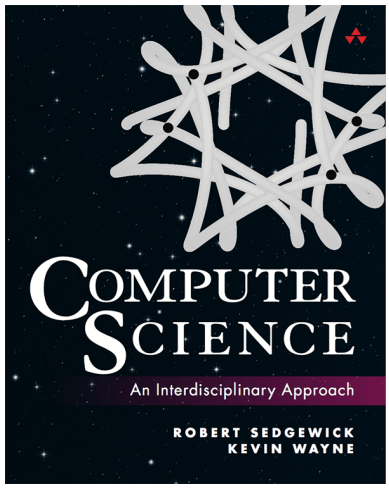
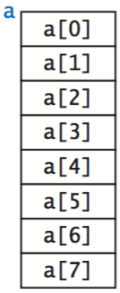


1.4 Arrays

In this section, we consider a fundamental construct known as the *array*. An array stores a sequence of values that are all of the same type. We want not just to store values but also to be able to quickly access each individual value. The method that we use to refer to individual values in an array is to number and then *index* them—if we have n values, we think of them as being numbered from 0 to $n-1$.

Arrays in Java.



Making an array in a Java program involves three distinct steps:

- Declare the array name.
- Create the array.
- Initialize the array values.

We refer to an array element by putting its index in square brackets after the array name: the code a[i] refers to element `i` of array `a[]`. For example, the following code makes an array of n numbers of type double, all initialized to 0:

```
double[] a;                    // declare the array
a = new double[n];             // create the array
for (int i = 0; i < n; i++)    // elements are indexed from 0 to n-1
   a[i] = 0.0;                 // initialize all elements to 0.0
```

Typical array-processing code.

ArrayExamples.java contains typical examples of using arrays in Java.

create an array with random values	`double[] a = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  a[i] = Math.random();`

print the array values, one per line	`for (int i = 0; i < n; i++)` `  System.out.println(a[i]);`

find the maximum of the array values	`double max = Double.NEGATIVE_INFINITY;`
	`for (int i = 0; i < n; i++)`
	`  if (a[i] > max) max = a[i];`

compute the average of the array values	`double sum = 0.0;`
	`for (int i = 0; i < n; i++)`
	`  sum += a[i];`
	`double average = sum / n;`

reverse the values within an array	`for (int i = 0; i < n/2; i++)`
	`{`
	`  double temp = a[i];`
	`  a[i] = a[n-1-i];`
	`  a[n-i-1] = temp;`
	`}`

copy sequence of values to another array	`double[] b = new double[n];`
	`for (int i = 0; i < n; i++)`
	`  b[i] = a[i];`

Programming with arrays.

Before considering more examples, we consider a number of important characteristics of programming with arrays.

- *Zero-based indexing*. We always refer to the first element of an array `a[]` as `a[0]`, the second as `a[1]`, and so forth. It might seem more natural to you to refer to the first element as `a[1]`, the second value as `a[2]`, and so forth, but starting the indexing with 0 has some advantages and has emerged as the convention used in most modern programming languages.
- *Array length*. Once we create an array, its length is fixed. You can refer to the length of an `a[]` in your program with the code `a.length`.

- *Default array initialization*. For economy in code, we often take advantage of Java's default array initialization convention. For example, the following statement is equivalent to the four lines of code at the top of this page:

```
double[] a = new double[n];
```

  The default initial value is 0 for all numeric primitive types and `false` for type `boolean`.
- *Memory representation*. When you use `new` to create an array, Java reserves space in memory for it (and initializes the values). This process is called *memory allocation*.
- *Bounds checking*. When programming with arrays, you must be careful. It is your responsibility to use legal indices when accessing an array element.
- *Setting array values at compile time*. When we have a small number of literal values that we want to keep in array, we can initialize it by listing the values between curly braces, separated by a comma. For example, we might use the following code in a program that processes playing cards.

```
String[] SUITS = {
    "Clubs", "Diamonds", "Hearts", "Spades"
};

String[] RANKS = {
    "2", "3", "4", "5", "6", "7", "8", "9", "10",
    "Jack", "Queen", "King", "Ace"
};
```

