
}
```

8.7

```
public Polynomial derivative() {
  if (degree() < 1) return ZERO;
  Polynomial poly = new Polynomial();
  for (Node p = head.next; p != head; p = p.next) {
    double coef = p.exp*p.coef;
    if (coef != 0 && p.exp > 0)
      poly = poly.plus(new Polynomial(coef, p.exp-1));
  }
  return poly;
}
```

8.8

```
public class Polynomial {
  private Node head = new Node();  // dummy node;
  public static final Polynomial ZERO = new Polynomial();

  private Polynomial() {
  }

  public Polynomial(double coef, int exp) {
    head.next = new Node(coef, exp, head);
  }

  public Polynomial(Polynomial that) {  // copy constructor
    Node p = this.head, q = that.head.next;
    while (q != that.head) {
      p = p.next = new Node(q);
      p.next = head;
      q = q.next;
    }
  }

  public int degree() {
    return head.next.exp;
  }

  public boolean equals(Object object) {
    if (!(object instanceof Polynomial)) return false;
    Polynomial that = (Polynomial)object;
    Node p = this.head.next, q = that.head.next;
    while (q != that.head) {
      if (p.coef != q.coef || p.exp != q.exp) return false;
      p = p.next;
      q = q.next;
    }
    return p == head;
  }

  public Polynomial plus(Polynomial that) {
    if (this.equals(ZERO)) return new Polynomial(that);
    if (that.equals(ZERO)) return new Polynomial(this);
    Polynomial sum = new Polynomial(this);
    Node p = sum.head, q = that.head.next;
    while (p.next != sum.head && q != that.head) {
```

```
    if (p.next.exp <= q.exp) {
      if (p.next.exp == q.exp) p.next.coef += q.coef;
      else p.next = new Node(q.coef, q.exp, p.next);
      q = q.next;
    }
    p = p.next;
  }
  if (q != that.head) p.next = new Node(q.coef, q.exp, p.next);
  return sum;
}

public Polynomial times(double factor) {
  if (this.equals(ZERO) || factor==0.0) return ZERO;
  Polynomial result = new Polynomial(this);
  for (Node p = result.head.next; p != result.head; p = p.next)
    p.coef *= factor;
  return result;
}

public Polynomial times(Polynomial that) {
  if (this.equals(ZERO) || that.equals(ZERO)) return ZERO;
  Polynomial result = ZERO;
  for (Node p = this.head.next; p != this.head; p = p.next) {
    Polynomial temp = new Polynomial(that);
    for (Node q = temp.head.next; q != temp.head; q = q.next){
      q.coef *= p.coef;
      q.exp += p.exp;
    }
    result = result.plus(temp);
  }
  return result;
}

public double[] toArray() {
  int n = degree() + 1;
  double[] a = new double[n];
  for (int i=0; i<n; i++)
    a[i] = 0.0;
  for (Node p = head.next; p != head; p = p.next)
    a[p.exp] = p.coef;
  return a;
}
```

```
  public String toString() {
    if (this.equals(ZERO)) return "0";
    Node p = head.next;
    StringBuffer buf = new StringBuffer(p.coef + "x^" + p.exp);
    for (p = p.next; p != head; p = p.next) {
      buf.append( (p.coef<0 ? " - " : " + ") + Math.abs(p.coef));
      if (p.exp > 0) buf.append("x");
      if (p.exp > 1) buf.append("^" + p.exp);
    }
    return buf.toString();
  }

  public double valueAt(double x) {
    if (this.equals(ZERO)) return 0.0;
    double y = 0.0;
    Node p = head.next;
    int exp = p.exp;
    while (p.next != head) {
      double coef = p.coef;
      int nextExp = p.next.exp;
      y = (y + coef)*Math.pow(x, exp - nextExp);
      exp = nextExp;
      p = p.next;
    }
    return (y + p.coef)*Math.pow(x, exp);
  }

  private static class Node {
    double coef;
    int exp;
    Node next;
    Node() { this(0.0, -1, null); next = this; }
    Node(Node that) { this(that.coef, that.exp, that.next); }
    Node(double c, int e, Node n) { coef = c; exp = e; next = n; }
  }
}
```

Chapter 9
Lists

Exercises

9.1 No.

Suppose `Ann` and `Bob` both hash to 5 in an empty hash table with open addressing. If `Ann` is inserted first, it will be put in slot 5, and `Bob` will be put in slot 6. But if `Bob` is inserted first, it will be put in slot 5, and `Ann` will be put in slot 6.

9.2 See the solution on page 515.

9.3

a. Let $f(x) = \ln x$ and $g(x) = (x–1/x)/2$ for $x \geq 1$. Then $f'(x) = 1/x$ and $g'(x) = (1+1/x^2)/2$. Since $0 \leq (1–x)^2 = 1 –2x + x^2$, we have $2x \leq 1 + x^2$, so $1/x \leq (1+1/x^2)/2$. Thus, $f'(x) \leq g'(x)$ for all $x \geq 1$, and $f(1) = 0 = g(1)$. Therefore $f(x) \leq g(x)$ for all $x \geq 1$; *i.e.*, $\ln x \leq (x–1/x)/2$. Letting $q = x$ and $r = 1–1/x$ (so that $q = 1/(1–r)$), we have $\ln q \leq (x–1/x)/2 = (x^2–1)/(2x) = r(2–r)/(2(1–r))$. Thus, $\ln q – r\ln q = (1–r)\ln q \leq r(2–r)/2 = r – r^2/2$, $\ln q \leq r + r\ln q – r^2/2 = r(1 + \ln q – r/2)$, so $(\ln q)/r \leq 1 + \ln q – r/2$.

b. This is true because $(\ln q)/r = 1 + r/2 + r^2/3 + ... \geq 1 + r/2$.

c. This is true because $0 < r < 1$, so $r(1–r) < 1$ and $r < 1/(1–r) = q$.

d. This is true because $0 < r < 1$, so $(1–r) < 1$ and $q = 1/(1–r) > 1$; thus $q^2 > q$ and $(1 + q^2)/2 > (1 + q)/2$.

e. Ths follows directly from the fact that $q = 1 + r + r^2 + r^3 + ...$

f. Ths follows directly from the fact that $(\ln q)/r = 1 + r/2 + r^2/3 + r^3/4 + ...$

9.4 With linear probing, we don't need more than n = `size` probes. After finding n other elements, we will have checked them all and we'll know that the key is not in the table. You can't have more than n collisions.

The same is true with quadratic probing, although the `put()` method may fail once the load factor exceeds 50%, since the probe sequence reaches only half the slots.

The same is true with double hashing.

9.5 The worst case would be where all the keys are in one contiguous block.

9.6 This design would occupy a very large file. For example, if the keys are 100-byte strings, it would occupoy 200 GB. So this design would be very wasteful of space unless the table size were in the range of a hundred million.

This hash function is not perfect because the Java `hashCode()` method is not one-to-one.

9.7 With double hashing, the probe sequence is arithmetic, with a constant difference equal to the secondary hash value. If the array length is prime, then the probe sequence will reach every index of the array.

As an example of what can go wrong when the length is not prime, suppose it is 12. If the secondary hash value is a divisor (1, 2, 3, 4, or 6), then the probe sequence will not reach all the table slots. For example, suppose the primary hash value is 5 and the secondary hash value is 4. Then the probe sequence is 5, 9, 1, 5, 9, 1, 5, ... If slots 1, 5, and 9 are occupied, then the insert will fail even if the table is only 25% full.

Programming Problems

9.1

```
public class TestHashTable {
  public TestHashTable() {
    HashTable t = new HashTable(11);
    t.put("AT", new Country("Austria", "German", 32378, 8139299));
    t.put("FR", new Country("France", "French", 211200, 58978172));
    t.put("DE", new Country("Germany", "German", 137800, 82087361));
    t.put("GR", new Country("Greece", "Greek", 50900, 10707135));
    t.put("IT", new Country("Italy", "Italian", 116300, 56735130));
    t.put("PT", new Country("Portugal", "Portuguese",35672,9918040));
    t.put("SE", new Country("Sweden", "Swedish", 173732, 8911296));
    System.out.println("t.size(): " + t.size());
    System.out.println("t.get(\"AT\"): " + t.get("AT"));
    System.out.println("t.get(\"IT\"): " + t.get("IT"));
    System.out.println("t.remove(\"PT\"): " + t.remove("PT"));
    System.out.println("t.size(): " + t.size());
    System.out.println("t.remove(\"US\"): " + t.remove("US"));
    System.out.println("t.size(): " + t.size());
  }

  public static void main(String[] args) {
    new TestHashTable();
  }
```

9.2

```
// the same test driver as in Problem 9.1 can be used.
```

9.5

```
private int nextProbe(int h, int i) {
  return (h + i*i)%entries.length;      // Quadratic Probing
}
```

9.6

```
public class HashTable implements Map {  // open addressing
  private Entry[] entries;
  private int size, used;
  private float loadFactor;
  private final Entry NIL = new Entry(null,null);  // dummy

  public HashTable(int capacity, float loadFactor) {
    entries = new Entry[capacity];
    this.loadFactor = loadFactor;
  }

  public HashTable(int capacity) {
    this(capacity,0.75F);
  }

  public HashTable() {
    this(101);
  }

  public Object get(Object key) {
    int h=hash(key);
    int h2=hash2(key);
    for (int i=0; i<entries.length; i++) {
      int j = nextProbe(h, h2, i);
      Entry entry=entries[j];
      if (entry==null) break;
      if (entry==NIL) continue;
      if (entry.key.equals(key)) return entry.value;  // success
    }
    return null;  // failure: key not found
  }

  public Object put(Object key, Object value) {
    if (used>loadFactor*entries.length) rehash();
    int h=hash(key);
    int h2=hash2(key);
    for (int i=0; i<entries.length; i++) {
      int j = nextProbe(h, h2, i);
      Entry entry=entries[j];
      if (entry==null) {
        entries[j] = new Entry(key,value);
        ++size;
```

```
      ++used;
      return null;  // insertion success
    }
    if (entry==NIL) continue;
    if (entry.key.equals(key)) {
      Object oldValue=entry.value;
      entries[j].value = value;
      return oldValue;  // update success
    }
  }
  return null;  // failure: table overflow
}

public Object remove(Object key) {
  int h=hash(key);
  int h2=hash2(key);
  for (int i=0; i<entries.length; i++) {
    int j = nextProbe(h, h2, i);
    Entry entry=entries[j];
    if (entry==null) break;
    if (entry==NIL) continue;
    if (entry.key.equals(key)) {
      Object oldValue=entry.value;
      entries[j] = NIL;
      --size;
      return oldValue;  // success
    }
  }
  return null;  // failure: key not found
}

public int size() {
  return size;
}

public String toString() {
  StringBuffer buf = new StringBuffer();
  for (int i=0; i<entries.length; i++)
    if (entries[i] != null)
      buf.append(entries[i].key + ": " + entries[i].value + "\n");
  return buf.toString();
}
```

```
  private class Entry {
    Object key, value;
    Entry(Object k, Object v) { key=k; value=v; }
  }

  private int hash(Object key) {
    if (key==null) throw new IllegalArgumentException();
    return (key.hashCode() & 0x7FFFFFFF) % entries.length;
  }

  private int hash2(Object key) {
    if (key==null) throw new IllegalArgumentException();
    return 1 + (key.hashCode() & 0x7FFFFFFF) % (entries.length/2 -
1);
  }

  private int nextProbe(int h, int h2, int i) {
    return (h + h2*i)%entries.length;      // Double Hashing
  }

  private void rehash() {
    Entry[] oldEntries = entries;
    entries = new Entry[2*oldEntries.length+1];
    for (int k=0; k<oldEntries.length; k++) {
      Entry entry=oldEntries[k];
      if (entry==null || entry==NIL) continue;
      int h=hash(entry.key);
      int h2=hash2(entry.key);
      for (int i=0; i<entries.length; i++) {
        int j = nextProbe(h, h2, i);
        if (entries[j]==null) {
          entries[j] = entry;
          break;
        }
      }
    }
    used = size;
  }
}
```

9.7

```
public class Main {
   int m = 17;  // table size
   String[] data = {"AT","BE","DE","DK","ES","FR","GB",
                    "GR","IE","IT","LU","NL","SE"};
   int n = data.length;

   private int hash(String key) {
     return key.hashCode() % m;
   }

   private int lin(int h, int i) {
     return (h + i) % m;
   }

   private int quad(int h, int i) {
     return (h + i*i) % m;
   }

   public Main() {
     System.out.println("Linear Probing:");
     String[] table = new String[m];
     int c=0;
     for (int i=0; i<n; i++) {
       String s = data[i];
       int j=0, h=hash(s), k = h;
      System.out.print((i<9?"  ":" ") + (i+1) + ". " + s +" -> "+h);
       while (table[k] != null && j<m) {
         ++c;
         k = lin(h, ++j);
         System.out.print(" -> " + k);
       }
       table[k] = s;
       System.out.println();
     }
     System.out.println(c + " collisions.");
     System.out.println("Quadratic Probing:");
     table = new String[m];
     c=0;
     for (int i=0; i<n; i++) {
       String s = data[i];
       int j=0, h=hash(s), k = h;
      System.out.print((i<9?"  ":" ") + (i+1) + ". " + s +" -> "+h);
```

```
        while (table[k] != null && j<m) {
          ++c;
          k = quad(h, ++j);
          System.out.print(" -> " + k);
        }
        table[k] = s;
        System.out.println();
      }
      System.out.println(c + " collisions.");
    }

    public static void main(String[] args) {
      new Main();
    }
  }
```

9.8

```
  public class Main {
    int m = 17;  // table size
    String[] data = {"AT","BE","DE","DK","ES","FR","IT","LU","SE"};
    int n = data.length;

    private int hash(String key) {
      return key.hashCode() % m;
    }

    private int hash2(String key) {
      return 1 + key.hashCode() % (m-1);
    }

    private int lin(int h, int i) {
      return (h + i) % m;
    }

    private int doub(int h, int h2, int i) {
      return (h + i*h2) % m;
    }

    public Main() {
      System.out.println("Linear Probing:");
      String[] table = new String[m];
      int c=0;
      for (int i=0; i<n; i++) {
        String s = data[i];
```

```
      int j=0, h=hash(s), k = h;
      System.out.print((i<9?"  ":" ") + (i+1) + ". " + s +" -> "+h);
      while (table[k] != null && j<m) {
        ++c;
        k = lin(h, ++j);
        System.out.print(" -> " + k);
      }
      table[k] = s;
      System.out.println();
    }
    System.out.println(c + " collisions.");
    System.out.println("Double Hashing:");
    table = new String[m];
    c=0;
    for (int i=0; i<n; i++) {
      String s = data[i];
      int j=0, h=hash(s), k = h, h2=hash2(s);
      System.out.print((i<9?"  ":" ") + (i+1) + ". " + s +" -> "+h);
      while (table[k] != null && j<m) {
        ++c;
        k = doub(h, h2, ++j);
        System.out.print(" -> " + k);
      }
      table[k] = s;
      System.out.println();
    }
    System.out.println(c + " collisions.");
  }

