Exercises

1. Suppose you have algorithms with the five running times listed below. (Assume these are the exact running times.) How much slower do each of these algorithms get when you (a) double the input size, or (b) increase the input size by one?

   (a) n^2

   (b) n^3

   (c) $100n^2$

   (d) $n \log n$

   (e) 2^n

2. Suppose you have algorithms with the six running times listed below. (Assume these are the exact number of operations performed as a function of the input size n.) Suppose you have a computer that can perform 10^{10} operations per second, and you need to compute a result in at most an hour of computation. For each of the algorithms, what is the largest input size n for which you would be able to get the result within an hour?

   (a) n^2

   (b) n^3

   (c) $100n^2$

   (d) $n \log n$

   (e) 2^n

   (f) 2^{2^n}

3. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$f_1(n) = n^{2.5}$$
$$f_2(n) = \sqrt{2n}$$
$$f_3(n) = n + 10$$
$$f_4(n) = 10^n$$
$$f_5(n) = 100^n$$
$$f_6(n) = n^2 \log n$$

4. Take the following list of functions and arrange them in ascending order of growth rate. That is, if function $g(n)$ immediately follows function $f(n)$ in your list, then it should be the case that $f(n)$ is $O(g(n))$.

$$g_1(n) = 2^{\sqrt{\log n}}$$
$$g_2(n) = 2^n$$
$$g_4(n) = n^{4/3}$$
$$g_3(n) = n(\log n)^3$$
$$g_5(n) = n^{\log n}$$
$$g_6(n) = 2^{2^n}$$
$$g_7(n) = 2^{n^2}$$

5. Assume you have functions f and g such that $f(n)$ is $O(g(n))$. For each of the following statements, decide whether you think it is true or false and give a proof or counterexample.

(a) $\log_2 f(n)$ is $O(\log_2 g(n))$.

(b) $2^{f(n)}$ is $O(2^{g(n)})$.

(c) $f(n)^2$ is $O(g(n)^2)$.

6. Consider the following basic problem. You're given an array A consisting of n integers $A[1], A[2], \ldots, A[n]$. You'd like to output a two-dimensional n-by-n array B in which $B[i, j]$ (for $i < j$) contains the sum of array entries $A[i]$ through $A[j]$—that is, the sum $A[i] + A[i+1] + \cdots + A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)

Here's a simple algorithm to solve this problem.

```
For $i = 1, 2, \ldots, n$
  For $j = i + 1, i + 2, \ldots, n$
    Add up array entries $A[i]$ through $A[j]$
    Store the result in $B[i, j]$
  Endfor
Endfor
```

(a) For some function f that you should choose, give a bound of the form $O(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm).

(b) For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.)

(c) Although the algorithm you analyzed in parts (a) and (b) is the most natural way to solve the problem—after all, it just iterates through the relevant entries of the array B, filling in a value for each—it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $O(g(n))$, where $\lim_{n\to\infty} g(n)/f(n) = 0$.

7. There's a class of folk songs and holiday songs in which each verse consists of the previous verse, with one extra line added on. "The Twelve Days of Christmas" has this property; for example, when you get to the fifth verse, you sing about the five golden rings and then, reprising the lines from the fourth verse, also cover the four calling birds, the three French hens, the two turtle doves, and of course the partridge in the pear tree. The Aramaic song "Had gadya" from the Passover Haggadah works like this as well, as do many other songs.

These songs tend to last a long time, despite having relatively short scripts. In particular, you can convey the words plus instructions for one of these songs by specifying just the new line that is added in each verse, without having to write out all the previous lines each time. (So the phrase "five golden rings" only has to be written once, even though it will appear in verses five and onward.)

There's something asymptotic that can be analyzed here. Suppose, for concreteness, that each line has a length that is bounded by a constant c, and suppose that the song, when sung out loud, runs for n words total. Show how to encode such a song using a script that has length $f(n)$, for a function $f(n)$ that grows as slowly as possible.

8. You're doing some stress-testing on various models of glass jars to determine the height from which they can be dropped and still not break. The setup for this experiment, on a particular type of jar, is as follows. You have a ladder with n rungs, and you want to find the highest rung from which you can drop a copy of the jar and not have it break. We call this the *highest safe rung*.

It might be natural to try binary search: drop a jar from the middle rung, see if it breaks, and then recursively try from rung $n/4$ or $3n/4$ depending on the outcome. But this has the drawback that you could break a lot of jars in finding the answer.

If your primary goal were to conserve jars, on the other hand, you could try the following strategy. Start by dropping a jar from the first rung, then the second rung, and so forth, climbing one higher each time until the jar breaks. In this way, you only need a single jar—at the moment