  After creating the two arrays, we might use them to print a random card name such as `Queen of Clubs`, as follows.

```
int i = (int) (Math.random() * RANKS.length);
int j = (int) (Math.random() * SUITS.length);
System.out.println(RANKS[i] + " of " + SUITS[j]);
```

- *Setting array values at run time*. A more typical situation is when we wish to compute the values to be stored in an array For example, we might use the following code to initialize an array of length 52 that represents a deck of playing cards, using the arrays `RANKS[]` and `SUITS[]` just defined.

```
String[] deck = new String[RANKS.length * SUITS.length];
for (int i = 0; i < RANKS.length; i++)
    for (int j = 0; j < SUITS.length; j++)
        deck[SUITS.length*i + j] = RANKS[i] + " of " + SUITS[j];
System.out.println(RANKS[i] + " of " + SUITS[j]);
```

Shuffling and sampling.

Now we describe some useful algorithms for rearranging the elements in an array.

- *Exchange*. Frequently, we wish to exchange two values in an array. Continuing our example with playing cards, the following code exchanges the card at position `i` and the card at position `j`:

```
String temp = deck[i];
deck[i] = deck[j];
deck[j] = temp;
```

- *Shuffling*. The following code shuffles our deck of cards:

```
int n = deck.length;
for (int i = 0; i < n; i++) {
   int r = i + (int) (Math.random() * (n-i));
   String temp = deck[r];
   deck[r] = deck[i];
   deck[i] = temp;
}
```

Proceeding from left to right, we pick a random card from `deck[i]` through `deck[n-1]` (each card equally likely) and exchange it with `deck[i]`. This code is more sophisticated than it might seem: see the textbook for details. Deck.java contains the full code for creating and shuffling a deck of cards.

- *Sampling without replacement*. In many situations, we want to draw a random sample from a set such that each member of the set appears at most once in the sample. Sample.java takes two command-line arguments `m` and `n`, and creates a *permutation* of length `n` whose first `m` entries comprise a random sample. See the textbook for details.

Precomputed values.

One simple application of arrays is to save values that you have computed, for later use. As an example, suppose that you are writing a program that performs calculations using small values of the harmonic numbers. One easy way to accomplish such a task is to save the values in an array with the following code

```
double[] harmonic = new double[n];
for (int i = 1; i < n; i++)
    harmonic[i] = harmonic[i-1] + 1.0/i;
```

and then simply use the code `harmonic[i]` to refer to any of the values. Precomputing values in this way in an example of a *space-time tradeoff*: by investing in space (to save the values) we save time (since we do not need to recompute them). This method is not effective if we need values for huge `n`, but it is very effective if we need a huge number of values for small `n`.

Simplifying repetitive code.

As an example of another simple application of arrays, consider the following code fragment, which prints the name of a month given its number (1 for January, 2 for February, and so forth):

```
if      (m ==  1) System.out.println("Jan");
else if (m ==  2) System.out.println("Feb");
else if (m ==  3) System.out.println("Mar");
else if (m ==  4) System.out.println("Apr");
else if (m ==  5) System.out.println("May");
else if (m ==  6) System.out.println("Jun");
else if (m ==  7) System.out.println("Jul");
else if (m ==  8) System.out.println("Aug");
else if (m ==  9) System.out.println("Sep");
else if (m == 10) System.out.println("Oct");
else if (m == 11) System.out.println("Nov");
else if (m == 12) System.out.println("Dec");
```

We could also use a `switch` statement, but a much more compact alternative is to use an array of strings consisting of the names of each month:

```
String[] MONTHS = {
    "", "Jan", "Feb", "Mar", "Apr", "May", "Jun",
    "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
};
```

```
...
System.out.println(MONTHS[m]);
```

This technique would be especially useful if you needed to access the name of a month by its number in several different places in your program. Note that we intentionally waste one slot in the array (element 0) to make `MONTHS[1]` correspond to January, as required.

Coupon collection

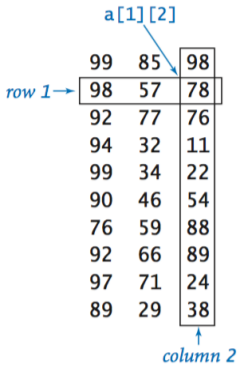
Coupon collector.