  public static void main(String[] args) {
    new Main();
  }
}
```

Chapter 10
Recursion

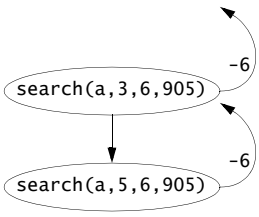
Exercises

10.1 `factorial(8)` makes 7 recursive calls.

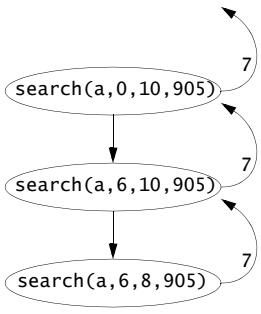
10.2 The top disk is the smallest, so no other disk can be placed on top of it. Step 2 requires all three pegs, but peg B could not be used.

10.3

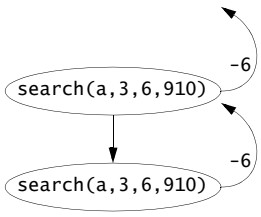
a.



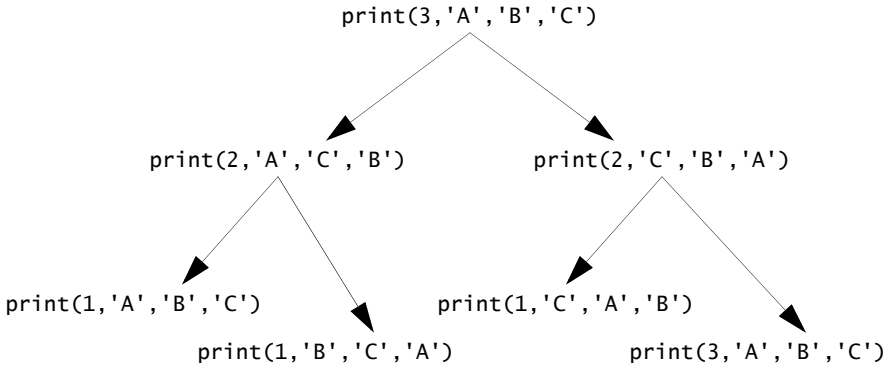
b.



c.



10.4



10.5 **a.** 15.
b. 31.
c. $2^n - 1$.
Let $f(n)$ be the number of moves for n disks, and let $g(n) = 2^n - 1$.
If $n = 1$, it makes 1 move, and $g(1) = 2^1 - 1 = 1$. Thus, $f(1) = g(1)$.
If $n > 1$, then it makes $f(n-1) + 1 + f(n-1)$ moves; *i.e.*, $f(n) = 2f(n-1) + 1$. By the inductive hypothesis, $f(n-1) = g(n-1) = 2^{n-1} - 1$. Thus, $f(n) = 2(2^{n-1} - 1) + 1 = 2^n - 1 = g(n)$.
Thus, $f(n) = g(n)$ for all $n \geq 1$.

10.6 40.

10.7 Let $f(n) = n!$ and let $g(n) = 2^n$.
Then $f(4) = 4! = 24 > 16 = 2^4 = g(4)$.
If $n > 4$, then $f(n) = n! = n(n-1)! = nf(n-1) > ng(n-1)$, by the inductive hypothesis. Thus, $f(n) > ng(n-1) > 2g(n-1)$, since $n > 2$. Thus, $f(n) > 2g(n-1) = 2\,2^{n-1} = 2^n = g(n)$. Thus, $f(n) > g(n)$ for all $n \geq 4$. Thus, $g(n)/f(n) < 1$ for all $n \geq 4$; *i.e.*, $g(n)/f(n)$ is bounded.

10.8 $F_1 + F_2 + F_3 + \cdots + F_{10} = 1 + 1 + 2 + 3 + 5 + 8 + 13 + 21 + 34 + 55 = 143 = 144 - 1 = F_{12} - 1$. More generally:

$$F_0 + F_1 + F_2 + \cdots + F_n = (F_2 - F_1) + (F_3 - F_2) + (F_4 - F_3) + \cdots + (F_{n+2} - F_{n-1})$$

The sum on the right "telescopes", leaving $F_{n+2} - F_1$. Thus:

$$F_0 + F_1 + F_2 + \cdots + F_n = F_{n+2} - F_1 = F_{n+2} - 1$$

10.9 Let $T(n)$ be the run time for the recursive Fibonacci function $f(n)$. Then:

$$\begin{aligned} T(n) &= c + T(n-1) + T(n-2) \\ &> c + 2T(n-2) \\ &> c + 2[c + 2T(n-4)] \\ &= 3c + 4T(n-4) \\ &> 3c + 4[c + 2T(n-2)] \\ &= 7c + 8T(n-6) \\ &\cdots \\ &= (2^k - 1)c + 2^k T(n-2k) \end{aligned}$$

Thus, $T(n) > (2^k - 1)c + 2^k T(n-2k)$, for $k = 1, 2, 3, \ldots$
Let $k = n/2$ so $2k = n$ (assuming n is even). Then $T(n) > (2^{n/2} - 1)c + 2^{n/2} T(0) = \Theta(2^n)$.

10.10

$$\phi = \frac{1+\sqrt{5}}{2}$$

$$\psi = \frac{1-\sqrt{5}}{2}$$

$$\phi + \psi = \frac{1+\sqrt{5}}{2} + \frac{1-\sqrt{5}}{2} = \frac{2}{2} = 1$$

$$\phi - \psi = \frac{1+\sqrt{5}}{2} - \frac{1-\sqrt{5}}{2} = \frac{2\sqrt{5}}{2} = \sqrt{5}$$

$$\phi^2 = \left(\frac{1+\sqrt{5}}{2}\right)^2 = \frac{1+2\sqrt{5}+5}{4} = \frac{6+2\sqrt{5}}{4} = \frac{3+\sqrt{5}}{2} = \frac{1+\sqrt{5}}{2} + \frac{2}{2} = \phi + 1$$

$$\psi^2 = \left(\frac{1-\sqrt{5}}{2}\right)^2 = \frac{1-2\sqrt{5}+5}{4} = \frac{6-2\sqrt{5}}{4} = \frac{3-\sqrt{5}}{2} = \frac{1-\sqrt{5}}{2} + \frac{2}{2} = \psi + 1$$

a.

$$f_n = \frac{\phi^n - \psi^n}{\sqrt{5}}$$

$$f_0 = \frac{\phi^0 - \psi^0}{\sqrt{5}} = \frac{1-1}{\sqrt{5}} = 0$$

$$f_1 = \frac{\phi^1 - \psi^1}{\sqrt{5}} = \frac{\phi - \psi}{\sqrt{5}} = \frac{\sqrt{5}}{\sqrt{5}} = 1$$

$$f_2 = \frac{\phi^2 - \psi^2}{\sqrt{5}} = \frac{(\phi+1) - (\psi+1)}{\sqrt{5}} = \frac{\phi - \psi}{\sqrt{5}} = 1$$

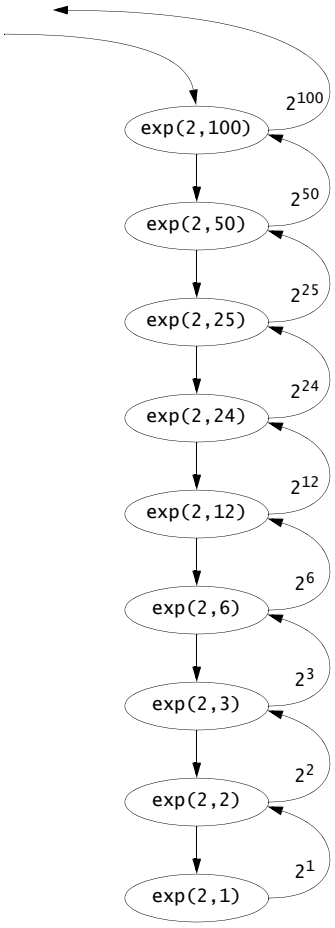
b.

$$\begin{aligned} f_{n-1} + f_{n-2} &= \frac{\phi^{n-1} - \psi^{n-1}}{\sqrt{5}} + \frac{\phi^{n-2} - \psi^{n-2}}{\sqrt{5}} \\ &= \frac{1}{\sqrt{5}}[(\phi^{n-1} + \phi^{n-2}) - (\psi^{n-1} + \psi^{n-2})] \\ &= \frac{1}{\sqrt{5}}[(\phi+1)\phi^{n-2} - (\psi+1)\psi^{n-2}] \\ &= \frac{1}{\sqrt{5}}[(\phi^2)\phi^{n-2} - (\psi^2)\psi^{n-2}] \\ &= \frac{1}{\sqrt{5}}[\phi^n - \psi^n] \\ &= f_n \end{aligned}$$

10.11 On each iteration x is halved. Therefore, the number of iterations that it takes to reduce x from 1.0 to 0.01 is $\lg(1.0/0.01) = \lg(1/0.01)$. More generally, the number of iterations to reduce x to $1/t$ will be $\lg(1/t)$; *i.e.*, the run-time is $T = \Theta(\lg(1/t))$ Thus, if $t' = t^2$, then $T' = \Theta(\lg(1/t')) = \Theta(\lg(1/t^2)) = \Theta(\lg((1/t)^2)) = \Theta(2\lg(1/t)) = 2\Theta(\lg(1/t)) = 2T$.

10.12 If n is even, then $T(n) = c_1 + T(n/2)$, and (since then $n + 1$ will be odd), then $T(n + 1) = c_2 + T(n) = c_2 + c_1 + T(n/2)$. Thus, on average, $T(n) = c + T(n/2)$. The same analysis as on page 314 then applies to obtain $T(n) = \Theta(\lg n)$.

10.13



Programming Problems

10.1

```
public class IterativeFactorial {
  public static void main(String[] args) {
    for (int i=0; i<10; i++)
      System.out.println(i + "\t" + factorial(i));
  }
  static long factorial(int n) {
    if (n<2) return 1;
    long f = n;
    for (int i=2; i<n; i++)
        f *= i;
    return f;
  }
}
```

10.2

```
public class LargestFib {
  private static final double SR5=Math.sqrt(5); // 2.2360679774997896
  private static final double PHI=(1+SR5)/2;    // 1.6180339887498948
  private static final double PSI=(1-SR5)/2;    //-0.6180339887498948
  public static void main(String[] args) {
    long f0 = 0;
    long f1 = 1;
    long f2 = 1;
    for (int i=3; i<93; i++) {
      f0 = f1;
      f1 = f2;
      f2 += f0;
      if (i > 89) System.out.println("\tf(" + i + ") =\t " + f2);
    }
    System.out.println("Long.MAX_VALUE = " + Long.MAX_VALUE);
    System.out.println("\tf(93) =\t" + (f1 + f2));
  }
}
```

10.3

```
import java.math.*;
public class BigFib {
  private static BigDecimal one = new BigDecimal("1");
  private static BigDecimal two = new BigDecimal("2");
  private static BigDecimal five = new BigDecimal("5");
  private static BigDecimal sr5;
  private static BigDecimal phi;
  private static BigDecimal psi;
```

```
  public static void main(String[] args) {
    BigInteger f0 = new BigInteger("0");
    BigInteger f1 = new BigInteger("1");
    BigInteger f2 = new BigInteger("1");
    for (int i=3; i<=1000; i++) {
      f0 = f1;
      f1 = f2;
      f2 = f0.add(f1);
      if (i > 997)
        System.out.println("f(" + i + ") =\t" + f2);
    }
  }
}
```

10.4

```
public class TestBinarySearch {
  TestBinarySearch() {
    int[] a = {22, 33, 44, 55, 66, 77, 88, 99};
    int n = a.length;
    System.out.println("search(" + 55 + "): " + search(a, 0, n, 55));
    System.out.println("search(" + 50 + "): " + search(a, 0, n, 50));
  }
  int search(int[] a, int p, int q, int x) {
    if (q <= p) return -p-1;  // base
    int m = (p + q)/2;
    if (a[m] == x) return m;
    if (a[m] < x) return search(a, m+1, q, x);
    return search(a, p, m, x);
  }
  public static void main(String[] args) {
    new TestBinarySearch();
  }
}
```

10.5

```
public class TestExp {
  TestExp() {
    System.out.println("exp1(3, 4): " + exp1(3, 4));
    System.out.println("exp2(3, 4): " + exp2(3, 4));
    System.out.println("exp1(5, -2): " + exp1(5, -2));
    System.out.println("exp2(5, -2): " + exp2(5, -2));
    System.out.println("exp1(2, -5): " + exp1(2, -5));
    System.out.println("exp2(2, -5): " + exp2(2, -5));
  }
```

```
  double exp1(double x, int n) {
    if (n == 0) return 1.0;
    if (n < 0) return 1.0/exp1(x, -n);
    double y = 1.0;
    for (int i = 0; i < n; i++)
      y *= x;
    return y;
  }
  double exp2(double x, int n) {
    if (n == 0) return 1.0;
    if (n < 0) return 1.0/exp2(x, -n);
    double factor = (n%2 == 0 ? 1.0 : x);
    if (n < 2) return factor;       // base
    return factor*exp2(x*x, n/2);    // recursion
  }
  public static void main(String[] args) {
    new TestExp();
  }
}
```

10.6

```
public double valueAt(double x) {
  if (this.equals(ZERO)) return 0.0;
  double y = 0.0;
  int e = exp;
  Poly p = this;
  while (p != null) {
    for (int i=1; i < e - p.exp; i++)
      y *= x;
    y += p.coef;
    e = p.exp;
    p = p.next;
  }
  return y;
}
```

10.7

```
String commaString(long n) {
  String s="";
  int c = 0;
  while (n > 0) {
    if (c>0 && c%3==0) s = "," + s;
    s = (n%10) + s;
    n /= 10;
    ++c;
  }
```

```
  return s;
}
String commaString(long n) {
  String s="";
  if (n==0) return "0";
  for (int i=0; i<3; i++) {
    s = (n%10) + s;
    n /= 10;
    if (n==0) return s;
  }
  return commaString(n) + "," + s;  // recursion
}
```

10.8

```
int max(int[] a, int p, int q) {
  if (q - p == 1) return a[p];
  int m = (p + q)/2;
  int max1 = max(a, p, m);  // recursion
  int max2 = max(a, m, q);  // recursion
  return ( max1 > max2 ? max1 : max2 );
}
```

10.9

```
int intBinLog(int n) {
  if (n < 2) return 0;
  return 1 + intBinLog(n/2);  // recursion
}
```

10.10

```
boolean isPalindrome(String s) {
  int n = s.length();
  if (n < 2) return true;
  if (s.charAt(0) != s.charAt(n-1)) return false;
  return (isPalindrome(s.substring(1,n-1)));  // recursion
}
```

10.11

```
String bin(int n) {
  if (n == 0) return "0";
  if (n == 1) return "1";
  return bin(n/2) + bin(n%2);  // recursion
}
```

10.12

```
String hex(int n) {
  if (n < 10) return String.valueOf(n) ;
  if (n < 16) {
    switch (n) {
      case 10: return "A";
      case 11: return "B";
      case 12: return "C";
      case 13: return "D";
```

```
        case 14: return "E";
        case 15: return "F";
      }
    }
    return hex(n/16) + hex(n%16);  // recursion
  }
```

10.13
```
  String reverse(String s) {
    if (s.length() < 2) return new String(s);
    return reverse(s.substring(1)) + s.charAt(0);  // recursion
  }
```

10.14 The computable domain for the `factorial()` method was $0 \le n \le 20$.

10.15 The computable domain for the `fibonacci()` method was $0 \le n \le 48$.

10.16 The computable domain for the Ackermann function is:

$0 \le n <$ `Long.MAX_VALUE`, for $m = 0$;
$0 \le n <$ `Long.MAX_VALUE - 1`, for $m = 1$;
$0 \le n <$ `Long.MAX_VALUE/2 - 1`, for $m = 2$;
$0 \le n \le 60$, for $m = 3$;
$n = 0$, for all $m > 3$.

10.17
```
  public class TestTrigFunctions {
    static final double TOL = 0.00005;
    public static void main(String[] args) {
      for (double x=0.0; x<1.0; x+=0.1)
        System.out.println(sin(x) + "\t" + mathSin(x));
      for (double x=0.0; x<1.0; x+=0.1)
        System.out.println(cos(x) + "\t" + mathCos(x));
    }

    static double sin(double x) {
      if (-TOL<x && x<TOL) return x - x*x*x/6;  // base
      return 2*sin(x/2)*cos(x/2);               // recursion
    }

    static double cos(double x) {
      if (-TOL<x && x<TOL) return 1.0 - x*x/2;  // base
      double y = sin(x/2);                      // recursion
      return 1 - 2*y*y;
    }

    static double mathSin(double x) {
      return Math.sin(x);
    }
```

```
  static double mathCos(double x) {
    return Math.cos(x);
  }
}
```

10.18

```
public class TestTrigFunctions {
  public static void main(String[] args) {
    for (double x=0.0; x<1.0; x+=0.1)
      System.out.println(sin(x) + "\t" + mathSin(x));
    for (double x=0.0; x<1.0; x+=0.1)
      System.out.println(cos(x) + "\t" + mathCos(x));
  }

  static double sin(double x) {
    if (-0.01<x && x<0.01) return x*(1 - x*x/6*(1 - x*x/20));  // base
    return 2*sin(x/2)*cos(x/2);                               // recursion
  }

  static double cos(double x) {
    if (-0.01<x && x<0.01) return 1 - x*x/2*(1 - x*x/12);  // base
    double y = sin(x/2);                                   // recursion
    return 1 - 2*y*y;
  }

  static double mathSin(double x) {
    return Math.sin(x);
  }
  static double mathCos(double x) {
    return Math.cos(x);
  }
}
```

10.19

```
public class TestHyperbolicFunctions {
  public static void main(String[] args) {
    for (double x=0.0; x<1.0; x+=0.1)
      System.out.println(sinh(x) + "\t" + mathSinh(x));
    for (double x=0.0; x<1.0; x+=0.1)
      System.out.println(cosh(x) + "\t" + mathCosh(x));
  }

  static double sinh(double x) {
    if (-0.01<x && x<0.01) return x + x*x*x/6;  // base
    return 2*sinh(x/2)*cosh(x/2);               // recursion
  }
```

```
  static double cosh(double x) {
    if (-0.01<x && x<0.01) return 1.0 + x*x/2;  // base
    return 1 + 2*sinh(x/2)*sinh(x/2);           // recursion
  }

  static double mathSinh(double x) {
    return (Math.exp(x) - Math.exp(-x))/2;
  }
  static double mathCosh(double x) {
    return (Math.exp(x) + Math.exp(-x))/2;
  }
}
```

10.20

```
static double tan(double x) {
  if (-0.01<x && x<0.01) return x + x*x*x/3;  // base
  double t = tan(x/2);
  return 2*t/(1.0 - t*t);               // recursion
}
```

10.21

```
public class Timing {
  static double tol = 0.01;

  public static void main(String[] args) {
    System.out.println(time(1e-2));
    System.out.println(time(1e-3));
    System.out.println(time(1e-4));
    System.out.println(time(1e-5));
    System.out.println(time(1e-6));
    System.out.println(time(1e-7));
  }

  static long time(double t) {
    long t1 = System.currentTimeMillis();
    for (double x=0.0; x<1.0; x+=0.00001)
      sin(x);
    for (double x=0.0; x<1.0; x+=0.00001)
      cos(x);
    long t2 = System.currentTimeMillis();
    return t2 - t1;
  }
```

```
  static double sin(double x) {
    if (-tol<x && x<tol) return x - x*x*x/6;  // base
    return 2*sin(x/2)*cos(x/2);               // recursion
  }

  static double cos(double x) {
    if (-tol<x && x<tol) return 1.0 - x*x/2;  // base
    double s = sin(x/2);
    return 1 - 2*s*s;
  }
}
```

Chapter 11
Trees

Exercises

11.1

a. True. A node is an ancestor if and only if it lies on the root-path.

b. False. Its size is one more than the number of its descendants.

c. True. If x is a descendant of y, then x's root-path contains y's root-path.

d. False. x could be a nephew of y.

e. True. If the root is a leaf, then the tree has only one node.

f. True. Each leaf of the subtree is an element of the supertree and has no children.

g. False. If one tree is a subtree of another and if the two roots are the same node, then the two trees must be equal.

h. True. The set of all leaf-paths from x in R is the same as in S and in T.

i. True. A leaf is a node with no children.

j. False. A tree of height 2 with only one leaf (*i.e.*, a linear tree) has two internal nodes.

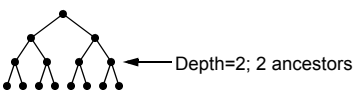
k. False. A tree of height 1 with four nodes is not full.

l. True. If a leaf is omitted, then all ancestors of that leaf must also be omitted.

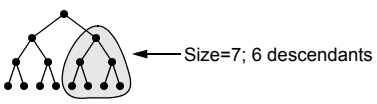
m. True. If a leaf is omitted, then all ancestors of that leaf must also be omitted.

11.2

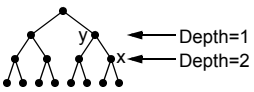
a.



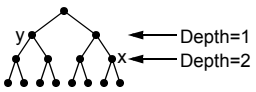
b.



c.

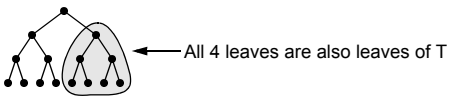


d.

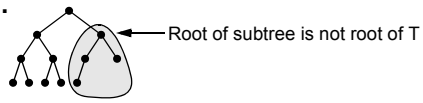


e.

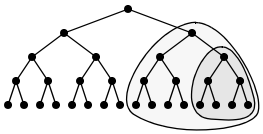
f.



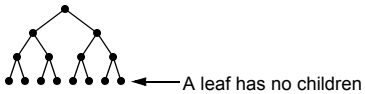
g.



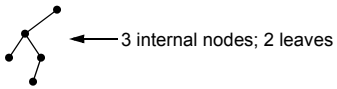
h.



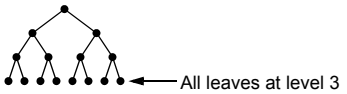
i.



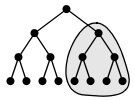
j.



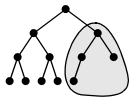
k.



l.



m.

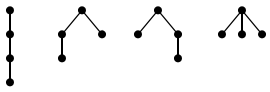


11.3 **a.** Unordered; e.g., the order of {`float`, `double`} is irrelevant.
b. 4
c. 5
d. 7
e. 46

11.4

11.5

11.6



11.7 `final`

11.8 **a.** C, F, H, J, L, M, N, O, P
b. G
c. 2
d. 4
e. D, A
f. J, K, L, M, O, P
g. H, I, J, K, L, M
h. 4
i. 6
j. 4

11.9
a. 129
b. 85
c. 1,111
d. 1,398,101

11.10

$$n = \frac{d^{h+1}-1}{d-1}$$

$$n(d-1) = d^{h+1}-1$$

$$nd-n+1 = d^{h+1}$$

$$\log_d(nd-n+1) = h+1$$

$$h = \log_d(nd-n+1)-1$$

11.11 w = number of leaves = d^h

11.12 path length = $0 \cdot 1 + 1 \cdot d + 2 \cdot d^2 + 3 \cdot d^3 + \cdots + h \cdot d^h =$

$$\sum_{j=0}^{h} jd^j = \sum_{j=1}^{h}\sum_{i=1}^{j} jd^j$$

$$= \sum_{i=1}^{h}\sum_{j=i}^{j} d^j$$

$$= \sum_{i=1}^{h} d^i \sum_{j=i}^{j} d^{j-i}$$

$$= \sum_{i=1}^{h} d^i \sum_{k=0}^{h-i} d^k$$

$$= \sum_{i=1}^{h} d^i \frac{d^{h-i+1}-1}{d-1}$$

$$= (d-1)^{-1}\sum_{i=1}^{h}(d^{h+1}-d^i)$$

$$= (d-1)^{-1}\left(hd^{h+1}-\left(\frac{d^{h+1}-1}{d-1}-1\right)\right)$$

$$= (d-1)^{-2}((d-1)hd^{h+1}-((d^{h+1}-1)+(d-1)))$$

$$= \frac{hd^{h+2}-(h+1)d^{h+1}+d}{(d-1)^2}$$

11.13 **a.** A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P
b. A, B, E, H, I, N, F, C, D, G, J, K, O, P, L, M
c. H, N, I, E, F, B, C, J, O, P, K, L, M, G, D, A

11.14 **a.** level order and preorder
b. postorder
c. postorder
d. preorder

11.15 **Algorithm. Iterative Preorder Traversal**

1. Create an empty stack.
2. Push the tree (root) onto the stack.
3. While the stack is not empty, do steps 4-6:
4. Pop the stack to obtain x.
5. Visit the root of x.
6. Traverse the subtree list of x in reverse order, pushing each subtree onto the stack

11.16 Let x by any internal node of a full tree of degree d. Let l be the level of node x, and let y be the left-most node at that same level. Then, by Formula 11.4, the index of y is $i(y) = (d^l - 1)/(d - 1)$, and the index of the first child y' of y is $i(y') = (d^{l+1} - 1)/(d - 1)$.

Let k be the number of nodes on the left of x at the same level l. Then x has index $i(x) = (d^l - 1)/(d - 1) + k$. Since the tree is full, each node at level l had d children. Thus, there are dk children on the left of the first child of x (at level l+1). Thus, the first child x' of x has index $i(x') = (d^{l+1} - 1)/(d - 1) + dk = (d^{l+1} - 1 + d^2k - dk)/(d - 1)$. This is the same as $d \cdot i(x) + 1 = d[(d^l - 1)/(d - 1) + k] + 1 = (d^{l+1} - 1 + d^2k - dk)/(d - 1)$. Thus the index of the first child of x is $d \cdot i(x) + 1$. Thus the indexes of all d children of x are $d \cdot i(x) + 1$, $d \cdot i(x) + 2$, $d \cdot i(x) + 3$, ..., $d \cdot i(x) + d$.

Programming Problems

11.1

