linear Relating to a straight line. Here, we talk about graphing how the time taken by an algorithm depends on the size of the data it is processing. Linear algorithms have straight-line graphs that can describe this relationship.

linear search A search that probes each item in a list or sequence, from first, until it finds what it is looking for. It is used for searching for a target in unordered lists of items.

Merge algorithm An efficient algorithm that merges two already sorted lists, to produce a sorted list result. The merge algorithm is really a pattern of computation that can be adapted and reused for various other scenarios, such as finding words that are in a book, but not in a vocabulary.

probe Each time we take a look when searching for an item is called a probe. In our chapter on *Iteration* we also played a guessing game where the computer tried to guess the user's secret number. Each of those tries would also be called a probe.

test-driven development (TDD) A software development practice which arrives at a desired feature through a series of small, iterative steps motivated by automated tests which are *written first* that express increasing refinements of the desired feature. (see the Wikipedia article on Test-driven development for more information.)

14.11 Exercises

1. The section in this chapter called Alice in Wonderland, again! started with the observation that the merge algorithm uses a pattern that can be reused in other situations. Adapt the merge algorithm to write each of these functions, as was suggested there:
   (a) Return only those items that are present in both lists.
   (b) Return only those items that are present in the first list, but not in the second.
   (c) Return only those items that are present in the second list, but not in the first.
   (d) Return items that are present in either the first or the second list.
   (e) Return items from the first list that are not eliminated by a matching element in the second list. In this case, an item in the second list "knocks out" just one matching item in the first list. This operation is sometimes called *bagdiff*. For example `bagdiff([5,7,11,11,11,12,13], [7,8,11])` would return `[5,11,11,12,13]`
2. Modify the queens program to solve some boards of size 4, 12, and 16. What is the maximum size puzzle you can usually solve in under a minute?
3. Adapt the queens program so that we keep a list of solutions that have already printed, so that we don't print the same solution more than once.
4. Chess boards are symmetric: if we have a solution to the queens problem, its mirror solution — either flipping the board on the X or in the Y axis, is also a solution. And giving the board a 90 degree, 180 degree, or 270 degree rotation is also a solution. In some sense, solutions that are just mirror images or rotations of other solutions — in the same family — are less interesting than the unique "core cases". Of the 92 solutions for

the 8 queens problem, there are only 12 unique families if you take rotations and mirror images into account. Wikipedia has some fascinating stuff about this.

(a) Write a function to mirror a solution in the Y axis,

(b) Write a function to mirror a solution in the X axis,

(c) Write a function to rotate a solution by 90 degrees anti-clockwise, and use this to provide 180 and 270 degree rotations too.

(d) Write a function which is given a solution, and it generates the family of symmetries for that solution. For example, the symmetries of `[0,4,7,5,2,6,1,3]` are

```
[[0,4,7,5,2,6,1,3],[7,1,3,0,6,4,2,5],
 [4,6,1,5,2,0,3,7],[2,5,3,1,7,4,6,0],
 [3,1,6,2,5,7,4,0],[0,6,4,7,1,3,5,2],
 [7,3,0,2,5,1,6,4],[5,2,4,6,0,3,1,7]]
```

(e) Now adapt the queens program so it won't list solutions that are in the same family. It only prints solutions from unique families.

5. Every week a computer scientist buys four lotto tickets. She always chooses the same prime numbers, with the hope that if she ever hits the jackpot, she will be able to go onto TV and Facebook and tell everyone her secret. This will suddenly create widespread public interest in prime numbers, and will be the trigger event that ushers in a new age of enlightenment. She represents her weekly tickets in Python as a list of lists:

```
my_tickets = [ [ 7, 17, 37, 19, 23, 43],
               [ 7,  2, 13, 41, 31, 43],
               [ 2,  5,  7, 11, 13, 17],
               [13, 17, 37, 19, 23, 43] ]
```

Complete these exercises.

(a) Each lotto draw takes six random balls, numbered from 1 to 49. Write a function to return a lotto draw.

(b) Write a function that compares a single ticket and a draw, and returns the number of correct picks on that ticket:

```
test(lotto_match([42,4,7,11,1,13], [2,5,7,11,13,17]) == 3)
```

(c) Write a function that takes a list of tickets and a draw, and returns a list telling how many picks were correct on each ticket:

```
test(lotto_matches([42,4,7,11,1,13], my_tickets) == [1,2,3,1])
```

(d) Write a function that takes a list of integers, and returns the number of primes in the list:

```
test(primes_in([42, 4, 7, 11, 1, 13]) == 3)
```

(e) Write a function to discover whether the computer scientist has missed any prime numbers in her selection of the four tickets. Return a list of all primes that she has missed:

```
test(prime_misses(my_tickets) == [3, 29, 47])
```

(f) Write a function that repeatedly makes a new draw, and compares the draw to the four tickets.

   i. Count how many draws are needed until one of the computer scientist's tickets has at least 3 correct picks. Try the experiment twenty times, and average out the number of draws needed.

   ii. How many draws are needed, on average, before she gets at least 4 picks correct?

   iii. How many draws are needed, on average, before she gets at least 5 correct? (Hint: this might take a while. It would be nice if you could print some dots, like a progress bar, to show when each of the 20 experiments has completed.)

Notice that we have difficulty constructing test cases here, because our random numbers are not deterministic. Automated testing only really works if you already know what the answer should be!

6. Read *Alice in Wonderland*. You can read the plain text version we have with this textbook, or if you have e-book reader software on your PC, or a Kindle, iPhone, Android, etc. you'll be able to find a suitable version for your device at http://www.gutenberg.org/. They also have html and pdf versions, with pictures, and thousands of other classic books!