Suppose that you have a shuffled deck of cards and you turn them face up, one by one. How many cards do you need to turn up before you have seen one of each suit? This is an example of the famous *coupon collector* problem. In general, suppose that a trading card company issues trading cards with n different possible cards: how many do you have to collect before you have all n possibilities, assuming that each possibility is equally likely for each card that you collect? CouponCollector.java takes an integer command-line argument n and simulates this process. See the textbook for details.

Sieve of Eratosthenes.

The prime counting function $\pi(n)$ is the number of primes less than or equal to n. For example $\pi(17) = 7$ since the first seven primes are 2, 3, 5, 7, 11, 13, and 17. PrimeSieve.java takes an integer command-line argument n and computes $\pi(n)$ using the Sieve of Eratosthenes. See the textbook for details.

Two-dimensional arrays.



In many applications, a natural way to organize information is to use a table of numbers organized in a rectangle and to refer to rows and columns in the table. The mathematical abstraction corresponding to such tables is a *matrix*; the corresponding Java construct is a two-dimensional array.

- *Two-dimensional arrays in Java*. To refer to the element in row `i` and column `j` of a two-dimensional array `a[][]`, we use the notation `a[i][j]`; to declare a two-dimensional array, we add another pair of brackets; to create the array, we specify the number of rows followed by the number of columns after the type name (both within brackets), as follows:

  ```
  double[][] a = new double[m][n];
  ```

  We refer to such an array as an *m-by-n array*. By convention, the first dimension is the number of rows and the second dimension is the number of columns.

- *Default initialization*. As with one-dimensional arrays, Java initializes all entries in arrays of numbers to 0 and in arrays of booleans to `false`. Default initialization of two-dimensional arrays is useful because it masks more code than for one-dimensional arrays. To access each of the elements in a two-dimensional array, we need nested loops:

  ```
  double[][] a;
  a = new double[m][n];
  for (int i = 0; i < m; i++)
     for (int j = 0; j < n; j++)
        a[i][j] = 0;
  ```

- *Memory representation*. Java represents a two-dimensional array as an array of arrays. A matrix with `m` rows and `n` columns is actually an array of length `m`, each entry of which is an array of length `n`. In a

two-dimensional Java array, we can use the code `a[i]` to refer to the ith row (which is a one-dimensional array). Enables ragged arrays.

- *Setting values at compile time*. The following code initializes the 11-by-4 array `a[][]`:

```
double[][] a = {
    { 99.0, 85.0, 98.0, 0.0 },
    { 98.0, 57.0, 79.0, 0.0 },
    { 92.0, 77.0, 74.0, 0.0 },
    { 94.0, 62.0, 81.0, 0.0 },
    { 99.0, 94.0, 92.0, 0.0 },
    { 80.0, 76.5, 67.0, 0.0 },
    { 76.0, 58.5, 90.5, 0.0 },
    { 92.0, 66.0, 91.0, 0.0 },
    { 97.0, 70.5, 66.5, 0.0 },
    { 89.0, 89.5, 81.0, 0.0 },
    {  0.0,  0.0,  0.0, 0.0 }
};
```

- *Ragged arrays*. There is no requirement that all rows in a two-dimensional array have the same length—an array with rows of nonuniform length is known as a *ragged array*. The possibility of ragged arrays creates the need for more care in crafting array-processing code. For example, this code prints the contents of a ragged array:

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

- *Multidimensional arrays*. The same notation extends to arrays that have any number of dimensions. For instance, we can declare and initialize a three-dimensional array with the code

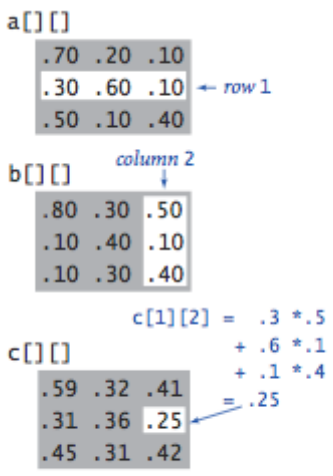
```
double[][][] a = new double[n][n][n];
```

and then refer to an entry with code like `a[i][j][k]`.