```
import java.util.*;

public class OrderedTree {
  private Object root;
  private List subtrees;
  private int size;

  public OrderedTree() {  // constructs the empty tree
  }
  public OrderedTree(Object root) {  // constructs a singleton
    this.root = root;
    subtrees = new LinkedList();  // constructs the empty list
    size = 1;
  }
  public OrderedTree(Object root, List trees) {
    this(root);
    for (Iterator it=trees.iterator(); it.hasNext(); ) {
      Object object=it.next();
      if (object instanceof OrderedTree) {
        OrderedTree tree = (OrderedTree)object;
        subtrees.add(tree);
        size += tree.size();
      }
    }
  }
  public int size() {
    return size;
  }
}
```

11.2

```
public void levelOrderPrint() {
  LinkedList queue = new LinkedList();
  queue.addLast(this);
  while (!queue.isEmpty()) {
    OrderedTree tree = (OrderedTree)queue.removeFirst();
    System.out.print(tree.root + " ");
    for (Iterator it=tree.subtrees.iterator(); it.hasNext(); )
      queue.addLast(it.next());
  }
}
```

11.3

```
public void preorderPrint() {
  System.out.print(root + " ");
  for (Iterator it=subtrees.iterator(); it.hasNext(); )
    ((OrderedTree)it.next()).preorderPrint();
}
```

11.4

```
public void postorderPrint() {
  for (Iterator it=subtrees.iterator(); it.hasNext(); )
    ((OrderedTree)it.next()).postorderPrint();
  System.out.print(root + " ");
}
```

11.5

```
public void preorderPrint() {
  if (root == null) return;
  Stack stack = new Stack();
  stack.push(this);
  while (!stack.isEmpty()) {
    OrderedTree x = (OrderedTree)stack.pop();
    System.out.print(x.root + " ");
    List s = x.subtrees;
    ListIterator lit = s.listIterator(s.size());
    while (lit.hasPrevious())
      stack.push(lit.previous());
  }
}
```

Chapter 12
Binary Trees

Exercises

12.1 Let T be a binary tree of height h and size n. Having height h means that it must contain at least one root-to-leaf path of length h. Any such path has $h + 1$ elements. Thus, $n \geq h + 1$. By Formula 12.1, the *full* binary tree of height h has $2^{h+1} - 1$ elements. If T is not full, then it can be transformed into that full binary tree by adding elements. Thus, $n \leq 2^{h+1} - 1$.

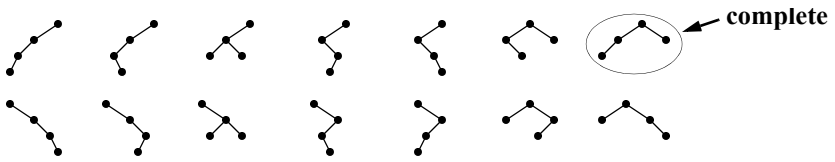
12.2 By Formula 12.2, we have $n \leq 2^{h+1} - 1$. Thus $n < n - 1 \leq 2^{h+1}$, so $\lg n < h + 1$. Since $\lfloor x \rfloor \leq x$ (see Formula C.1.3 on page 565), we have $\lfloor \lg n \rfloor \leq \lg n$. Thus, $\lfloor \lg n \rfloor < h + 1$. But both $\lfloor \lg n \rfloor$ and h are integers. Thus, $\lfloor \lg n \rfloor \leq h$.

12.3 Let T be a *complete* binary tree of height h and size n. By Formula 12.1, $n \leq 2^{h+1} - 1$. Moreover, T contains the full binary tree of height $h - 1$ because T includes all the elements of the full binary tree of height h except possibly for some missing leaf elements at level h. Therefore, $n \geq 2^{(h-1)+1} - 1 = 2^h - 1$. But by having height h, T must include at least one element at level h, which means that it must have at least one element more than the complete binary tree of height $h - 1$. Thus, $n > 2^h - 1$. Therefore, $n \geq 2^h$ (since n and 2^h are integers).

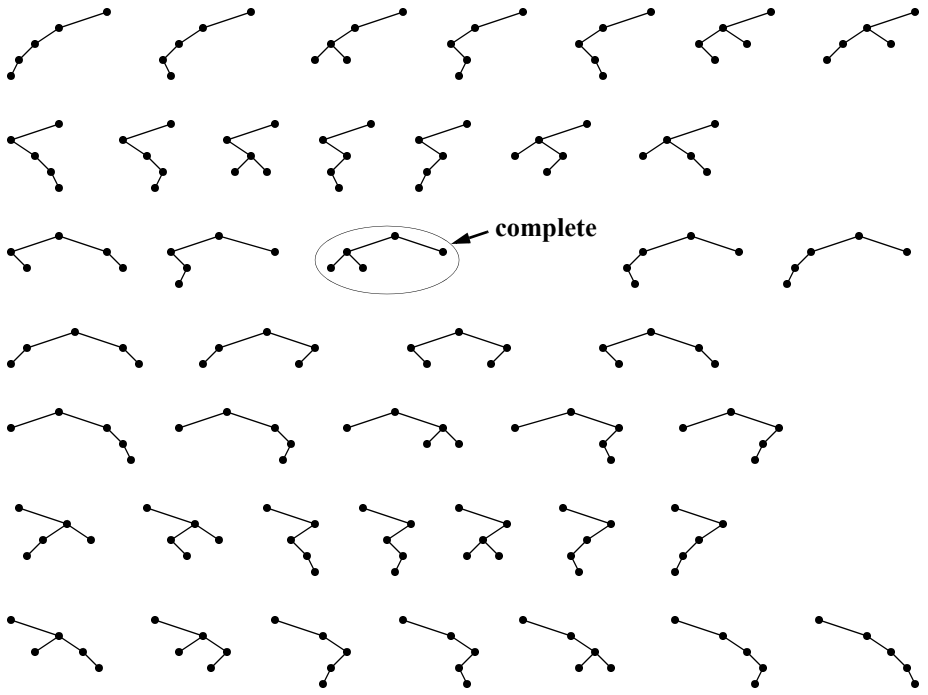
12.4 Let T be a *complete* binary tree of height h and size n. Then T includes all the elements of the full binary tree of height h except possibly for some missing leaf elements at level h. Therefore, T has more elements than the full binary tree of height $h - 1$, but not more than the full binary tree of height h. Thus, by Formula 12.1, $2^h = 2^{(h-1)+1} < n \leq 2^{h+1} - 1$. Thus, $h < \lg n < h + 1$. Therefore, by Formula C.1.4 on page 565, we have $h = \lfloor \lg n \rfloor$.

12.5 Let n be the number of nodes at lebel l in a binary tree. If $l = 0$, then $n = 1$, since the only node at that level is the root node. If $l = 1$, then $n = 1$ or 2, since the only nodes at that level are the children of the root, and in a binary tree each node has at most 2 children. Similary, at any level l, n is the number of children of the nodes at level $l - 1$. By the induction hypothesis, there are between 1 and 2^{l-1} nodes at level $l - 1$. Therefore, n must be between 1 and $2(2^{l-1}) = 2^l$.

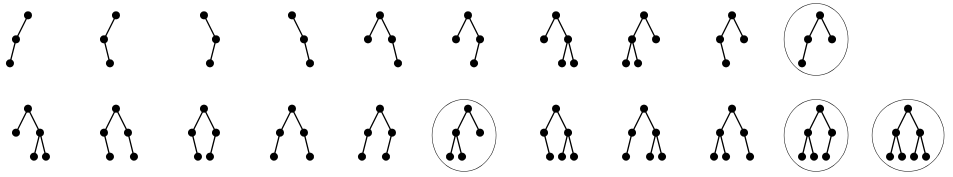
12.6



12.7



12.8

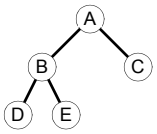


12.9 There are 16 complete binary trees with height $h = 4$.

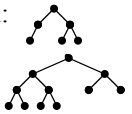
12.10 There are $C_6 = 132$ binary trees with size $n = 6$. [See Section C.7.]

12.11 There are $C_7 = 429$ binary trees with size $n = 7$. [See Section C.7.]

12.12 Label the subtrees "A", "B", *etc.*, according to their root labels:
Subtrees D, E, and C are (valid) binary trees because they are singletons. Then subtree B is a binary tree because it is a triple (x, L, R), where x is the node (labeled "B"), and L and R are the binary trees D and E. Then subtree A is a binary tree because it is a triple (x', L', R'), where x is the node (labeled "A"), and L' and R' are the binary trees B and C.



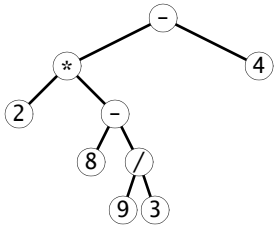
12.13 False. All three leaves of this binary tree are at the same level, but it is not full:



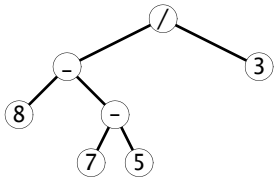
12.14 False. Every proper subtree of this binary tree is full, but the tree itself is not:

12.15
- **a.** Level order: A B C D E F G H I J K L M N O P Q R S T U V
- **b.** Preorder: A B D H E I J N O S C F K P T U G L M O V R
- **c.** Inorder: D H B I E N J S O A F T P U K C L G V O M R
- **d.** Postorder: H D I N S O J E B T U P K F L V O R M G C A

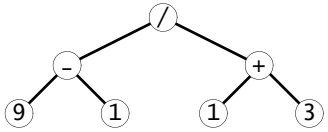
12.16 Prefix: `- * 2 - 8 / 9 3 4`
Infix: `2 * 8 - 9 / 3 - 4`
Postfix: `2 8 9 3 / - * 4 -`
Value: `6`



12.17 Postfix: `8 7 5 - - 3 /`
Infix: `8 - 7 - 5 / 3`
`(8 - (7 - 5)) / 3`
Value: `2`



12.18 Prefix: `/ - 0 1 + 1 3`
Infix: `9 - 1 / 1 + 3`
`(9 - 1) / ( 1 + 3)`
Value: `2`

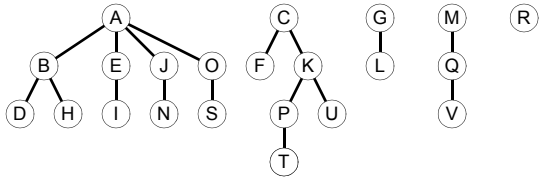


12.19 Algorithm: Evaluating an Expression from Its Prefix Representation

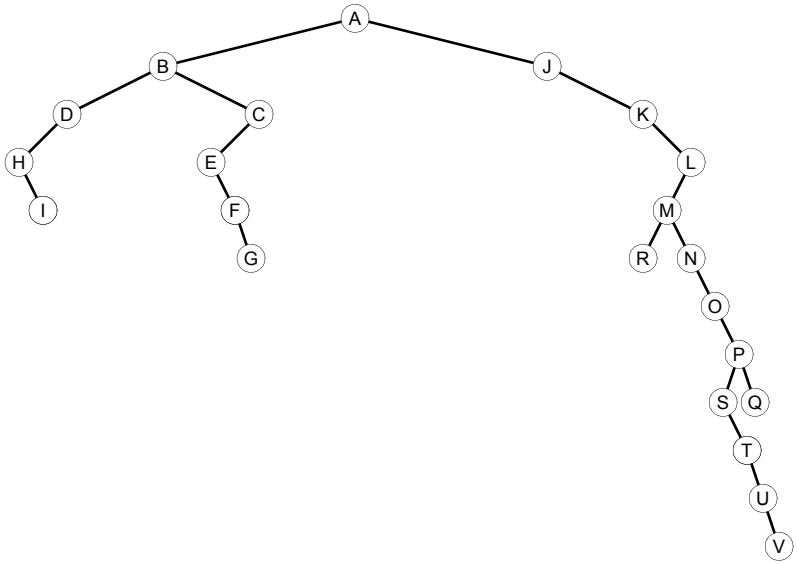
1. Create a stack for operators and operands.
2. Read the first token into x.
3. Repeat steps 4-9 until both the input and the stack are empty:
4. If x and the stack top both are operands, do steps 5-7:
5. Pop y from the stack.
6. Pop z from the stack.
7. Let x be the result of applying the operator z to y and x.
8. Otherwise, push x onto the stack and read the next token into x.

12.20 Let T be complete binary tree of size n, and let p be the number of the first leaf and q be the number of its last leaf. (For example, the tree in Figure 12.13 has $n = 11$, $p = 5$ and $q = 10$.) Then clearly, $q = n - 1$. Moreover, the parent of the last leaf is the last internal node of the complete tree. By Formula 12.6, that internal node has number $(q - 1)/2$. The node to its right is the first leaf node, so $p = (q - 1)/2 + 1 = (q + 1)/2 = n/2$.

12.21



12.22



Programming Problems

12.1

```
public class BinaryTree {
   private Object root;
   private BinaryTree left, right;

  public BinaryTree(Object root) {
    this.root = root;
  }

  public BinaryTree(Object root, BinaryTree left, BinaryTree right) {
    this(root);
    if (left!=null) this.left = left;
    if (right!=null) this.right = right;
  }

  public BinaryTree(BinaryTree that) {
    this.root = that.root;
    if (that.left!=null) this.left = new BinaryTree(that.left);
    if (that.right!=null) this.right = new BinaryTree(that.right);
  }

  public Object getRoot() {
    return this.root;
  }

  public BinaryTree getLeft() {
    return this.left;
  }

  public BinaryTree getRight() {
    return this.right;
  }

  public Object setRoot(Object object) {
    Object oldRoot = this.root;
    this.root = object;
    return oldRoot;
  }

  public BinaryTree setLeft(BinaryTree tree) {
    BinaryTree oldLeft = this.left;
```

```
    this.left = tree;
    return oldLeft;
  }

  public BinaryTree setRight(BinaryTree tree) {
    BinaryTree oldRight = this.right;
    this.right = tree;
    return oldRight;
  }
}
```

12.2

```
public String toString() {
  StringBuffer buf = new StringBuffer("[");
  if (left!=null) buf.append(left + ",");
  buf.append(root);
  if (right!=null) buf.append("," + right);
  return buf + "]";
}
```

12.3

```
public boolean isLeaf() {
  return left==null && right==null;
}
```

12.4

```
public int size() {
  if (left==null && right==null) return 1;
  if (left==null) return 1 + right.size();
  if (right==null) return 1 + left.size();
  return 1 + left.size() + right.size();
}
```

12.5

```
public int height() {
  if (left==null && right==null) return 0;
  if (left==null) return 1 + right.height();
  if (right==null) return 1 + left.height();
  return 1 + Math.max(left.height(), right.height());
}
```

12.6

```
public boolean contains(Object object) {
  if (this.root==object) return true;
  if (left!=null && left.contains(object)) return true;
  if (right!=null && right.contains(object)) return true;
  return false;
}
```

12.7

```
public int numLeaves() {
  if (left==null && right==null) return 1;
  if (left==null) return right.numLeaves();
  if (right==null) return left.numLeaves();
```

```
    return left.numLeaves() + right.numLeaves();
  }
```

12.8

```
  public int frequency(Object x) {
    int frequency = ( x.equals(root) ? 1 : 0 );
    if (left!=null) frequency += left.frequency(x);
    if (right!=null) frequency += right.frequency(x);
    return frequency;
  }
```

12.9

```
  public boolean isFull() {
    if (left==null && right==null) return true;
    if (left==null || right==null) return false;
    return left.isFull() && right.isFull()
        && left.height()==right.height();
  }
```

12.10

```
  public boolean isBalanced() {
    if (left==null && right==null) return true;
    if (left==null) return right.isLeaf();
    if (right==null) return left.isLeaf();
    int balance = left.height() - right.height();
    return balance > -2 && balance < 2;
  }
```

12.11

```
  public int pathLength() {
    if (left==null && right==null) return 0;
    if (left==null) return right.pathLength() + right.size();
    if (right==null) return left.pathLength() + left.size();
    return left.pathLength() + right.pathLength() + size() - 1;
  }
```

12.12

```
  public BinaryTree reverse() {
    if (left==null && right==null)
      return new BinaryTree(this.root);
    if (left==null)
      return new BinaryTree(this.root,right.reverse(),null);
    if (right==null)
      return new BinaryTree(this.root,null,left.reverse());
    return new BinaryTree(this.root,right.reverse(),left.reverse());
  }
```

12.13

```
  public int level(Object object) {
    if (this.root==object) return 0;
    int levelInLeft = ( left==null ? -1 : left.level(object) );
    if (levelInLeft>-1) return 1 + levelInLeft;
    int levelInRight = ( right==null ? -1 : right.level(object) );
    if (levelInRight>-1) return 1 + levelInRight;
```

```
  return -1;
}
```

12.14

```
public boolean isDisjointFrom(BinaryTree that) {
  if (that==null) return true;
  if (this.contains(that.root)) return false;
  return this.isDisjointFrom(that.left)
      && this.isDisjointFrom(that.right);
}
```

12.15

