EXERCISES

1.4.1 Show that the number of different triples that can be chosen from N items is precisely $N(N-1)(N-2)/6$. *Hint*: Use mathematical induction.

1.4.2 Modify `ThreeSum` to work properly even when the `int` values are so large that adding two of them might cause overflow.

1.4.3 Modify `DoublingTest` to use `StdDraw` to produce plots like the standard and log-log plots in the text, rescaling as necessary so that the plot always fills a substantial portion of the window.

1.4.4 Develop a table like the one on page 181 for `TwoSum`.

1.4.5 Give tilde approximations for the following quantities:

a. $N + 1$

b. $1 + 1/N$

c. $(1 + 1/N)(1 + 2/N)$

d. $2N^3 - 15N^2 + N$

e. $\lg(2N)/\lg N$

f. $\lg(N^2 + 1) / \lg N$

g. $N^{100} / 2^N$

1.4.6 Give the order of growth (as a function of N) of the running times of each of the following code fragments:

a.
```
int sum = 0;
for (int n = N; n > 0; n /= 2)
  for(int i = 0; i < n; i++)
     sum++;
```

b.
```
int sum = 0;
for (int i = 1 i < N; i *= 2)
    for (int j = 0; j < i; j++)
        sum++;
```

c.
```
int sum = 0;
for (int i = 1 i < N; i *= 2)
   for (int j = 0; j < N; j++)
       sum++;
```

1.4.7 Analyze `ThreeSum` under a cost model that counts arithmetic operations (and comparisons) involving the input numbers.

1.4.8 Write a program to determine the number pairs of values in an input file that are equal. If your first try is quadratic, think again and use `Arrays.sort()` to develop a linearithmic solution.

1.4.9 Give a formula to predict the running time of a program for a problem of size N when doubling experiments have shown that the doubling factor is 2^b and the running time for problems of size N_0 is T.

1.4.10 Modify binary search so that it always returns the element with the smallest index that matches the search element (and still guarantees logarithmic running time).

1.4.11 Add an instance method `howMany()` to `StaticSETofInts` (page 99) that finds the number of occurrences of a given key in time proportional to log N in the worst case.

1.4.12 Write a program that, given two sorted arrays of N `int` values, prints all elements that appear in both arrays, in sorted order. The running time of your program should be proportional to N in the worst case.

1.4.13 Using the assumptions developed in the text, give the amount of memory needed to represent an object of each of the following types:

a. `Accumulator`

b. `Transaction`

c. `FixedCapacityStackOfStrings` with capacity C and N entries

d. `Point2D`

e. `Interval1D`

f. `Interval2D`

g. `Double`