Matrix operations.

Typical applications in science and engineering involve implementing various mathematical operations with matrix operands. For example, we can *add* two *n*-by-n matrices as follows:

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```



Matrix multiplication

Similarly, we can *multiply* two matrices. Each entry `c[i][j]` in the product of `a[]` and `b[]` is computed by taking the dot product of row `i` of `a[]` with column `j` of `b[]`.

```
double[][] c = new double[n][n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++)  {
        for (int k = 0; k < n; k++)  {
            c[i][j] += a[i][k]*b[k][j];
        }
    }
}
```

Self-avoiding walk.

SelfAvoidingWalk.java is an application of two-dimensional arrays to chemistry. See textbook for details.

Exercises

2. Describe and explain what happens when you try to compile a program HugeArray.java with the following statement:

```
int n = 1000;
int[] a = new int[n*n*n*n];
```

4. Write a code fragment that reverses the order of values in a one-dimensional string array. Do not create another array to hold the result. *Hint*: Use the code in the text for exchanging two elements.

   Solution.

```
int n = a.length;
for (int i = 0; i < n/2; i++) {
    String temp = a[n-i-1];
    a[n-i-1] = a[i];
    a[i] = temp;
}
```

5. What is wrong with the following code fragment?

```
int[] a;
for (int i = 0; i < 10; i++)
   a[i] = i * i;
```

   Solution: It does not allocate memory for `a[]` with `new`. The code results in a `variable might not have been initialized` compile-time error.

9. What does the following code fragment print?

```
int[] a = { 1, 2, 3 };
int[] b = { 1, 2, 3 };
System.out.println(a == b);
```

   Solution: It prints false. The == operator compares whether the (memory addresses of the) two arrays are identical, not whether their corresponding values are equal.

10. Write a program Deal.java that takes an integer command-line argument `n` and prints `n` poker hands (five cards each) from a shuffled deck, separated by blank lines.

11. Write a program HowMany.java that takes a variable number of command-line arguments and prints how many there are.
12. Write a program DiscreteDistribution.java that takes a variable number of integer command-line arguments and prints the integer `i` with probability proportional to the `i`th command-line argument.
14. Write a code fragment Transpose.java to transpose a square two-dimensional array in place without creating a second array.

Creative Exercises

25. **Bad shuffling.** Suppose that you choose a random integer between 0 and n-1 in our shuffling code instead of one between i and n-1. Show that the resulting order is not equally likely to be one of the n! possibilities. Run the test of the previous exercise for this version.

    Partial solution: when n = 3, all 3! = 6 outcomes are possible, but some are more likely:

ABC	ACB	BAC	BCA	CAB	CBA
4/27	5/27	6/27	4/27	5/27	3/27

    Here's what happened to PlanetPoker when they used a broken shuffling algorithm that could only generate only about 200,000 of the possible 52! shuffles.

26. **Inverse permutation.** Write a program InversePermutation.java that reads in a permutation of the integers `0` to `n-1` from `n` command-line arguments and prints the *inverse permutation*. (If the permutation is in an array `a[]`, its *inverse* is the array `b[]` such that `a[b[i]] = b[a[i]] = i`.) Be sure to check that the input is a valid permutation.
29. **Hadamard matrix.** The n-by-n Hadamard H(n) matrix is a boolean matrix with the remarkable property that any two rows differ in exactly n/2 bits. (This property makes it useful for designing *error-correcting codes*.) H(1) is a 1-by-1 matrix with the single entry true, and for n > 1, H(2n) is obtained by aligning four copies of H(n) in a large square, and then inverting all of the entries in the lower right n-by-n copy, as shown in the following examples (with T representing true and F representing false, as usual).

```
H(1)  H(2)    H(4)
------------------
 T    T T   T T T T
      T 0   T 0 T 0
            T T 0 0
            T 0 0 T
```