```
public boolean isValid() {
  if (left==null && right==null) return true;
  if (left==null)  // so right!=null
    if (right.contains(this.root) || !right.isValid())
      return false;
  if (right==null)  // so left!=null
    if (left.contains(this.root) || !left.isValid())
      return false;
  return left.isDisjointFrom(right);
}
```

12.16

```
public boolean equals(Object object) {
  if (this == object) return true;
  if (!(object instanceof BinaryTree)) return false;
  BinaryTree that = (BinaryTree)object;
  if (!this.root.equals(that.root)) return false;
  boolean leftEqual = this.left==null && that.left==null
    || this.left!=null && this.left.equals(that.left);
  boolean rightEqual = this.right==null && that.right==null
    || this.right!=null && this.right.equals(that.right);
  return leftEqual && rightEqual;
}
```

12.17

```
public void preorderPrint() {
  if (left!=null) left.preorderPrint();
  System.out.print(root + " ");
  if (right!=null) right.preorderPrint();
}
public void inorderPrint() {
  if (left!=null) left.inorderPrint();
  System.out.print(root + " ");
  if (right!=null) right.inorderPrint();
}
public void postorderPrint() {
  if (left!=null) left.postorderPrint();
  if (right!=null) right.postorderPrint();
```

```
    System.out.print(root + " ");
  }
```

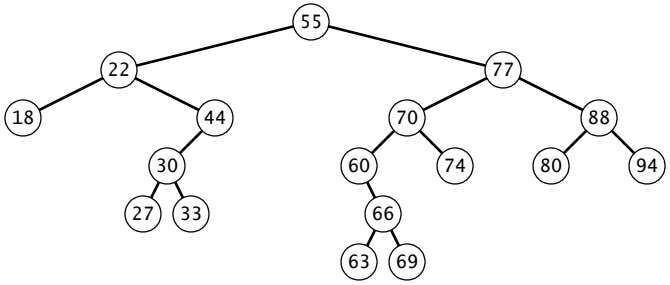
12.18

```
  public void levelOrderPrint() {
    java.util.List queue = new java.util.LinkedList();
    queue.add(this);
    while (!queue.isEmpty()) {
      BinaryTree tree = (BinaryTree)queue.remove(0);
      System.out.print(tree.root + " ");
      if (tree.left != null) queue.add(tree.left);
      if (tree.right != null) queue.add(tree.right);
    }
  }
```

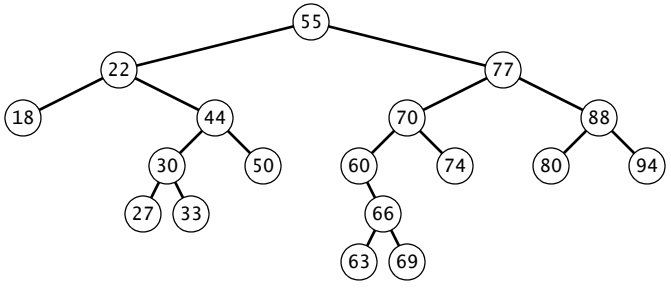
Chapter 13
Lists

Exercises

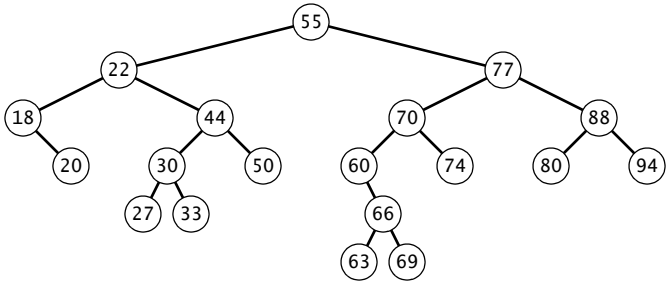
13.1


55
22
77
18
44
70
88
30
60
74
80
94
27
33
66
63
69

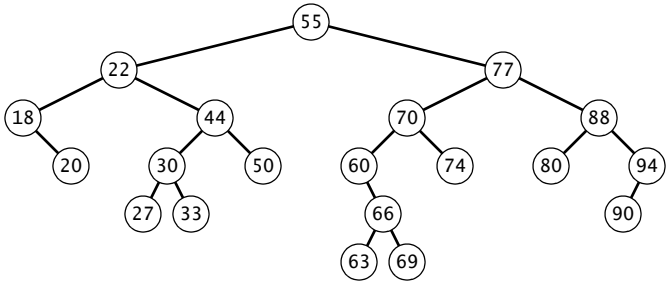
50


55
22
77
18
44
70
88
30
50
60
74
80
94
27
33
66
63
69

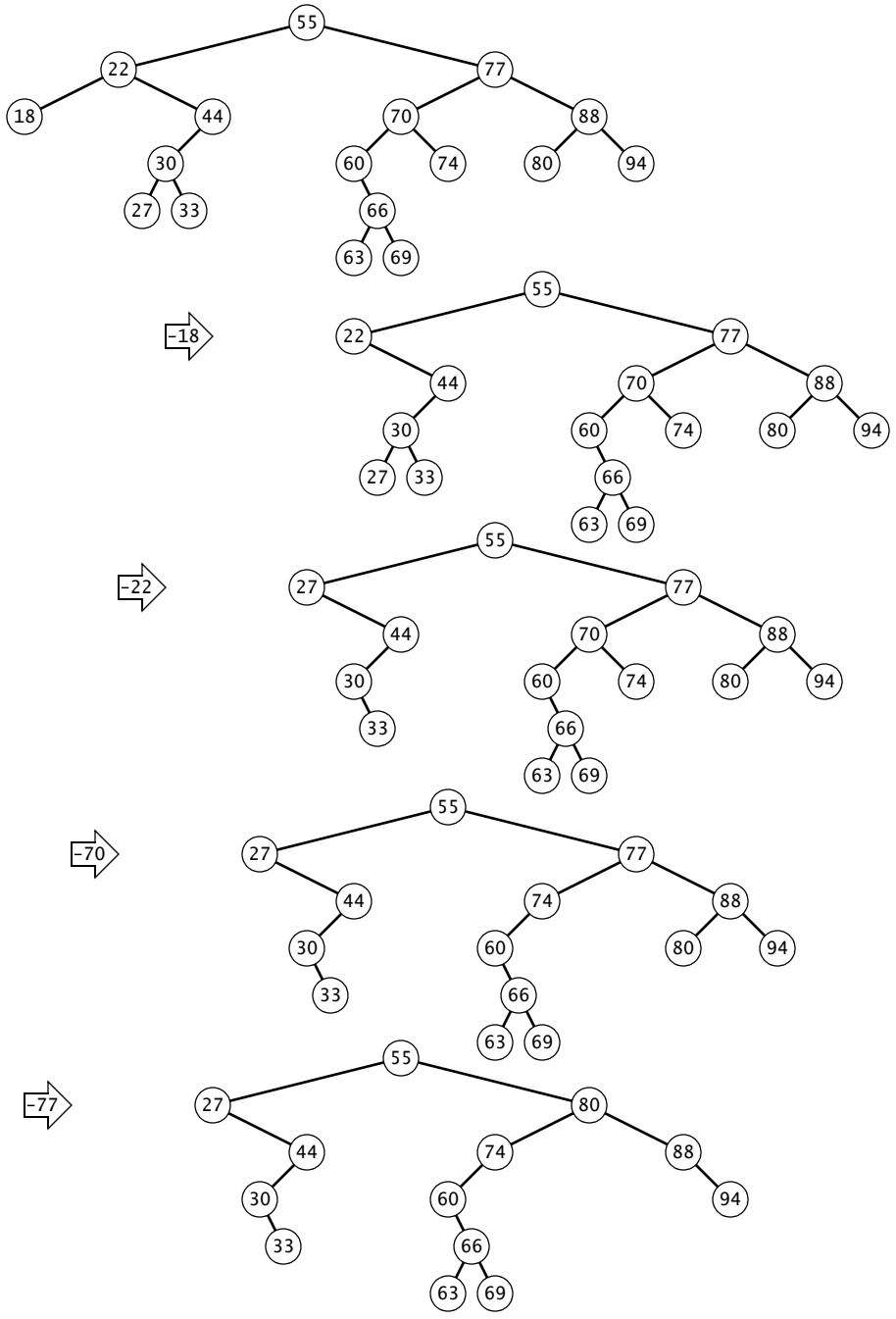
20


55
22
77
18
44
70
88
20
30
50
60
74
80
94
27
33
66
63
69

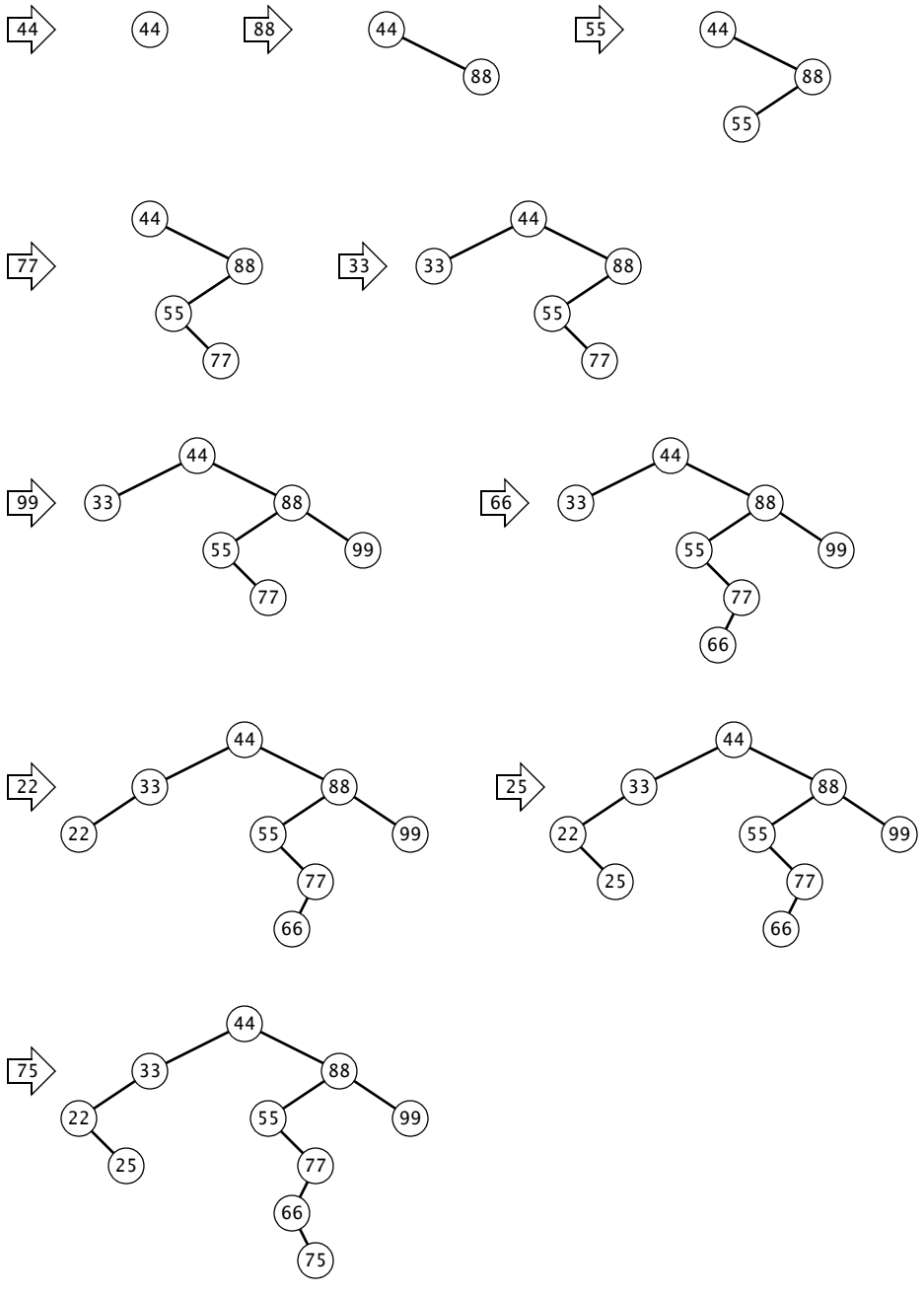
90


55
22
77
18
44
70
88
20
30
50
60
74
80
94
27
33
66
90
63
69

13.2


55
22
77
18
44
70
88
30
60
74
80
94
27
33
66
63
69
-18
55
22
77
44
70
88
30
60
74
80
94
27
33
66
63
69
-22
55
27
77
44
70
88
30
60
74
80
94
33
66
63
69
-70
55
27
77
44
74
88
30
60
80
94
33
66
63
69
-77
55
27
80
44
74
88
30
60
94
33
66
63
69

13.3


44
44
88
44
88
55
44
88
55
77
44
88
55
77
33
44
33
88
55
77
99
44
33
88
55
99
77
66
22
44
33
88
22
55
99
77
66
25
44
33
88
22
55
99
25
77
66
75
44
33
88
22
55
99
25
77
66
75

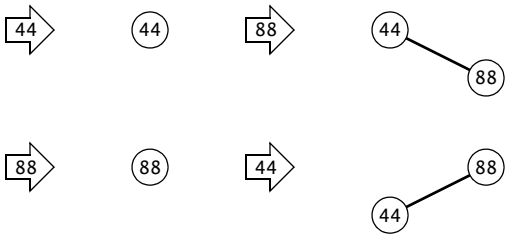
13.4 If y is the inorder successor of x in a BST, then the result of deleting x and y is independent of the order in which they are deleted.

13.5

a. True: the BST property holds for all the elements of the subtree.

b. False: the root key may not be less than all the keys in the right subtree.

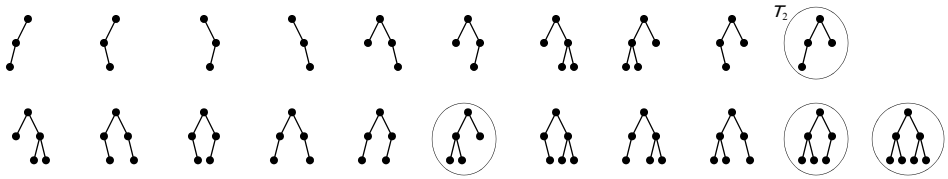
c. False: here are the two (different) BSTs resulting from inserting 44 and 88 in different orders:



d. True: the balance number of a node depends entirely on the heights of its two subtrees.

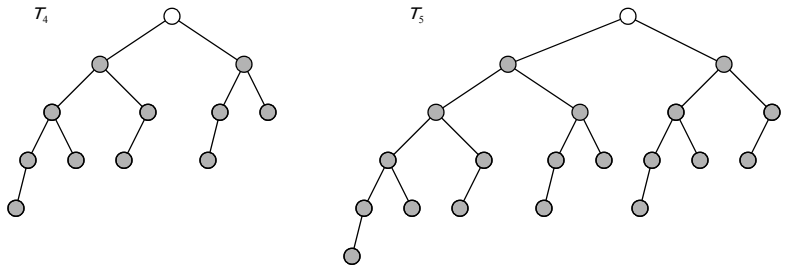
e. False: the two subtrees could have heights that differ by more than 1.

13.6



13.7 There is 1 AVL tree of height 1 and 3 AVL trees of height 2. So the number of AVL trees of height 3 is $3 \cdot 3 + 2(3 \cdot 1) = 15$. Thus, the number of AVL trees of height 4 is $15 \cdot 15 + 2(15 \cdot 3) = 315$.

13.8



13.9 If $h = 0$, $n = |T_0| = 1$, which is between $1.5^0 = 1$ and $2^0 = 1$. If $h = 1$, $n = |T_1| = 2$, which is between $1.5^1 = 1.5$ and $2^1 = 2$. If $h = 2$, $n = |T_2| = 4$, which is between $1.5^2 = 2.25$ and $2^2 = 4$. Assume the formula holds for all Fibonacci trees of height less than h. Then:

$$1.5^{h-2} \le |T_{h-2}| \le 2^{h-2}$$

$$1.5^{h-1} \le |T_{h-1}| \le 2^{h-1}$$

$$|T_h| = |T_{h-1}| + |T_{h-2}| + 1$$

$$\le 2^{h-1} + 2^{h-2} + 1$$

$$< 2^{h-1} + 2^{h-2} + 2^{h-2} = 2^h$$

$$|T_h| \ge 1.5^{h-1} + 1.5^{h-2} + 1$$

$$> 1.5^{h-1} + 1.5^{h-2} = 2.5(1.5^{h-2})$$

$$> 2.25(1.5^{h-2}) = 1.5^h$$

13.10

$$A(n) = \frac{1}{n}\sum_{i=0}^{n-1} A_i(n)$$

$$= \frac{1}{n}\sum_{i=0}^{n-1}\left[\frac{1}{n} + \frac{i}{n}[1 + A(i)] + \frac{n-i-1}{n}[1 + A(n-i-1)]\right]$$

$$= \frac{1}{n^2}\left[\sum_{i=0}^{n-1} 1 + \sum_{i=0}^{n-1} i[1 + A(i)] + \sum_{i=0}^{n-1}(n-i-1)[1 + A(n-i-1)]\right]$$

$$= \frac{1}{n^2}\left[n + 2\sum_{i=0}^{n-1} i[1 + A(i)]\right]$$

$$= \frac{1}{n} + \frac{2}{n^2}\left[\sum_{i=0}^{n-1} i + \sum_{i=0}^{n-1} i\, A(i)\right]$$

$$= \frac{1}{n} + \frac{2}{n^2}\left[\frac{(n-1)n}{2} + \sum_{i=0}^{n-1} i\, A(i)\right]$$

$$= 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} i\, A(i)$$

13.11

$$A(n) = 1 + \frac{2}{n^2}\sum_{i=0}^{n-1} i\,A(i)$$

$$n^2A(n) = n^2 + 2\sum_{i=0}^{n-1} i\,A(i)$$

$$(n+1)^2A(n+1) = (n+1)^2 + 2\sum_{i=0}^{n} i\,A(i)$$

$$(n+1)^2A(n+1) - n^2A(n) = [(n+1)^2 - n^2] + 2\left[\sum_{i=0}^{n} i\,A(i) - \sum_{i=0}^{n-1} i\,A(i)\right]$$

$$(n+1)^2A(n+1) - n^2A(n) = [2n+1] + 2[nA(n)]$$

$$(n+1)^2A(n+1) - (n^2+2n)A(n) = 2n+1$$

$$\frac{(n+1)^2A(n+1)}{(n+1)(n+2)} - \frac{(n^2+2n)A(n)}{(n+1)(n+2)} = \frac{2n+1}{(n+1)(n+2)}$$

$$\frac{(n+1)A(n+1)}{n+2} - \frac{nA(n)}{n+1} = \frac{2n+1}{(n+1)(n+2)}$$

$$\frac{n+1}{n+2}A(n+1) - \frac{n}{n+1}A(n) = \frac{2n+1}{(n+1)(n+2)}$$

$$\left(\frac{n+1}{n+2}\right)A(n+1) = \left(\frac{n}{n+1}\right)A(n) + \frac{2n+1}{(n+1)(n+2)}$$

13.12

$$f(m) = \left(\frac{m}{m+1}\right)A(m)$$

$$f(1) = \left(\frac{1}{2}\right)A(1) = \left(\frac{1}{2}\right)(1) = \frac{1}{2}$$

$$f(n+1) = f(n) + \frac{2n+1}{(n+1)(n+2)}$$

$$f(2) = f(1) + \frac{2(1)+1}{((1)+1)((1)+2)} = \frac{1}{2} + \frac{3}{2 \cdot 3}$$

$$f(3) = f(2) + \frac{2(2)+1}{((2)+1)((2)+2)} = \left(\frac{1}{2} + \frac{3}{2 \cdot 3}\right) + \frac{5}{3 \cdot 4}$$

$$f(4) = f(3) + \frac{2(3)+1}{((3)+1)((3)+2)} = \left(\frac{1}{2} + \frac{3}{2 \cdot 3} + \frac{5}{3 \cdot 4}\right) + \frac{7}{4 \cdot 5}$$

etc .

$$f(n) = \frac{1}{1 \cdot 2} + \frac{3}{2 \cdot 3} + \frac{5}{3 \cdot 4} + \frac{7}{4 \cdot 5} + \ldots + \frac{2n-1}{n(n+1)}$$

$$f(n) = \sum_{k=1}^{n} \frac{2k-1}{k(k+1)}$$

13.13

$$
\begin{aligned}
f(n) &= \sum_{k=1}^{n} \frac{2k-1}{k(k+1)} \\
&= 2\sum_{k=1}^{n} \frac{k}{k(k+1)} - \sum_{k=1}^{n} \frac{1}{k(k+1)} \\
&= 2\sum_{k=1}^{n} \frac{1}{k+1} - \sum_{k=1}^{n} \left(\frac{1}{k} - \frac{1}{k+1}\right) \\
&= 3\sum_{k=1}^{n} \frac{1}{k+1} - \sum_{k=1}^{n} \frac{1}{k} \\
&= 3\sum_{k=2}^{n+1} \frac{1}{k} - \sum_{k=1}^{n} \frac{1}{k} \\
&= 3\left(\sum_{k=1}^{n+1} \frac{1}{k} - 1\right) - \left(\sum_{k=1}^{n+1} \frac{1}{k} - \frac{1}{n+1}\right) \\
&= 3(H_{n+1} - 1) - \left(H_{n+1} - \frac{1}{n+1}\right) \\
&= 2H_{n+1} - 3 + \frac{1}{n+1}
\end{aligned}
$$

13.14 The largest B-tree of degree m and height h has the maximum $m - 1$ keys in each node, and every internal node has the maximum m children. Thus, it has 1 node at level 0, m nodes at level 1, m^2 nodes at level 2, m^3 nodes at level 3, *etc*. The total number of nodes is:

$$1 + m + m^2 + m^3 + \ldots + m^h = \frac{m^{h+1} - 1}{m - 1}$$

Since each node has $m - 1$ keys, the total maximum number of keys is $m^{h+1} - 1$.

The smallest B-tree of degree m and height h has 1 key in its root and the minimum $\lceil m/2 \rceil$ keys in every other node. Let $x = \lceil m/2 \rceil$. Then this minimal B-tree has 1 node at level 0, 2 nodes at level 1, $2x$ nodes at level 2, $2x^2$ nodes at level 3, *etc*. The total number of non-root nodes is:

$$\begin{aligned} 2 + 2x + 2x^2 + \ldots + 2x^{h-1} &= 2(1 + 2x + 2x^2 + \ldots + 2x^{h-1}) \\ &= 2\left(\frac{x^h - 1}{x - 1}\right) \end{aligned}$$

Since each non-root node has $x - 1$ keys, the total number of keys in all of the non-root nodes is:

$$\begin{aligned} (x-1)\left(2\left(\frac{x^h - 1}{x - 1}\right)\right) &= 2(x^h - 1) \\ &= 2x^h - 2 \end{aligned}$$

There is one key in the root. Thus the total minimum number of keys is $2\lceil m/2 \rceil^h - 1$.

13.15 Let $x = \lceil m/2 \rceil$. Then we have from Formula 13.8: $2x^h - 1 \le n$. Solving for x^h:

$$2x^h - 1 \le n$$

$$2x^h \le n + 1$$

$$x^h \le \frac{n+1}{2}$$

Since $n \ge 1$, $(n + 1)/2 \le (n + n)/2 = n$. Thus, $x^h \le n$ and $h \le \log_x n$.

Now $x = \lceil m/2 \rceil \ge m/2$, so $\lg x \ge \lg(m/2)$. Thus:

$$\begin{aligned} h &\le \log_x n \\ &= \frac{\lg n}{\lg x} \\ &\le \frac{\lg n}{\lg(m/2)} \\ &= \log_{m/2} n \end{aligned}$$

Finally, since m is constant, $h \le c \lg n = \Theta(\lg n)$, where $c = 1/\lg(m/2)$.

Programming Problems

13.1