    Write a program Hadamard.java that takes one command-line argument n and prints H(n). Assume that n is a power of 2.
36. **Random walkers.** Suppose that n random walkers, starting in the center of an n-by-n grid, move one step at a time, choosing to go left, right, up, or down with equal probability at each step. Write a program RandomWalkers.java to help formulate and test a hypothesis about the number of steps taken before all cells are touched.
38. **Birthday problem.** Suppose that people enter an empty room until a pair of people share a birthday. On average, how many people will have to enter before there is a match? Write a program Birthday.java to simulate one experiment. Write a program Birthdays.java to repeat the experiment many times and estimate the average value. Assume birthdays to be uniform random integers between 0 and 364.
41. **Binomial coefficients.** Write a program BinomialDistribution.java that builds and prints a two-dimensional ragged array a such that `a[n][k]` contains the probability that you get exactly k heads when you toss a coin n times. Take a command-line argument to specify the maximum value of n. These numbers are known as the *binomial distribution*: if you multiply each entry in row i by 2^n, you get the *binomial coefficients*—the coefficients of x^k in (x+1)^n—arranged in *Pascal's triangle*. To compute them, start with `a[n][0] = 0.0` for all n and `a[1][1] = 1.0`, then compute values in successive rows, left to right, with `a[n][k] = (a[n-1][k] + a[n-1][k-1]) / 2.`

```
Pascal's triangle   Binomial distribution
-----------------------------------------
1                   1
1 1                 1/2  1/2
1 2 1               1/4  1/2  1/4
1 3 3 1             1/8  3/8  3/8  1/8
1 4 6 4 1           1/16 1/4  3/8  1/4  1/16
```

Web Exercises

1. **Birthday problem.** Modify Birthday.java so that it compute the probability that two people have a birthday within a day of each other.
2. **Above average.** 90% of incoming college students rate themselves as above average. Write a program `AboveAverage.java` that takes a command-line argument n, reads in n integers from standard input, and prints the fraction of values that are strictly above the average value.
3. **Random permutation.** Write a program Permutation.java so that it takes a command-line argument N and prints a random permutation of the integers 0 through N-1. Also print a *checkerboard visualization* of the permutation. As an example, the permutation { 4, 1, 3, 0, 2 } corresponds to:

```
4 1 3 0 2
* * * Q *
* Q * * *
* * * * Q
* * Q * *
Q * * * *
```

4. **8 queens checker.** A permutation of the integer 0 to N-1 corresponds to a placement of queens on an N-by-N chessboard so that no two queens are in the same row or column. Write a program `QueensChecker.java` that determines whether or not a permutation corresponds to a placement of queens so that no two are in the same row, column, or *diagonal*. As an example, the permutation { 4, 1, 3, 0, 2 } is a legal placement:

```
* * * Q *
* Q * * *
* * * * Q
* * Q * *
Q * * * *
```

   Try to do it without using any extra arrays besides the length N input permutation `q`. *Hint*: to determine whether setting q[i] conflicts with q[j] for i < j.

   - if q[i] equals q[j]: two queens are placed in the same column
   - if q[i] - q[j] equals j - i: two queens are on same major diagonal
   - if q[j] - q[i] equals j - i: two queens are on same minor diagonal
5. **Finding your beer.** A large number of college students are attending a party. Each guest is drinking a can of beer (or soda of they are under 21). An emergency causes the lights to go out and the fire alarm to go off. The guests calmly put down their beer and exit the building. When the alarm goes off, they re-enter and try to retrieve their beer. However, the lights are still off, so each student randomly grabs a bottle of beer. What are the chances that at least one student gets his or her original beer? Write a program `MyBeer.java` that takes a command-line argument n and runs 1,000 simulations this event, assuming their are n guests. Print the fraction of times that at least one guest gets their original beer. As n gets large, does this fraction approach 0 or 1 or something in between?
6. **Linear feedback shift register.** Rewrite linear feedback shift register from Chapter 1 by using an array to streamline it and makes it more extensible, e.g., if the number of cells in the shift register increases.

Program LFSR.java uses a `boolean` *Hint*: use the `^` operator to take the exclusive or of two boolean values.

7. **Lockers.** Your are in a locker room with 100 open lockers, numbered 1 to 100. Toggle all of the lockers that are even. By *toggle*, we mean close if it is open, and open if it is closed. Now toggle all of the lockers that are multiples of three. Repeat with multiples of 4, 5, up to 100. How many lockers are open? *Answer*: lockers 1, 4, 9, 16, 25, ..., 100 will be open. Guess you don't need an array once you see the pattern.
8. **Scheduling with deadline.** Suppose that you have N tasks to schedule. Each task takes 1 unit of time and has a deadline by which time it is expected to finish. If a task is not completed by its deadline, you pay a $1,000 fine. Find a schedule that minimizes the penalty. *Hint*: schedule the tasks in order of their deadline, but don't bother with any task that won't finish by its deadline.
9. **Calendar.** Repeat Exercise 1.33 to produce a calendar for a given month and year. Use arrays to store the names of the days of the week, the names of the months, and the number of days in a month.
10. **Connect Four.** Given an N-by-N grid with each cell either occupied by an 'X', an 'O', or empty, write a program to find the longest sequence of consecutive 'X's either horizontal, vertically, or diagonally. To test your program, you can create a random grid where each cell contains an 'X' or 'O' with probability 1/3.
11. **Thai kickboxing**. Write a program KickBoxer.java that takes an integer weight w as a command line input and prints the corresponding kickboxing weight-class according to the table below.

```
weight class               from    to
-------------------------------------
Fly Weight                    0   112
Super Fly Weight            112   115
Bantam Weight",             115   118
Super Bantam Weight         118   122
Feather Weight              122   126
Super Feather Weight        126   130
Light Weight                130   135
Super Light Weight          135   140
Welter Weight               140   147
Super Welter Weight         147   154
Middle Weight               154   160
Super Middle Weight         160   167
Light Heavy Weight          167   174
Super Light Heavy Weight    174   183
Cruiser Weight              183   189
Super Cruiser Weight        189   198
Heavy Weight                198   209
Super Heavy Weight          209
```

Use an integer array to store the weight limits and a string array to store the weight categories (ranging from Flyweight to Super Heavyweight).

12. **N-ary counter**. Write a program that counts in base N from 0 to N^{20} - 1. Use an array of 20 elements.
13. **Terrain analysis.** Given an N-by-N grid of elevation values (in meters), a *peak* is a grid point for which all four neighboring cells are strictly lower. Write a code fragment that counts the number of peaks in a given N-by-N grid.
14. **Magic squares.** Write a program MagicSquare.java that reads in an odd integer N from the command line and prints out an N-by-N magic square. The square contains each of the integers between 1 and N^2 exactly once, such that all row sums, column sums, and diagonal sums are equal.

```
4  9  2     11 18 25  2  9
3  5  7     10 12 19 21  3
8  1  6      4  6 13 20 22
            23  5  7 14 16
            17 24  1  8 15
```

One simple algorithm is to assign the integers 1 to N^2 in ascending order, starting at the bottom, middle cell. Repeatedly assign the next integer to the cell adjacent diagonally to the right and down. If

this cell has already been assigned another integer, instead use the cell adjacently above. Use wrap-around to handle border cases.

15. **Banner.** Write a program `Banner.java` that takes a string as a command line argument and prints the string in large letters as below.

```
% java Banner "Kevin"
#    #  ######  #    #     #    #    #
#   #   #       #    #     #    ##   #
####    #####   #    #     #    # #  #
#  #    #       #    #     #    #  # #
#   #   #        #  #      #    #   ##
#    #  ######    ##       #    #    #
```

Mimics the Unix utility `banner`.

16. **Voting and social choice theory.** Plurality (US presidential election), run-off elections, sequential run-off elections (Australia, Ireland, Princeton faculty committees), Condorcet. Kemeny rank aggregation. Arrow's impossibility theorem. Same ideas for sports, google, meta-search, machine learning
17. **Borda count.** In 1781, Borda proposed a positional method for determining the outcome of a political election with K voters and N candidates. Each voter ranks the candidates in increasing order of preference (from 1 to N). Borda's method assigns a score to each candidate equal to the sum of their rankings. The candidate with the highest sum wins. This is used in Major League Baseball to determine the MVP.
18. **Kendall's tau distance.** Given two permutations, Kendall's tau distance is the number of pairs out of position. "Bubblesort metric." Useful in top-k lists. Optimal Kemeny rank aggregation in voting theory minimizes Kendall tau distance. Also useful for ranking genes using several expression profiles, ranking search engine results, etc.
19. **Spearman's footrule distance.** Given two permutations, Spearman's footrule distance is the L1 distance between the permutations as vectors. Useful in top-k lists.

```
int footrule = 0;
for (int i = 0; i < N; i++)
    footrule = footrule + Math.abs(p[i] - q[i]);
```