```
public boolean isBST() {
  if (this == NIL) return true;
  if (this.isSingleton()) return true;
  if (!(this.root instanceof Comparable)) return false;
  if (!(this.left.root instanceof Comparable)) return false;
  if (!(this.right.root instanceof Comparable)) return false;
  Comparable thisRoot = (Comparable)this.root;
  Comparable leftRoot = (Comparable)left.root;
  Comparable rightRoot = (Comparable)right.root;
  if (thisRoot.compareTo(leftRoot) < 0) return false;
  if (thisRoot.compareTo(rightRoot) > 0) return false;
  return left.isBST() && right.isBST();
}
```

13.2

```
public boolean isAVL() {
  if (!isBST()) return false;
  int leftHeight = ( left!=null ? left.height() : -1 );
  int rightHeight = ( right!=null ? right.height() : -1);
  return (Math.abs(leftHeight - rightHeight) < 2);
}
```

13.3

```
public int getHeight() {
  return height;
}
public AVLTree getLeft() {
  if (this == NIL) throw new IllegalStateException();
  return left;
}
public AVLTree getRight() {
  if (this == NIL) throw new IllegalStateException();
  return right;
}
public int getRoot() {
  if (this == NIL) return -1;
  return key;
}
```

13.4

```
public boolean contains(int x) {
  if (this == NIL) return false;
  if (key == x) return true;
  return left.contains(x) || right.contains(x);
}
```

13.5

```
public AVLTree get(int x) {
  public AVLTree get(int x) {
  if (this == NIL || key == x) return this;
```

```
    if (x < key) return left.get(x);
    return right.get(x);
  }
```

13.6

```
  public boolean equals(Object object) {
    if (object == this) return true;
    if (!(object instanceof AVLTree)) return false;
    AVLTree that = (AVLTree)object;
    if (that.key != this.key) return false;
    if (that.height != this.height) return false;
    return that.left.equals(this.left) &&
           that.right.equals(this.right);
    }
```

13.7

```
  public AVLTree(int[] a) {
    this(a[0]);
    for (int i=1; i<a.length; i++)
      add(a[i]);
  }
```

Chapter 14
Heaps and Priority Queues

Exercises

14.1
- **a.** `Byte.MAX_VALUE` ($2^7 - 1 = 127$)
- **b.** `Short.MAX_VALUE` ($2^{31} - 1 = 2{,}147{,}483{,}647$)
- **c.** `Long.MAX_VALUE` ($2^{63} - 1 = 9{,}223{,}372{,}036{,}854{,}775{,}808$)
- **d.** `Float.MAX_VALUE` ($2^{128} - 2^{104} \approx 3.4 \cdot 10^{38}$)
- **e.** `Double.MAX_VALUE` ($2^{1024} - 2^{971} \approx 1.7 \cdot 10^{308}$)

14.2
- **a.** (51, 25, 12, 5, 2, 0); $\lfloor \lg 51 \rfloor = \lfloor 5.67 \rfloor = 5$
- **b.** (333, 166, 82, 40, 19, 9, 4, 1, 0); $\lfloor \lg 333 \rfloor = \lfloor 8.38 \rfloor = 8$
- **c.** (1000, 499, 249, 124, 61, 30, 14, 6, 2, 0); $\lfloor \lg 1000 \rfloor = \lfloor 9.97 \rfloor = 9$
- **d.** (1024, 511, 255, 127, 63, 31, 15, 7, 3, 1, 0); $\lfloor \lg 1024 \rfloor = \lfloor 10.0 \rfloor = 10$

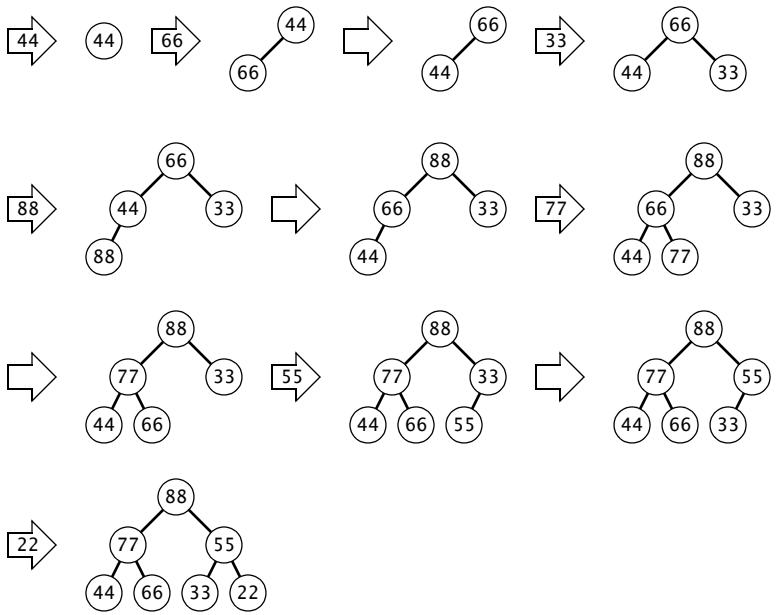
14.3 Only trees **b** and **e** are heaps.

14.4 Only arrays **b** and **d** are heaps.

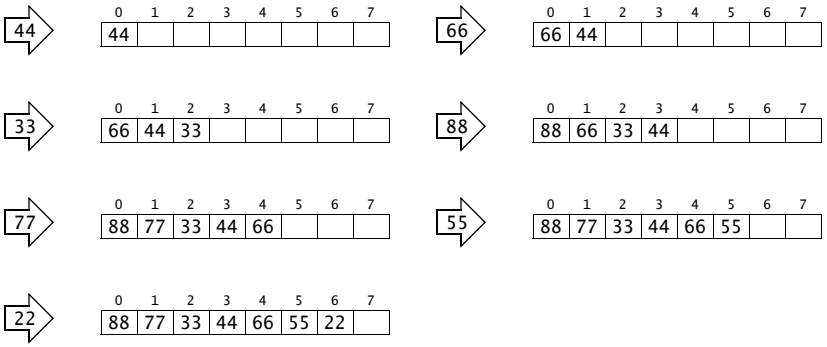
14.5 Let $m = n/2$ (integer division). Then $n = 2m$ or $2m + 1$, according to whether n is even or odd. But in either case, $2m \le n$. Now assume that $i < n/2$. Then $i < m$. Since n and m are integers, it follows that $i \le m - 1$. Thus, $2i + 1 \le 2(m - 1) + 1 = 2m - 1 \le n - 1 < n$.

14.6
- **a.** $\lfloor \lg 10 \rfloor = \lfloor 3.23 \rfloor = 3$
- **b.** $\lfloor \lg 1000 \rfloor = \lfloor 9.97 \rfloor = 9$
- **c.** $\lfloor \lg 1024 \rfloor = \lfloor 10.0 \rfloor = 10$
- **d.** $\lfloor \lg 1{,}000{,}000 \rfloor = \lfloor 19.93 \rfloor = 19$

14.7



14.8



14.9 **a.** The worst case is when the array is in descending order.
b. The algorithm makes only 1 comparison in the bast case.
c. The best case is when the array is in ascending order.

14.10 The parent of the node at `a[i]` will be at `a[i/2]`, and the children will be at `a[2*i]` and `a[2*i+1]`.

14.11 The parent of the node at `a[i]` will be at `a[i/m]`, and the children will be at `a[m*i]`, `a[m*i+1]`, `a[m*i+2]`, ..., `a[m*i+m-1]`.

14.12 Both methods run in logarithmic time: $\Theta(\lg n)$.

14.13 If elements A, B, C, and D with keys 44, 44, 33, and 22, are inserted in that order, A will be removed before B. But if their keys are 44, 44, 33, and 66, then B will be removed before A.

14.14 Each iteration of the loop moves the larger child up to become the parent of its sibling. The loop stops when neither child is greater than the original x value, which is copied into the last position vacated by the loop. This renders the affected path descending, all the way down to the leaf level. Since the first step of the loop moves the maximum element up into the root, the precondition guarantees that all other root-to-leaf paths are now descending.

14.15 If S is a subtree of T, then every root-to-leaf path in S is part of some root-to-leaf path in T. So if T is a heap, then every root-to-leaf path in S must be descending.

14.16 The derivation on page 419 shows that $W(n)$ is the greatest number of comparisons that the Build Heap algorithm can make on a hep of size n, where $W(n) = 2W(n) + 2\lg n$. We show by direct substitution that $f(n) = 5n - 2\lg n - 4$ is a solution to this recurrence equation:

$$\begin{aligned} f(n/2) &= 5(n/2) - 2\lg(n/2) - 4 \\ &= 5n/2 - 2(\lg n - \lg 2) - 4 \\ &= 5n/2 - 2(\lg n - 1) - 4 \\ &= 5n/2 - 2\lg n - 2 \\ 2f(n/2) + 2\lg n &= 2(5n/2 - 2\lg n - 2) + 2\lg n \\ &= 5n - 4\lg n - 4 + 2\lg n \\ &= 5n - 2\lg n - 4 \\ &= f(n) \end{aligned}$$

The inital value is $f(1) = 5(1) - 2\lg(1) - 4 = 5 - 0 - 4 = 1$, which satisfies $W(1) = 1$.

14.17 Let x be the largest element in a heap. If x has a parent, then its value cannot be less than x because of the heap property. And since x is the maximum element, its parent cannot be greater than x. Thus, if x has a parent, it must have the same value. By the same reasoning, every ancestor of x must have the same value. The root is an ancestor of x. Thus, the root has the maximum value.

14.18

a. $l(i) = 2i + 1$

b. $r(i) = 2i + 2$

c. $l(i, k) = 2^k(i + 1) - 1$

d. $r(i, k) = 2^k(i + 2) - 2$

e. $a(i, k) = i/2^k$

Programming Problems

14.1

```
public boolean isHeap(int[] a, int size) {
  return isHeap(a, 0, size);
}
public boolean isHeap(int[] a, int k, int size) {
  if (k >= size/2) return true;  // singletons are heaps
  int left = 2*k + 1, right = left + 1;
  if (a[k] < a[left]) return false;
  if (a[k] < a[right]) return false;
  return isHeap(a, left, size) && isHeap(a, left, size);
}
```

14.2

```
public boolean isHeap(int[] a, int size) {
  for (int i=(size-1)/2; i<size; i++) {
    int j=i;
    while (j > 0) {
      int pj=(j-1)/2;  // a[pj] is the parent of a[j]
      if (a[j] > a[pj]) return false;
      j = pj;
    }
  }
  return true;
}
```

14.3

```
import java.util.*;
public class UnsortedListPriorityQueue implements PriorityQueue {
  private List list = new LinkedList();
  public void add(Object object) {
    if (!(object instanceof Comparable))
      throw new IllegalArgumentException();
    list.add(object);
  }
  public Object best() {
    if (list.isEmpty()) throw new IllegalMonitorStateException();
    Iterator it=list.iterator();
    Comparable best = (Comparable)it.next();
    while (it.hasNext()) {
      Comparable x = (Comparable)it.next();
      if (x.compareTo(best) > 0) best = x;
    }
    return best;
  }
```

```
  public Object remove() {
    Object best = best();
    list.remove(best);
    return best;
  }
  public int size() {
    return list.size();
  }
}
```

14.4

```
import java.util.*;
public class SortedListPriorityQueue implements PriorityQueue {
  private List list = new LinkedList();
  public void add(Object object) {
    if (!(object instanceof Comparable))
      throw new IllegalArgumentException();
    if (list.isEmpty()) {
      list.add(object);
      return;
    }
    Iterator it=list.iterator();
    Comparable x = (Comparable)object;
    int i=0;
    while (it.hasNext()) {
      Comparable y = (Comparable)it.next();
      if (y.compareTo(x) <= 0) break;
      ++i;
    }
    list.add(i, x);
  }
  public Object best() {
    if (list.isEmpty()) throw new IllegalMonitorStateException();
    return list.get(0);
  }
  public Object remove() {
    return list.remove(0);
  }
  public int size() {
    return list.size();
  }
}
```

14.5

```
public class PriorityQueue {
  private final int CAPACITY=100;
  private boolean[][] q = new boolean[CAPACITY][CAPACITY];
  private int[] len = new int[CAPACITY];
  public void add(int x) {
    if (x >= CAPACITY) throw new IllegalArgumentException();
    if (len[x] >= CAPACITY) throw new IllegalStateException();
    q[x][len[x]++] = true;
  }
  public int best() {
    if (size() == 0) throw new IllegalStateException();
    for (int i=CAPACITY-1; i>=0; i--) {
      if (len[i] > 0) {
        return i;
      }
    }
    throw new IllegalStateException();
  }
  public int remove() {
    if (size() == 0) throw new IllegalStateException();
    for (int i=CAPACITY-1; i>=0; i--) {
      if (len[i] > 0) {
        --len[i];
        return i;
      }
    }
    throw new IllegalStateException();
  }
  public int size() {
    int size = 0;
    for (int i=0; i<CAPACITY; i++) {
      size += len[i];
    }
    return size;
  }
}
```

14.6

```
public class TestHeapify {
  public TestHeapify() {
    int[] a = {33,88,77,52,66,73,75,44,48,69};
    int n = a.length;
    IntArrays.print(a);
    heapify(a, 0, n);
```

```
    IntArrays.print(a);
  }
  public static void main(String[] args) {
    new TestHeapify();
  }
  void heapify(int[] a, int i, int n) {
    if (i >= n/2) return;     // a[i] is a leaf
    int j = 2*i + 1;
    if (j+1 < n && a[j+1] > a[j]) ++j;
    if (a[j] <= a[i]) return;
    int ai = a[i];
    a[i] = a[j];
    a[j] = ai;
    heapify(a, j, n);
  }
}
```

14.7

```
public class PQStack implements Stack {
  private PriorityQueue pq = new HeapPriorityQueue();
  private static int nextIndex;
  private static class PQElement implements Comparable {
    Object object;
    int index;
    PQElement(Object object) {
      this.object  = object;
      this.index = PQStack.nextIndex++;
    }
    public int compareTo(Object object) {
      PQElement that = (PQElement)object;
      Long diff = new Long(this.index - that.index);
      return diff.intValue();
    }
  }
  public void push(Object object) {
    pq.add(new PQElement(object));
  }
  public Object peek() {
    PQElement e = (PQElement)pq.best();
    return e.object;
  }
  public boolean isEmpty() {
    return pq.size()==0;
  }
```

```
  public Object pop() {
    PQElement e = (PQElement)pq.remove();
    return e.object;
  }
  public int size() {
    return pq.size();
  }
}
```

14.8