20. **US postal barcodes.** The POSTNET barcode is used by the US Postal System to route mail. Each decimal digit in the zip code is encoded using a sequence of 5 short and long lines for use by scanners as follows:

VALUE	ENCODING
0	||ıı
1	ıı||
2	ı|ı|
3	ı||ı
4	|ıı|
5	|ı|ı
6	||ıı
7	|ıı|
8	|ı|ı
9	|ı|ıı

A sixth checksum digit is appended: it is computed by summing up the original five digits mod 10. In addition, a long line is added to the beginning and appended to the end. Write a program ZipBarCoder.java that reads in a five digit zip code as the command line parameter and prints the corresponding postal barcode. Print the code vertically instead of horizontally, e.g, the following encodes 08540 (with the check digit of 7).

```
*****
*****
*****
**
**
**
*****
**
**
*****
**
**
*****
**
*****
**
**
*****
**
**
*****
*****
*****
**
**
**
*****
**
**
**
*****
*****
```

21. **US postal barcodes.** Repeat the previous exercise, but plot the output using Turtle graphics.
22. **Gaps with no primes.** Find the longest consecutive sequence of integers with no primes. Write a program PrimeGap.java that takes a command line parameter N and prints the largest block of integers between 2 and N with no primes.
23. **Goldbach conjecture.** In 1742, Christian Goldbach conjectured that every even number greater than 2 could be written as the sum of two primes. For example, 16 = 3 + 13. Write a program Goldbach.java that takes one command line parameter N and expresses N as the sum of two primes. Goldbach's conjecture is still unresolved, but it is known to be true for all $N < 10^{14}$.
24. **Minima in permutations.** Write a program that takes an integer n from the command line, generates a random permutation, prints the permutation, and prints the number of left-to-right minima in the permutation (the number of times an element is the smallest seen so far). Then write a program that takes integers m and n from the command line, generates m random permutations of length n, and prints the average number of left-to-right minima in the permutations generated. *Extra credit*: Formulate a hypothesis about the number of left-to-right minima in a permutation of length n, as a function of n.
25. **In-place inverse permutation.** Redo Exercise 1.4.25, but compute the permutation in-place, i.e., do not allocate a second array for the inverse permutation. *Caveat*: this is hard.
26. **Most likely roll.** Alice and Bob are in a heated argument about whether if they repeatedly roll a die until the sum is more than 12, is 13 the most likely sum? Write a program MostLikelyRoll.java to simulate the process a million times and produce a table of the fraction of times the sum is 13, 14, 15, 16, 17, and 18.
27. **Spiraling 2-D array.** Given a 2-D array, write a program Spiral.java to print it out in spiral order.

```
 1  2  3  4
 5  6  7  8
 9 10 11 12
13 14 15 16

1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
```

28. **Sudoko verifier.** Given a 9-by-9 array of integers between 1 and 9, check if it is a valid solution to a Sudoku puzzle: each row, column, and block should contain the 9 integers exactly once.

```
 5 3 4 | 6 7 8 | 9 1 2
 6 7 2 | 1 9 5 | 3 4 8
 1 9 8 | 3 4 2 | 5 6 7
-------+-------+------
 8 5 9 | 7 6 1 | 4 2 3
 4 2 6 | 8 5 3 | 7 9 1
 7 1 3 | 9 2 4 | 8 5 6
-------+-------+------
 9 6 1 | 5 3 7 | 2 8 4
 2 8 7 | 4 1 9 | 6 3 5
 3 4 5 | 2 8 6 | 1 7 9
```