```
public class PQQueue implements Queue {
  private PriorityQueue pq = new HeapPriorityQueue();
  private static int nextIndex;
  private static class PQElement implements Comparable {
    Object object;
    int index;
    PQElement(Object object) {
      this.object  = object;
      this.index = PQQueue.nextIndex++;
    }
    public int compareTo(Object object) {
      PQElement that = (PQElement)object;
      Long diff = new Long(that.index - this.index);
      return diff.intValue();
    }
  }
  public void add(Object object) {
    pq.add(new PQElement(object));
  }
  public Object first() {
    PQElement e = (PQElement)pq.best();
    return e.object;
  }
  public boolean isEmpty() {
    return pq.size()==0;
  }
  public Object remove() {
    PQElement e = (PQElement)pq.remove();
    return e.object;
  }
  public int size() {
    return pq.size();
  }
}
```

14.9

```
public class PriorityQueue {
  private final int CAPACITY=100;
  private int[] a = new int[CAPACITY];
  private int size;
  public void add(int x) {
    if (size == CAPACITY) throw new IllegalArgumentException();
    a[size++] = x;
  }
  public int best() {
    return a[indexOfBest()];
  }
  public int remove() {
    if (size() == 0) throw new IllegalStateException();
    int b=indexOfBest();
    int best = a[b];
    --size;
    for (int i=b; i<size; i++)
      a[i] = a[i+1];
    return best;
  }
  public int size() {
    return size;
  }
  private int indexOfBest() {
    if (size() == 0) throw new IllegalStateException();
    int b=0;
    for (int i=1; i<size; i++)
      if (a[i] >= a[b]) b = i;
    return b;
  }
}
```

14.10

```
public class PriorityQueue {
  private Node start = new Node(0, null);  // dummy
  private int size;
  public void add(int x) {
    start.next = new Node(x, start.next);
    ++size;
  }
  public int best() {
    return predecessorOfBest().next.key;
  }
```

```
  public int remove() {
    Node b = predecessorOfBest();
    int best = b.next.key;
    b.next = b.next.next;
    --size;
    return best;
  }
  public int size() {
    return size;
  }
  private Node predecessorOfBest() {
    if (size==0) throw new IllegalStateException();
    Node b = start;
    for (Node p=b.next; p.next!=null; p = p.next)
      if (p.next.key > b.next.key) b = p;
    return b;
  }
  private static class Node {
    int key;
    Node next;
    Node(int key, Node next) {
      this.key = key;
      this.next = next;
    }
  }
}
```

Chapter 15
Sorting

Exercises

15.1 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1		M	O					
2					T	U		
3						E	U	
4							R	U
5					E	T		
6						R	T	
7				E	P			
8			E	O				
9	C	E	M	O	P	R	T	U

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1					R			U
2						E	T	
3					E	R		
4				E	P			
5	C	E	M	O	P	R	T	U

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1		M	O					
2					T	U		
3		E	M	O	P	T	U	
4	C	E	M	O	P	R	T	U

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1			E				M	
2					M		U	
3						R		T
4		E	O					
5			M	O	P			
6	C	E	M	O	P	R	T	U

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1					T	U		
2		M	O		E	R	T	U
3	C	E	M	O	P	R	T	U

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1	C							
2		E		O			P	
3					R			U
4	C	E	M	O	P	R	T	U

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	C	O	M	P	U	T	E	R
1				R				P
2			T			M		
3		U			O			
4	U	R		P				C
5	C							U
6	T		M			C		
7	E						T	

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
8	R	P		E	O			
9	C					R		
10	P	O			C			
11	C				P			
12	O	E		C				
13	C			O				
14	M		C					
15	C		M					
16	E	C						
17	C	E	M	O	P	R	T	U

h. Bucket Sort:

a = 'A', $b - 1$ = 'Z', $b - a$ = 26, n = 8, Δx = 26/8 = 3.25:

C, O, M, P, U, T, E, R →

Bucket #0 ('A' through 'C'): C

Bucket #1 ('D' through 'F'): E

Bucket #2 ('G' through 'I'):

Bucket #3 ('J' through 'L'):

Bucket #4 ('M' through 'P'): O, M, P → M, O, P

Bucket #5 ('Q' through 'S'): R

Bucket #6 ('T' through 'V'): U, T → T, U

Bucket #7 ('W' through 'Z'):

→ C, E, M, O, P, R, T, U

15.2 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1		G	I									
2			C	I								
3					M	O						
4								T	U			
5									E	U		
6										R	U	
7											S	U
8		C	G									
9								E	T			
10									R	T		
11										S	T	
12							E	P				
13						E	O					
14					E	M						
15				E	I							
16	B	C	E	G	I	M	O	P	R	S	T	U

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1								S				U
2									R		T	
3								E		S		
4							E	P				
5					E		O					
6		E			I							
7			C	G								
8	B	C	E	G	I	M	O	P	R	S	T	U

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1		G	I									
2		C	G	I								
3					M	O						
4								T	U			
5			E	G	I	M	O	P	T	U		
6									R	T	U	
7	B	C	E	G	I	M	O	P	R	S	T	U

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1							E			P		
2								R			U	
3									S			T
4		G	I									
5		C	G	I								
6					M	O						
7			E	G	I	M	O					
8								P	R	S		
9	B	C	E	G	I	M	O	P	R	S	T	U

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1		G	I									
2					M	O						
3		C	G	I								
4								T	U			
5							E	P	R	S	T	U
6	B	C	E	G	I	M	O	P	R	S	T	U

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1		E			I					O		
2		C	E	G								
3							O	P		U		
3									S	R	T	U
4	B	C	E	G	I	M	O	P	R	S	T	U

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]
0	B	I	G	C	O	M	P	U	T	E	R	S
1						S						M
2					R						O	
3				U				C				
4			S			M						G
5		U		T					I			
6	U	T		I					B			
7	G											U
7	T	R			O						G	
8	G										T	
10	S		P				G					
11	E									S		
12	R	O			E							
13	B								R			
14	P		M			B						
15	C							P				
16	O	I		C								
17	G						O					
18	M		G									
19	B					M						
20	I	E			B							
21	B				I							
22	G		B									
23	C			G								
24	E	C										
25	B		E									
26	C	B										
27	B	C	E	G	I	M	O	P	R	S	T	U

h. Bucket Sort:

a = 'A', $b - 1$ = 'Z', $b - a$ = 26, n = 12, Δx = 26/12 = 3.17:

B, I, G, C, O, M, P, U, T, E, R, S →

Bucket #0 ('A' and 'B'):B

Bucket #1 ('C' and 'D'):C

Bucket #2 ('E' and 'F'):E

Bucket #3 ('G' and 'H'):G

Bucket #4 ('I' and 'J'):I

Bucket #5 ('K', 'L', and 'M'):M

Bucket #6 ('N' and 'O'):O

Bucket #7 ('P' and 'Q'): P

Bucket #8 ('R' and 'S'):R, S

Bucket #9 ('T' and 'U'):U, T → T, U

Buckets #10 ('V' and 'W') and #11 ('X', 'Y', and 'Z') are empty

→ B, C, E, G, I, M, O, P, R, S, T, U

15.3 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1	44	77						
2		22	77					
3					55	99		
4						33	99	
5							66	99
6	22	44						
7				55	88			
8					33	88		
9						66	88	
10			55	77				
11				33	77			
12					66	77		
13			33	55				
14	22	33	44	55	66	77	88	99

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1					66			99
2				33			88	
3	55					77		
4	33			55				
5		22	44					
6	22	33	44	55	66	77	88	99

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1	44	77						
2	22	44	77					
3			55	77	88	99		
4		33	44	55	77	88	99	
5	22	33	44	55	66	77	88	99

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1				66				88
2	22		77					
3			33		77		99	
4				55		66		
5		33	44					
6					66	77		
7	22	33	44	55	66	77	88	99

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1	44	77						
2	22	44	77					
3					55	99		
4					33	55	66	99
5	22	33	44	55	66	77	88	99

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1	66			33	55	77	99	88
2	55				66			
3	33			55				
4	22	33	44					
5	22	33	44	55	66	77	88	99

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	77	44	22	88	99	55	33	66
1			55			22		
2		99			44			
3	99	88		77				
4	66							99
5	88	77		66				
6	33						88	
7	77	66		33				
8	22					77		
9	66	44			22			
10	22				66			
11	55		22					
12	33			55				
13	44	33						
14	22	33	44					
15	33	22						
16	22	33	44	55	66	77	88	99

h. Bucket Sort:

$a = 0$, $b = 100$, $b - a = 100$, $n = 8$, $\Delta x = 100/8 = 12.5$:

77, 44, 22, 88, 99, 55, 33, 66 →

Bucket #0 (0 – 12):

Bucket #1 (13 – 24):22

Bucket #2 (25 – 37):33

Bucket #3 (38 – 49):44

Bucket #4 (50 – 62):55

Bucket #6 (63 – 74):66

Bucket #7 (75 – 87):77

Bucket #8 (88 – 99):88, 99

→ 22, 33, 44, 55, 66, 77, 88, 99

15.4 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1		38	83					
2			52	83				
3					47	94		
4						29	94	
5							76	94
6	38	61						

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
7		52	61					
8				47	83			
9					29	83		
10						76	83	
11			47	61				
12				29	61			
13		47	52					
14			29	52				
15		29	47					
16	29	38	47	52	61	76	83	94

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1					76			94
2		29					83	
3					47	76		
4	47				61			
5	38		47					
6	29	38	47	52	61	76	83	94

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1	38	61	83					
2		52	61	83				
3		47	52	61	83	94		
4	29	38	47	52	61	83	94	
5	29	38	47	52	61	76	83	94

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1		47				83		
2			29				38	
3	29		61					
4			38		61		94	
5						76		83
6		38	47					
7	29	38	47	52	61	76	83	94

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1	38	52	61	83				
2					47	94		
3					29	47	76	94
4	29	38	47	52	61	76	83	94

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1	29	47			61	94	83	
2		38	47					
3	29	38	47	52	61	76	83	94

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	61	83	38	52	94	47	29	76
1				76				52
2			47			38		
3		94			83			
4	94	83			61			
5	52							94
6	83	76		52				
7	29						83	
8	76	61			29			
9	38	52				76		
10	61	44		38				
11	29				61			
12	52	38		29				
13	29			52				
14	47		29					
15	29	33	47					
16	38	29						
17	29	38	47	52	61	76	83	94

h. Bucket Sort:

$a = 0$, $b = 100$, $b - a = 100$, $n = 8$, $\Delta x = 100/8 = 12.5$:

61, 83, 38, 52, 94, 47, 29, 76 →

Bucket #0 (0 – 12):

Bucket #1 (13 – 24):

Bucket #2 (25 – 37):29

Bucket #3 (38 – 49):38, 47

Bucket #4 (50 – 62):61, 52 → 52, 61

Bucket #6 (63 – 74):

Bucket #7 (75 – 87):83, 76 → 76, 83

Bucket #8 (88 – 99):94

→ 29, 38, 47, 52, 61, 76, 83, 94

15.5 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1	22	33	44	55	66	77	88	99

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	22	33	44	55	66	77	88	99
1				99				55
2			88				44	
3		99		55				33
4	99	66			22			
5	33							99
6	88		77			33		
7	44						88	
8	77		44					
9	33					77		
10	66	55		33				
11	22				66			
12	55	33		22				
13	22			55				
14	44		22					
15	22		44					
16	33	22						
17	22	33	44	55	66	77	88	99

h. Bucket Sort:

$a = 0$, $b = 100$, $b - a = 100$, $n = 8$, $\Delta x = 100/8 = 12.5$:

22, 33, 44, 55, 66, 77, 88, 99 $\rightarrow$

Bucket #0 (0 – 12):

Bucket #1 (13 – 24):22

Bucket #2 (25 – 37):33

Bucket #3 (38 – 49):44

Bucket #4 (50 – 62):55

Bucket #6 (63 – 74):66

Bucket #7 (75 – 87):77

Bucket #8 (88 – 99):88, 99

$\rightarrow$ 22, 33, 44, 55, 66, 77, 88, 99

15.6 **a.** Bubble Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	88	99						
2		77	99					
3			66	99				
4				55	99			
5					44	99		
6						33	99	
7							22	99
8	77	88						
9		66	88					
10			55	88				
11				44	88			
12					33	88		
13						22	88	
14	66	77						
15		55	77					
16			44	77				
17				33	77			
18					22	77		
19	55	66						
20		44	66					
21			33	66				
22				22	66			
23	44	55						
24		33	55					
25			22	55				
26	33	44						
27		22	44					
28	22	33	44	55	66	77	88	99

b. Selection Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	22							99
2		33					88	
3			44			77		
4	22	33	44	55	66	77	88	99

c. Insertion Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	88	99						
2	77	88	99					
3	66	77	88	99				
4	55	66	77	88	99			
5	44	55	66	77	88	99		
6	33	44	55	66	77	88	99	
7	22	33	44	55	66	77	88	99

d. Shell Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	55				99			
2		44				88		
3			33				77	
4				22				66
5	33		55					
6					77		99	
7		22		44				
8						66		88
9	22	33						
10			44	55				
11					66	77		
12	22	33	44	55	66	77	88	99

e. Merge Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	88	99						
2			66	77				
3					44	55		
4							22	33
5	66	77	88	99				
6					22	33	44	55
7	22	33	44	55	66	77	88	99

f. Quick Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	22							99
2		33					88	
3			44			77		
4	22	33	44	55	66	77	88	99

g. Heap Sort:

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
0	99	88	77	66	55	44	33	22
1	22							99
2	88	66		22				
3	33						88	
4	77		44			33		
5	33					77		
6	66	55			33			
7	33				66			
8	55	33						
9	22			55				
10	44		22					
11	22	33	44					
12	33	22						
13	22	33	44	55	66	77	88	99

h. Bucket Sort:

$a = 0$, $b = 100$, $b - a = 100$, $n = 8$, $\Delta x = 100/8 = 12.5$:

99, 88, 77, 66, 55, 44, 33, 22 $\rightarrow$

Bucket #0 (0 – 12):

Bucket #1 (13 – 24):22

Bucket #2 (25 – 37):33

Bucket #3 (38 – 49):44

Bucket #4 (50 – 62):55

Bucket #6 (63 – 74):66

Bucket #7 (75 – 87):77

Bucket #8 (88 – 99):88, 99

$\rightarrow$ 22, 33, 44, 55, 66, 77, 88, 99

15.7 **a.** Selection Sort:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	0
1		0								1						
2					0				1							
3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

b. Insertion Sort:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	0
1		0	1													
2			0	1												
3				0			1									
4					0				1							
5						0				1						
6	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

c. Merge Sort:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	0
1		0		1												
2					0		1									
3				0	1											
4															0	1
5													0		1	
6											0		1			
7	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

d. Quick Sort:

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]	[12]	[13]	[14]	[15]
0	0	1	0	0	1	1	0	1	0	0	1	1	1	1	1	0
1		0								1						
2					0				1							
3	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1

e. Counting Sort:

$a = \{0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0\}$, $n = 16$, $m = 2$,

$c = \{7, 9\} \rightarrow c = \{7, 16\}$

i	$k = a_i$	c_0	c_1	$j = c_k$	$b_j = k$
		7	16		
15	0	6		6	0
14	1		15	15	1
13	1		14	14	1
12	1		13	13	1
11	1		12	12	1
10	1		11	11	1
9	0	5		5	0
8	0	4		4	0
7	1		10	10	1
6	0	3		3	0
5	1		9	9	1
4	1		8	8	1
3	0	2		2	0
2	0	1		1	0
1	1		7	7	1
0	0	0		0	0

$b = \{0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1\}$

15.8 **a.** Selection Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	US	CA	CN	US	GB	JA	FR	CN	US	DE	JA	GB	US	CN	US	IT	AU	ES	US	US
1	ES																	US		
2				AU													US			
3									IT							US				
4													CN	US						
5						CN							JA							
6											GB	JA								
7									GB		IT									
8					DE					GB										
9							CN	FR												
10	CN						ES													
11					CN	DE														
12	AU	CA	CN	CN	CN	DE	ES	FR	GB	GB	IT	JA	JA	US	US	US	US	US	US	US

b. Insertion Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	US	CA	CN	US	GB	JA	FR	CN	US	DE	JA	GB	US	CN	US	IT	AU	ES	US	US
1	CA	US																		
2		CN	US																	
3			GB		US															
4				JA		US														
5			FR	GB	JA		US													
6			CN	FR	GB	JA		US												
7				DE	FR	GB	JA			US										
8								JA			US									
9							GB		JA			US								
10				CN	DE	FR		GB		JA				US						
11									IT		JA					US				
12	AU	CA			CN	DE	FR		GB	IT		JA					US			
13	AU	CA	CN	CN	CN	DE	ES	FR	GB	GB	IT	JA	JA	US	US	US	US	US	US	US

c. Merge Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	US	CA	CN	US	GB	JA	FR	CN	US	DE	JA	GB	US	CN	US	IT	AU	ES	US	US
1	CA	US																		
2				GB	US															
3		CN	GB	US																
4						FR	JA													
5									DE	US										
6						CN	DE	FR	JA											
7			CN	DE	FR	GB	JA	US	US											
8											GB	JA								
9													CN	US						
10											CN	GB	JA							
11																AU	IT			
11																	ES	IT		
12																				
13											AU	CN	ES	GB	IT	JA	US	US		
14	AU	CA	CN	CN	CN	DE	ES	FR	GB	GB	IT	JA	JA	US	US	US	US	US	US	US

d. Quick Sort:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	US	CA	CN	US	GB	JA	FR	CN	US	DE	JA	GB	US	CN	US	IT	AU	ES	US	US
1	ES																	US		
2	AU			CN	DE	CN	ES	FR		JA				GB			US			
3					CN	DE														
4									IT							US				
5									GB	GB	IT	JA		JA						
6	AU	CA	CN	CN	CN	DE	ES	FR	GB	GB	IT	JA	JA	US	US	US	US	US	US	US

e. Counting Sort:

a = {US, CA, CN, US, GB, JA, FR, CN, US, DE, JA, GB, US, CN, US, IT, AU, ES, US, US}, $n = 20$, $m = 10$:

k	Key	c_k	$c_{k'}$
0	AU	1	1
1	CA	1	2
2	CN	3	5
3	DE	1	6
4	ES	1	7
5	FR	1	8
6	GB	2	10
7	IT	1	11
8	JA	2	13
9	US	7	20

i	a_i	k	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8	c_9	$j = c_k$	$b_j = a_i$
			1	2	5	6	7	8	10	11	13	20		
19	US	9										19	19	US
18	US	9										18	18	US
17	ES	4					6						6	ES
16	AU	0	0										0	AU
15	IT	7								10			10	IT
14	US	9										17	17	US
13	CN	2			4								4	CN
12	US	9										16	16	US
11	GB	6							9				9	GB
10	JA	8									12		12	JA
9	DE	3				5							5	DE
8	US	9										15	15	US
7	CN	2			3								3	CN
6	FR	5						7					7	FR
5	JA	8									11		11	JA
4	GB	6							8				8	GB
3	US	9										14	14	US
2	CN	2			2								2	CN
1	CA	1		1									1	CA
0	US	9										13	13	US

b = {AU, CA, CN, CN, CN, DE, ES, FR, GB, GB, IT, JA, JA, US, US, US, US, US, US, US}

15.9 If $t = c\,n^2$ and $n' = 2n$, then $t' = c(n')^2 = c(2n)^2 = (4)(c\,n^2) = 4t$;
i.e., the time for an array of length $2n$ will be about 4 times as long.

a. 4(5 ms) = 20 ms

b. 4(20) ms = 80 ms

c. 4(80) ms = 320 ms

d. 4(320) ms = 1280 ms = 1.26 s

15.10 If $t = c\,n \lg n$ and $n' = n^2$, then $t' = c\,n' \lg n' = c\,(n^2)\lg(n^2) = (2n)(c\,n \lg n) = (2n)\,t$;
i.e., the time for an array of length n^2 will be about $2n$ times as long.

a. 2(200)(3 μs) = 1200 μs = 1.2 ms

b. 2(40,000)(1.2 ms) = 96,000 ms = 1600 s = 26 min 40 s

15.11 If $t = c\,n$ and $n' = 2n$, then $t' = c(n') = c(2n) = (2)(c\,n) = 2t$;
i.e., the time for an array of length $2n$ will be about twice as long.

a. 2(7 ms) = 14 ms

b. 2(14 ms) = 28 ms

c. 2(28 ms) = 56 ms

d. 2(56 ms) = 112 ms

15.12 If $t = c\,n \lg n$ and $n' = 2n$,
then $t' = c\,n' \lg n' = c\,(2n)\lg(2n) = 2(c\,n \lg n)(\lg(2n)/\lg n = 2t \lg(2n)/\lg n$.
But $\lg(2n) = \lg 2 + \lg n = 1 + \lg n$, so $\lg(2n)/\lg n = (1 + \lg n)/\lg n = 1 + 1/\lg n$.
Thus, $t' = 2t\,(1 + 1/\lg n)$.
With n = 1,000,000, $1/\lg n = \log 2/\log n = 0.301/6 = 0.05$.
Thus, $t' = 2t\,(1 + 0.05) = 2(20\text{ ms})(1.05) = 42$ ms.

15.13 Here is Listing 15.3:

```
1    void sort(int[] a) {
2      for (int i = a.length-1; i > 0; i--) {
3        for (int j = 0; j < i; j++)
4          if (a[j] > a[j+1]) swap(a, j, j+1);
5        // INVARIANTS: a[i] = max{a[0..i]};
6      }
7      // INVARIANT: a[i..a.length) is in ascending order;
8    }
```

From lines 3–4, we know that `a[j]` ≤ `a[j+1]` for all `j` < `i`.
Thus: `a[0]` ≤ `a[1]` ≤ `a[2]` ≤ `...` ≤ `a[i]`.
Therefore, `a[i]` = max{`a[0]`, `a[1]`, `a[2]`, `...`, `a[i]`} = `max{a[0..i]}`.

15.14 Continuing from the solution to Exercise 15.13:
From line 2, we see that the invariant on line 6 holds for each `i` > 0, including `i` = `a.length-1`. Thus: `a[0]` ≤ `a[1]` ≤ `a[2]` ≤ `...` ≤ `a[i]` ≤ `...` ≤ `a[a.length-1]`.

15.15 **Algorithm. Descending Bubble Sort**

1. Repeat steps 2–3 for $i = n - 1$ down to 1:
2. Repeat step 3 for $j = 0$ up to $i - 1$.
3. If $a_j < a_{j+1}$, swap them.

15.16 Algorithm. An Improved Bubble Sort

1. Initialize a *flag* to *false*.
2. Let $i = n - 1$.
3. Repeat steps 4-8 while the *flag* is *false*.
4. Set the `flag` to `true`.
5. Repeat step 6 for $j = 0$ up to $i - 1$.
6. If $a_j > a_{j+1}$, swap them and reset the *flag* to *false*.
7. If $i = 1$, *return*.
8. Decrement i.

With this improvement, the algorithm will terminate as soon as the inner repeat loop (steps 6-7) is able to iterate without making a swap. Consequently, in the bast case, where the list is already in ascending order, the main loop (steps 3-8) will iterate only once, thereby running in linear time: $\Theta(n)$.

The worst case is still $\Theta(n^2)$.

15.17 Algorithm. Selection Sort

1. Repeat step 2 for $i = 0$ up to $n - 2$.
2. Swap a_i with $\min\{a_i \ldots a_{n-1}\}$.

15.18 Algorithm. External Selection Sort

1. Open a new temporary file named *temp* for writing.
2. Open the specified file, henceforth called *specFile*, for reading.
3. Read through *specFile* to obtain its number of lines n.
4. Repeat steps 5–11 for $i = 0$ to $n - 2$.
5. Reset *specFile* to be read again from its beginning.
6. Read the first i lines of *specFile*.
7. Read the next line of *specFile* into a string named *line*.
8. Repeat lines 9–10 as long as *specFile* has more lines:
9. Read the next line of *specFile* into a string named *nextLine*.
10. If *nextLine* alphabetically precedes *line*, copy it into *line*.
11. Write *line* to the end of the *temp* file.
12. Copy the last line of *specFile* to *temp*.
13. Close both files.
14. Open the *temp* file for reading.
15. Open the *specFile* for writing.
16. Copy all n lines from *temp* to *specFile*.
17. Close both files.

15.19 The Indirect Insertion Sort:

```
1   int[] sort(int[] a, int[] k) {
2     int[] k = new int[a.length];
3     for (int i = 0; i < k.length; i++) {
4       k[i] = i;
5     for (int i = 1; i < a.length; i++) {
6       int ai = a[i], j = i;
7       for (j = i; j > 0 && a[k[j-1]] > ai; j--)
```

```
        k[j] = k[j-1];
      k[j] = ai;
    }
  }
```

15.20 The Merge Sort for Linked Lists:

```
  Node sorted(Node list) {
    if (list == null || list.next == null) return list;
    // temporarily add a dummy head node:
    Node temp = new Node(0, list);
    sort(temp);
    return temp.next;
  }

  // recursively sorts the data from p.next to q:
  void sort(Node left) {
    if (left.next==null || left.next.next==null) return;
    Node right = new Node(0, null);
    split(left, right);
    sort(left);
    sort(right);
    merge(left, right);
  }

  // cuts left in two, returning second half in right:
  void split(Node left, Node right) {
    Node p=left, q=p.next;
    while (q != null && q.next != null) {
      p = p.next;
      q = q.next.next;
    }
    right.next = p.next;
    p.next = null;
  }

  // merges left and right into left:
  void merge(Node left, Node right) {
    if (right.next==null) return;
    Node pp=left, p=pp.next, q=right.next;
    while (p != null && q != null) {
      if (p.data < q.data) {
        pp = p;
        p = p.next;
      } else {
        pp = pp.next = q;
        q = q.next;
        pp.next = p;
      }
    }
```

```
44      pp.next = (p==null? q: p);
45    }
```

15.21 The Shell Sort for subarrays:

```
1   void sort(int[] a, int p, int q) {
2     for (int d = (q-p)/2; d > 0; d /= 2)
3       for (int c = p; c < p+d; c++)
4         iSort(a, c, d);
5   }
```

15.22 Here is the code from Listing 15.12 on page 460:

```
1   void sort(int[] a) {
2     for (int i = (a.length-1)/2; i >= 0; i--)
3       heapify(a, i, a.length);
4     for (int j = a.length-1; j > 0; j--) {
5       swap(a, 0, j);
6       heapify(a, 0, j);
7     }
8   }
```

The steps in the trace in Figure 15.11 are numbered. This line that each is executing is shown in the following table:

Step	Line
1	3: heapify() at a[3]
2	3: heapify() at a[1]
3	3: heapify() at a[0]
4	5: swap() a[0] with a[7]
5	6: heapify() at a[0]
6	5: swap() a[0] with a[6]
7	6: heapify() at a[0]
8	5: swap() a[0] with a[5]
9	6: heapify() at a[0]
10	5: swap() a[0] with a[4]
11	6: heapify() at a[0]
12	5: swap() a[0] with a[3]
13	6: heapify() at a[0]
14	5: swap() a[0] with a[2]

15.23 Algorithm. A Modified Shell Sort

1. Let $d = 1$.
2. Repeat step 3 while $d < n$.
3. Set $d = 3d + 1$.
4. Set $d = (d - 1)/3$.
5. Repeat steps 6–7 while $d > 0$.
6. Apply the insertion sort to each of the d subsequences

$$\{a_0, a_d, a_{2d}, a_{3d}, \ldots\}$$
$$\{a_1, a_{d+1}, a_{2d+1}, a_{3d+1}, \ldots\}$$
$$\{a_2, a_{d+2}, a_{2d+2}, a_{3d+2}, \ldots\}$$
$$\vdots$$
$$\{a_{d-1}, a_{2d-1}, a_{3d-1}, a_{4d-1}, \ldots\}$$

7. Set $d = (d - 1)/3$.

15.24 **a.** The Insertion Sort.
b. The Selection Sort.
c. The Heap Sort.
d. The Heap Sort.

15.25 Algorithm. The Shaker Sort

1. Let $p = 0$ and $q = n - 2$.
2. Repeat steps 3–10 forever:
3. Repeat step 4 for $i = p$ up to q.
4. If $a_i > a_{i+1}$, swap them.
5. Decrement q.
6. If $p = q$, return.
7. Repeat step 8 for $j = q$ down to p.
8. If $a_j > a_{j+1}$, swap them.
9. Increment p.
10. If $p = q$, return.

This has the same complexity as the Bubble Sort: each element must make the same number of swaps to be moved into position because only adjacent elements are swapped.

15.26 Algorithm. The Parity Transport Sort

1. Repeat steps 2–9 $n/2$ times.
2. Let $i = 0$.
3. Repeat steps 4–5 while $i < n - 2$.
4. If $a_i > a_{i+1}$, swap them.
5. Increment i by 2.
6. Let $j = 1$.
7. Repeat steps 8–9 while $j < n - 3$.
8. If $a_j > a_{j+1}$, swap them.
9. Increment j by 2.

This has the same complexity as the Bubble Sort: each element must make the same number of swaps to be moved into position because only adjacent elements are swapped.

15.27 Algorithm. The Exchange Sort

1. Repeat steps 2–3 for $i = n - 1$ down to 1.
2. Repeat step 3 for $j = 0$ up to $i - 1$.
3. If $a_j > a_i$, swap them.

This has the same complexity as the Selection Sort, which is better than the Bubble Sort.

15.28 Algorithm. The Binary Insertion Sort

1. Do Steps 2–5 for $i = 1$ to $n-1$.
2. Hold the element a_i in a temporary space.
3. Use the Binary Search to find an index $j \le i$ for which $a_k \le a_i$ when $k \le j$ and for which $a_k \ge a_i$ when $j \le k \le i$.
4. Shift the subsequence $\{a_j, ..., a_{i-1}\}$ up one position, into $\{a_{j+1}, ..., a_i\}$.
5. Copy the held value of a_i into a_j.

This is more efficient than the Insertion Sort because the Binary Search makes frwer comparisons than the Sequential Search. However, it the same number of data movements.

15.29 Algorithm. The Binary Tree Sort

1. Do Step 2 for $i = 0$ to $n-1$.
2. Use Algorithm 13.2 to insert the element a_i into the BST T.
3. Use Algorithm 12.4 to traverse T, copying each element sequentially back into the sequence $\{a_0, ..., a_{n-1}\}$.

The average case for this algorithm is comparable to the Heap Sort. By Formula 13.3, insertions take about $1.39 \lg n$ steps, so steps 1–2 will take about $1.39 n \lg n = \Theta(n \lg n)$ steps. But in the worst case, it becomes $\Theta(n^2)$ because the BST degrades into a linked list. That is much worse than the Heap Sort. Of course, if an AVL tree or a red-black tree implementation is used, then this algorithm remains competetive with the Heap Sort even in the worst case.

15.30 There are $n!$ permutations of the a sequence of size n. Each one would require $n - 1$ comparisons to determine whether that permutation is the correct (sorted) sequence. Thus, the average complexity function is $n\, n!/2$. By Stirling's Formula (Formula C.10 on page 576), this is exponential: $O(2^n)$.

15.31 Algorithm. The Tournament Sort

1. Allocate an auxiliary sequence $\{b_i\}$ of size $2n$.
2. Do Step 3 for $i = 0$ up to $n-1$.
3. Copy a_i into b_{n+i}.
4. Do Steps 5–11 for $i = n-1$ down to 0.
5. Let $m = n$.
6. While $m > 1$, repeat steps 7–9.
7. Repeat step 8 for $j = m/2$ up to $m - 1$.
8. Copy the larger of b_{2j} and b_{2j+1} into b_j.
9. Divide m by 2.
10. Copy b_0 into a_i.
11. Find the value b_0 among $\{b_n, ..., b_{2n-1}\}$ and change it to $-\infty$.

This is comparable to the Bubble Sort. The loop in steps 6–9 makes n–1 comparisons and n–1 copies.

15.32 Among the ten sorting algorithms presented in this book, only the Selection Sort, the Quick Sort, and the Heap Sort are unstable. Moreover, the Selection Sort can easily be modified to make it stable.

15.33 The Shell Sort, Merge Sort, Quick Sort, Bucket Sort, and Counting Sort are parallelizable.

15.34 Ann's program crashes because it is using the Quick Sort on a large array that contains at least 99,000 duplicates. That array is nearly sorted already, causing the Quick Sort to run in nearly $\Theta(n^2)$ time.

15.35 The main loop of the Selection Sort iterates $n - 1$ times. On iteration i, it finds the maximum element among the subsequence $\{a_0, ..., a_i\}$ and then swaps it with a_i. That requires searching through all $i + 1$ elements of the subsequence. Thus the total number of comparisons made is $C = (n - 1) + (n - 1) + ... + 2 + 1 = n(n - 1)/2 = \Theta(n^2)$. [See Formula C.6 on page 572.]

Programming Problems

15.1

```
package chap15.prob01;
import chap03.list08.IntArrays;

public class TestBubbleSort {
  public TestBubbleSort() {
    int[] a = {66, 33, 99, 88, 44, 55, 22, 77};
    IntArrays.print(a);
    sort(a);
  }

  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--)
      for (int j = 0; j < i; j++)
        if (a[j] > a[j+1]) {
          IntArrays.swap(a, j, j+1);
          IntArrays.print(a);
        }
  }

  public static void main(String[] args) {
    new TestBubbleSort();
  }
}
```

15.2

```
package chap15.prob02;
import chap03.list08.IntArrays;

public class TestSelectionSort {
  public TestSelectionSort() {
    int[] a = {66, 33, 99, 88, 44, 55, 22, 77};
    IntArrays.print(a);
    sort(a);
  }

  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--) {
      int m=0;
      for (int j = 1; j <= i; j++) {
        if (a[j] > a[m]) m = j;
      }
      IntArrays.swap(a, i, m);
```

```
      IntArrays.print(a);
    }
  }

  public static void main(String[] args) {
    new TestSelectionSort();
  }
}
```

15.3

```
  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--)
      bubbleUp(a, i);
  }

  void bubbleUp(int[] a, int i) {
      for (int j = 0; j < i; j++)
        if (a[j] > a[j+1])
          IntArrays.swap(a, j, j+1);
  }
```

15.4

```
  public TestBubbleSort() {
    int[] a = IntArrays.randomInts(N, N);
    System.out.println("sorted: " + IntArrays.isSorted(a));
    sort(a);
    System.out.println("sorted: " + IntArrays.isSorted(a));
  }
```

15.5

```
package chap15.prob05;
import chap03.list08.IntArrays;

public class TestBubbleSort {
  final int N = 4000;

  public TestBubbleSort() {
    int[] a = IntArrays.randomInts(N, N);
    System.out.println("sorted: " + IntArrays.isSorted(a));
    sort(a);
    System.out.println("sorted: " + IntArrays.isSorted(a));
  }

  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--)
      for (int j = 0; j < i; j++)
        if (a[j] > a[j+1])
          IntArrays.swap(a, j, j+1);
```

```
  }

  public static void main(String[] args) {
      new TestBubbleSort();
  }
}
```

15.6

```
  void sort(int[] a) {
    boolean done = false;
    for (int i = a.length-1; i > 0 && !done; i--) {
      done = true;
      for (int j = 0; j < i; j++) {
        if (a[j] > a[j+1]) {
          IntArrays.swap(a, j, j+1);
          done = false;
        }
      }
    }
  }
```

15.7

```
package chap15.prob07;
import chap03.list08.IntArrays;

public class TestBubbleSort {
  final int N = 4000;

  public TestBubbleSort() {
    int[] a = IntArrays.randomInts(N, N);
    System.out.println("sorted: " + IntArrays.isSorted(a));
    System.out.println("sort1() time: " + sort1(a));
    System.out.println("sorted: " + IntArrays.isSorted(a));
    a = IntArrays.randomInts(N, N);
    System.out.println("sorted: " + IntArrays.isSorted(a));
    System.out.println("sort2() time: " + sort2(a));
    System.out.println("sorted: " + IntArrays.isSorted(a));
  }

  long sort1(int[] a) {
    long t1 = System.currentTimeMillis();
    boolean done = false;
    for (int i = a.length-1; i > 0 && !done; i--) {
      done = true;
      for (int j = 0; j < i; j++) {
        if (a[j] > a[j+1]) {
```

```
            IntArrays.swap(a, j, j+1);
            done = false;
          }
        }
      }
      long t2 = System.currentTimeMillis();
      return t2 - t1;
    }

    long sort2(int[] a) {
      long t1 = System.currentTimeMillis();
      boolean done = false;
      for (int i = a.length-1; i > 0 && !done; i--) {
        done = true;
        for (int j = 0; j < i; j++) {
          if (a[j] > a[j+1]) {
            IntArrays.swap(a, j, j+1);
            done = false;
          }
        }
      }
      long t2 = System.currentTimeMillis();
      return t2 - t1;
    }

    public static void main(String[] args) {
        new TestBubbleSort();
    }
  }
```

15.8

```
  void sort(int[] a) {
      sort(a, a.length);
  }

  void sort(int[] a, int n) {
    if (n < 1) return;
    for (int j = 0; j < n-1; j++)
      if (a[j] > a[j+1]) IntArrays.swap(a, j, j+1);
    sort(a, n-1);
  }
```

15.9

```
  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--) {
      for (int j = 0; j < i; j++) {
```

```
      if (a[j] > a[j+1]) IntArrays.swap(a, j, j+1);
    if (!invariant1(a, j))
      throw new RuntimeException("INVARIANT 1 VIOLATED");
    }
    if (!invariant2(a, i))
      throw new RuntimeException("INVARIANT 2 VIOLATED");
  }
}

boolean invariant1(int[] a, int j) {
  for (int k=0; k<j; k++)
    if (a[j+1] < a[k]) return false;
  return true;
}

boolean invariant2(int[] a, int i) {
  for (int k=0; k<i; k++)
    if (a[i] < a[k]) return false;
  for (int k=i; k<a.length-1; k++)
    if (a[k+1] < a[k]) return false;
  return true;
}
```

15.10

```
  void sort(int[] a) {
    for (int i = a.length-1; i > 0; i--) {
    int m=0;
    for (int j = 1; j <= i; j++) {
      if (a[j] > a[m]) m = j;
      // INVARIANT: a[m] = max{a[0], ..., a[j]};
    }
    IntArrays.swap(a, i, m);
    // INVARIANTS: a[i] = max{a[0], ..., a[i]};
    //             a[i] <= ... <= a[a.length-1];
  }
}
```

15.11

```
void sort(int[] a) {
  for (int i = a.length-1; i > 0; i--)
    select(a, i);
}

void select(int[] a, int i) {
  // moves max{a[0], ..., a[i]} into a[i]:
  int m=0;
```

```
    for (int j = 1; j <= i; j++) {
      if (a[j] > a[m]) m = j;
    }
    IntArrays.swap(a, i, m);
  }
```

15.12

```
  void sort(int[] a) {
    sort(a, a.length);
  }
  void sort(int[] a, int n) {
    if (n < 2) return;
    int m=0;
    for (int j=1; j<n; j++)
      if (a[j] > a[m]) m = j;
    IntArrays.swap(a, n-1, m);
    sort(a, n-1);
  }
```

15.13

```
  void sort(int[] a) {
    for (int i=1; i<a.length; i++)
      insert(a, i);
  }
  void insert(int[] a, int i) {
  // inserts a[i] into a[0..i] leaving it in ascending order:
    int ai=a[i], j=i;
    for (j=i; j>0 && a[j-1]>ai; j--)
      a[j] = a[j-1];
    a[j] = ai;
  }
```

15.14

```
  void sort(int[] a) {
    sort(a, a.length);
  }
  void sort(int[] a, int n) {
    if (n < 2) return;
    sort(a, n-1);
    int ai=a[n-1], j=n-1;
    for (j=n-1; j>0 && a[j-1]>ai; j--)
      a[j] = a[j-1];
    a[j] = ai;
  }
```

15.15

```
package chap15.prob15;
import chap03.list08.IntArrays;

public class TestMergeSort {
```

```
  public TestMergeSort(int size) {
    int[] a = IntArrays.randomInts(size, 100);
    IntArrays.print(a);
    System.out.println("sorted: " + IntArrays.isSorted(a));
    sort(a, 0, a.length);
    IntArrays.print(a);
    System.out.println("sorted: " + IntArrays.isSorted(a));
  }

  void sort(int[] a, int p, int q) {
    //  PRECONDITION: 0 < p < q <= a.length
    //  POSTCONDITION: a[p..q-1] is in ascending order
    if (q-p < 2) return;
    int m = (p+q)/2;
    sort(a, p, m);
    sort(a, m, q);
    merge(a, p, m, q);
  }

  void merge(int[] a, int p, int m, int q) {
    // PRECONDITIONS: a[p..m-1] and a[m..q-1] are in ascending order;
    // POSTCONDITION: a[p..q-1] is in ascending order;
    if (a[m-1] <= a[m]) return;  // a[p..q-1] is already sorted
    int i = p, j = m, k = 0;
    int[] aa = new int[q-p];
    while (i < m && j < q)
      if (a[i]<a[j]) aa[k++] = a[i++];
      else aa[k++] = a[j++];
    if (i < m) System.arraycopy(a,i,a,p+k,m-i); //shift a[i..m-1]
    System.arraycopy(aa,0,a,p,k);  // copy aa[0..k-1] to a[p..p+k-1];
  }

  public static void main(String[] args) {
    for (int i=10; i<=20; i++)
      new TestMergeSort(i);
  }
}
```

15.16 **a.**

```
void sort(int[] a, int p, int q) {
  for (int i = q-1; i > p; i--)
    for (int j = p; j < i; j++)
      if (a[j] > a[j+1]) IntArrays.swap(a, j, j+1);
}
```

b.
```
void sort(int[] a, int p, int q) {
  for (int i = q-1; i > p; i--) {
    int m=0;
    for (int j = 1; j <= i; j++)
      if (a[j] > a[m]) m = j;
    IntArrays.swap(a, i, m);
  }
}
```

c.
```
void sort(int[] a, int p, int q) {
  for (int i=p+1; i<q; i++) {
    int ai=a[i], j=i;
    for (j=i; j>p && a[j-1]>ai; j--)
      a[j] = a[j-1];
    a[j] = ai;
  }
}
```

15.17
```
package chap15.prob17;
abstract class AbstractTester {
  protected int[] a;
  private java.util.Random random;
  private final int N = 100;

  public AbstractTester() {  // initializes a[] and random
    a = new int[N];
    random = new java.util.Random();
    for (int i=0; i<N; i++)
      a[i] = 100 + random.nextInt(900);
  }

  public void randomize() {  // randomly permutes a[]
    for (int i=0; i<N; i++) {
      int j = random.nextInt(N);
      swap(a, i, j);
    }
  }

  public void reverse() {  // reverses the order of a[]
    for (int i=0; i<N/2; i++)
      swap(a, i, N-i);
  }

  abstract public void sort();  // sorts a[]
```

```
  public boolean isSorted() {
    for (int i=1; i<a.length; i++)
      if (a[i] < a[i-1]) return false;
    return true;
  }

  public void swap(int[] a, int i, int j) {
    int ai = a[i], aj = a[j];
    if (ai==aj) return;
    a[i] = aj;
    a[j] = ai;
  }
}
```

Chapter 16
Graphs

Exercises

16.1 If e is an edge between vertex i and veretex j in an undirected graph, then there is one "1" at location a_{ij} and a one "1" at location a_{ji} in the adjacency matrix. Thus, there are two 1s for each edge.

16.2 If e is an edge between vertex i and veretex j in an undirected graph, then there is a node "to j" in list i and a node "to i" in list j in the adjacency list. Thus, there are two nodes for each edge.

16.3

a. $n = 6$

b. $V = \{A, B, C, D, E, F\}$

c. $E = \{AB, BC, BD, CD, CE, CF, DE, DF\}$

d.

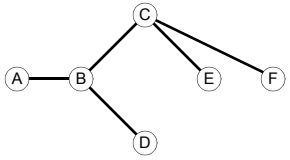
x:	A	B	C	D	E	F
$d(x)$	1	3	4	4	2	2

e. ABCD

f. ABCEDF

g. BCEDB

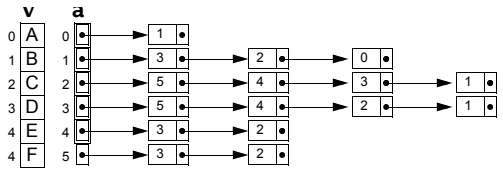
h.



i.

	A	B	C	D	E	F
A	0	1	0	0	0	0
B	1	0	1	1	0	0
C	0	1	0	1	1	1
D	0	1	1	0	1	1
E	0	0	1	1	0	1
F	0	0	1	1	0	0

j.



16.4

a. $n = 6$

b. $V = \{A, B, C, D, E, F\}$

c. $E = \{AD, BA, BD, CB, CD, CE, CF, DE, EC, FE\}$

d.

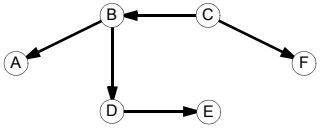
x:	A	B	C	D	E	F
$d(x)$:	1	2	4	1	1	1

e. ADEC

f. FCBADE

g. BDECB

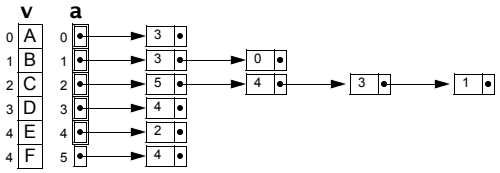
h.



i.

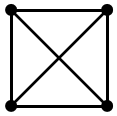
	A	B	C	D	E	F
A	0	0	0	1	0	0
B	1	0	0	1	0	0
C	0	1	0	1	1	1
D	0	0	0	0	1	0
E	0	0	1	0	0	0
F	0	0	0	0	1	0

j.

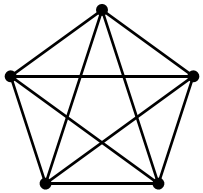


16.5

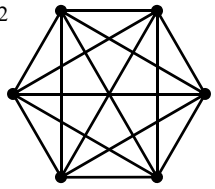
a. K_4



K_5



K_2

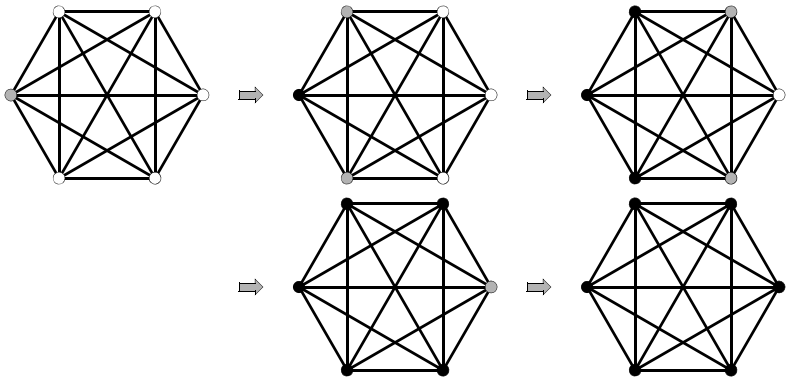


b. Each vertex of K_n has $n - 1$ adjacent vertices. So each vertex is an endpoint of $n - 1$ edges. There are n vertices. So there are $n(n - 1)$ edge endpoints. Each edge has two endpoints. So the number of edges is $n(n - 1)/2$.

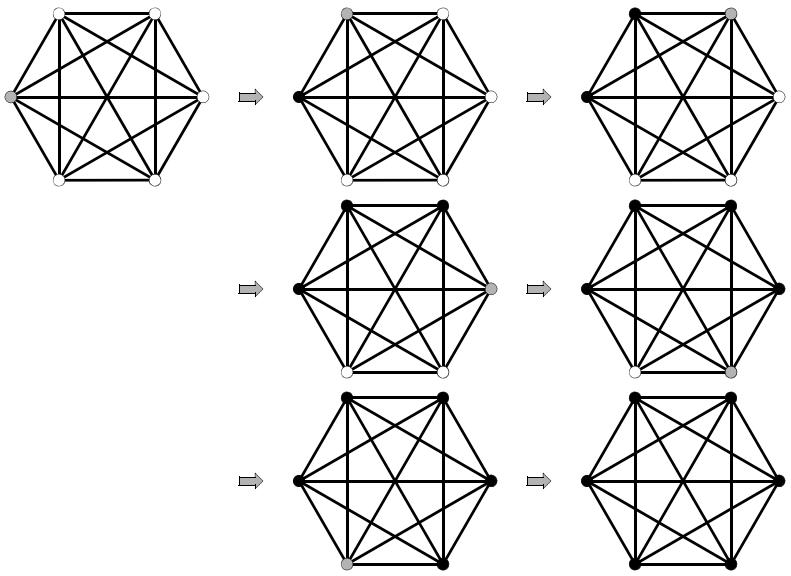
c. The adjacency matrix for K_n is an n-by-n matrix with 0s on the diagonal and 1s everywhere else.

d. The adjacency list for K_n is an n-component array `a[]` of lists. The list at component `a[i]` has $n - 1$ nodes: one for each vertex except vertex `i`.

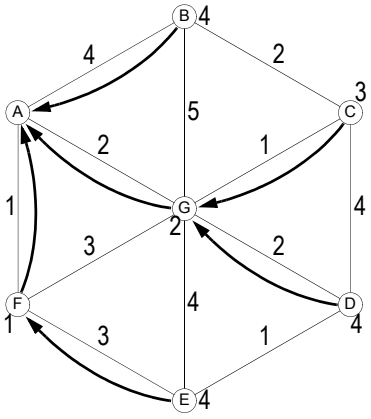
e.



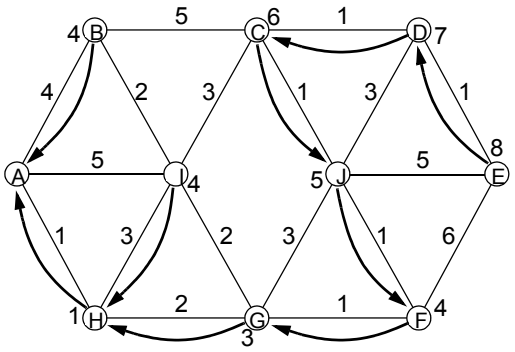
f.



16.6



16.7



16.8 **a.** Preorder traversal.
b. Level order traversal.

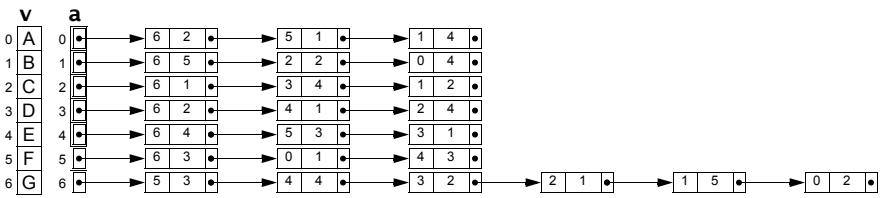
16.9 **a.**

	A	B	C	D	E	F	G
A	0	4	0	0	0	1	2
B	4	0	2	0	0	0	5
C	0	2	0	4	0	0	1
D	0	0	4	0	1	0	2
E	0	0	0	2	0	3	4
F	1	0	0	0	3	0	3
G	2	5	1	2	4	3	0

b.

	A	B	C	D	E	F	G
A	0	1	0	0	0	1	1
B	1	0	1	0	0	0	1
C	0	1	0	1	0	0	1
D	0	0	1	0	1	0	1
E	0	0	0	1	0	1	1
F	1	0	0	0	1	0	1
G	1	1	1	1	1	1	0

c.



Programming Problems

16.1

```
public Graph(String[] args) {
  size = args.length;
  vertices = new String[size];
  size = args.length;
  vertices = new String[size];
  for (int i=0; i<size; i++) {
    String ai = args[i];
    int j=i;
    while (j>0 && vertices[j-1].compareTo(ai)>0) {
      vertices[j] = vertices[j-1];
      --j;
    }
    vertices[j] = ai;
  }
  a = new boolean[size][size];
}
private int index(String v) {
  int p=0, q=size;
  while (p < q) {
    int i = (p + q)/2;
    if (vertices[i].equals(v)) return i;
    if (vertices[i].compareTo(v) < 0) p = i;
    else q = i;
  }
  if (vertices[p].equals(v)) return p;
  return a.length;
}
```

16.2

```
public boolean equals(Object object) {
  if (object == this) return true;
  if (!(object instanceof Graph)) return false;
  Graph that = (Graph)object;
  if (that.size != this.size) return false;
  String[] thisVertices = new String[size];
  System.arraycopy(this.vertices, 0, thisVertices, 0, size);
  Arrays.sort(thisVertices);
  String[] thatVertices = new String[size];
  System.arraycopy(that.vertices, 0, thatVertices, 0, size);
  Arrays.sort(thatVertices);
  if (!(Arrays.equals(thatVertices, thisVertices))) return false;
  for (int i=0; i<size; i++) {
```

```
      String vi = vertices[i];
      int i1 = this.index(vi);
      int i2 = that.index(vi);
      for (int j=0; j<size; j++) {
        String vj = vertices[i];
        int j1 = this.index(vj);
        int j2 = that.index(vj);
        if (this.a[i1][j1] != that.a[i2][j2]) return false;
      }
    }
    return true;
  }
```

16.3

```
public class Graph {
  private int size;
  private List[] a;  // adjacency list
  private String[] v;  // vertex list

  public Graph(String[] args) {
    size = args.length;
    a = new List[size];
    for (int i=0; i<size; i++)
      a[i] = new List();
    v = new String[size];
    System.arraycopy(args, 0, v, 0, size);
  }

  public void add(String v, String w) {
    a[index(v)].add(index(w));
    a[index(w)].add(index(v));
  }

  public String toString() {
    if (size == 0) return "{}";
    StringBuffer buf = new StringBuffer("{" + v[0] + ":" + a[0]);
    for (int i = 1; i < size; i++)
      buf.append("," + v[i] + ":" + a[i]);
    return buf + "}";
  }

  private int index(String vertex) {
    for (int i = 0; i < size; i++)
      if (vertex.equals(v[i])) return i;
```

```
    return a.length;
  }

  private class List {
    String vertex;
    Node edges;

    public void add(int j) {
      edges = new Node(j, edges);
    }

    public String toString() {
      if (edges==null) return "";
      StringBuffer buf = new StringBuffer();
      for (Node p = edges; p != null; p = p.next)
        buf.append(Graph.this.v[p.to]);
      return buf.toString();
    }

    private class Node {
      int to;
      Node next;
      Node(int to, Node next) {
        this.to = to;
        this.next = next;
      }
    }
  }
}
```

16.4

```
public boolean[][] getAdjacencyMatrix() {
  return a;
}
public String[] getVertices() {
  return vertices;
}
public int size() {
  return size;
}
```

16.5

```
public int[] degrees() {
  int[] d = new int[size];
  for (int i=0; i<size; i++) {
    d[i]=0;
```

```
    for (int j=0; j<size; j++)
      if (a[i][j]) ++d[i];
  }
  return d;
}
```

16.6

```
public int[] degrees() {
  int[] d = new int[size];
  for (int i=0; i<size; i++) {
    d[i]=0;
    for (List.Node p=a[i].edges; p != null; p = p.next)
      ++d[i];
  }
  return d;
}
```

16.7

```
public boolean isComplete() {
  for (int i=0; i<size; i++)
    for (int j=0; j<size; j++)
      if (j != i && !a[i][j]) return false;
  return true;
}
```

16.8

```
public boolean isComplete() {
  for (int i=0; i<size; i++) {
    boolean[] e = new boolean[size];
    for (List.Node p=a[i].edges; p != null; p = p.next)
      e[p.to] = true;
    for (int j=0; j<size; j++)
      if (j != i && !e[j]) return false;
  }
  return true;
}
```

16.9

```
public Graph(boolean[][] aa) {
  size = aa.length;
  a = new List[size];
  for (int i=0; i<size; i++)
    a[i] = new List();
  v = new String[size];
  for (int i=0; i<size; i++)
    v[i] = String.valueOf((char)('A'+i));
  for (int i=0; i<size; i++)
    for (int j=0; j<size; j++)
      if (aa[i][j]) a[i].add(j);
}
```

16.10
```
public Graph(String[] v, String[] e) {
  size = v.length;
  vertices = v;
  a = new boolean[size][size];
  for (int i=0; i<size; i++) {
    StringTokenizer tok = new StringTokenizer(e[i], " ,");
    while (tok.hasMoreElements()) {
      int j = Integer.parseInt(tok.nextToken());
      a[i][j] = true;
    }
  }
}
```

16.11
```
public boolean remove(String v, String w) {
  int i = index(v);
  if (i < 0) return false;
  int j = index(w);
  if (j < 0) return false;
  if (!a[i][j] || !a[j][i]) return false;
  a[i][j] = a[j][i] = false;
  return true;
}
```

16.12
```
public boolean remove(String v, String w) {
  boolean found = false;
  int i = index(v);
  if (i < 0) return false;
  int j = index(w);
  if (j < 0) return false;
  List.Node p=a[i].edges;
  if (p.to == j) {
    a[i].edges = p.next;
    found = true;
  } else {
    while (p.next != null) {
      if (p.next.to == j) {
        p.next = p.next.next;
        found = true;
        break;
      }
      p = p.next;
    }
  }
  List.Node q=a[j].edges;
```

```
    if (q.to == i) {
      a[j].edges = q.next;
      found = true;
    } else {
      while (q.next != null) {
        if (q.next.to == i) {
          q.next = q.next.next;
          found = true;
          break;
        }
        q = q.next;
      }
    }
    return found;
  }
```