29. **Sum of powers conjecture.** Redo Exercise 1.3.x, but precompute the 5th powers of all relevant integers. Evaluate how much time this saves. The program Euler.java searches for integral solutions to $a^5 + b^5 + c^5 + d^5 = e^5$.
30. **Haar wavelet transform.** Given, an array `a[]` of length 2^n, its 1D Haar transform is obtained as follows: Compute the average and difference of a[2i] and a[2i+1], and compute the array of the same length containing the averages, followed by the differences. Then apply the same technique to the averages (the first 2^n-1 entries) and so on. An example with 2^3 entries is shown below.

```
 448  768  704  640 1280 1408 1600 1600  (original)
 608  672 1344 1600 -160   32  -64    0  (step 1)
 640 1472  -32 -128 -160   32  -64    0  (step 2)
1056 -416  -32 -128 -160   32  -64    0  (step 3)
```

The *2D Haar wavelet transform* of a 2^n-by-2^n matrix, is obtained by applying the Haar wavelet transform to each row, and then to each column. The Haar wavelet transform is useful in signal processing, medical imaging, and data compression.

31. What happens when you try to compile a program with the following statement?

```
int[] a = new int[-17];
```

It compiles cleanly, but throws a `java.lang.NegativeArraySizeException` when you execute it.

32. **Blackjack.** Write a program Blackjack.java that takes three command line integers x, y, and z representing your two blackjack cards x and y, and the dealer's face-up card z, and prints the "standard strategy" for a 6 card deck in Atlantic city. Assume that x, y, and z are integers between 1 and 10, representing an ace through a face card. Report whether the player should hit, stand, or split according to these strategy tables. Encode the strategy tables using three 2-D boolean arrays.

    Modify `Blackjack.java` to allow *doubling*.

33. **Boltzmann distribution.** Here's a simple model to approximate the Boltzmann distribution from statistical physics: generate 100 random integers between 1 and 10. If the sum is exactly 200 keep this trial. Repeat this process until you get 1,000 trials that meet the criterion. Now plot a histogram of the number of times each of the 10 integers occurs.

34. **Doubly stochastic.** Write a program to read in an N-by-N matrix of real numbers and print `true` if the matrix is *doubly stochastic*, and `false` otherwise. A matrix is *stochastic* if all of the row and column sums are 1. Since you are dealing with floating point numbers, allow the sums to be between 1 - ε and 1 + ε where ε= 0.000000001.
35. Suppose that `b[]` is an array of 100 elements, with all entries initialized to 0, and that `a[]` is an array of N elements, each of which is an integer between 0 and 99. What is the effect of the following loop?

```
for (j = 0; j < N; j++)
   b[a[j]]++;
```

36. Modify RandomStudent.java so that it stores a parallel array of type boolean named `isFemale`, where element i is `true` if student i is female and `false` otherwise. Now, print one male student at random and one female student at random. *Hint*: use a `do-while` loop to generate random integers until you get one that indexes a male student.
37. Which of the following require using arrays. For each, the input comes from standard input and consists of N real numbers between 0.0 and 1.0.
    1. Print the maximum element.
    2. Print the maximum and minimum elements.
    3. Print the median element.
    4. Print the element that occurs most frequently.
    5. Print the sum of the squares of the elements.
    6. Print the average of the N elements.
    7. Print the element closest to 0.
    8. Print all the numbers greater than the average.
    9. Print the N elements in increasing order.
    10. Print the N elements in random order.
    11. Print histogram (with, say 10 bins of size 0.1).
38. Write a program Yahtzee.java that simulates the rolling of five dice and prints "Yahtzee" if all five dice are the same; otherwise it should print "Try again."
39. Modify DayOfWeek.java so that it reads in a date and print which day of the week that date falls on. Your program should take three command line arguments, M (month), D (day), and Y (year). Do not use any `if-else` statements; instead use a string array consisting of the names of the 7 days of the week.
40. Write a program Pascal.java to compute Pascal's triangle using a ragged array.
41. **Zero out matrix rows and columns.** Given an *m*-by-*n* integer matrix `a[][]`, if `a[i][j]` is 0, set row `i` and column `j` to 0. Do not use any extra arrays.

    Solution. First, check whether row 0 has a 0 and whether column 0 has a 0; record this information in two boolean variables. Next, for each element `a[i][j]` that is 0, set element `a[i][0]` and `a[0][j]` to 0. Finally, set `a[i][j]` to 0 if either `a[i][0]` or `a[0][j]`.

Last modified on March 06, 2019.