total of at most $cn^2/2$, again plus the time in subsequent recursive calls. At the third level, we have four problems each of size $n/4$, each taking time at most $c(n/4)^2 = cn^2/16$, for a total of at most $cn^2/4$. Already we see that something is different from our solution to the analogous recurrence (5.1); whereas the total amount of work per level remained the same in that case, here it's decreasing.

- *Identifying a pattern:* At an arbitrary level j of the recursion, there are 2^j subproblems, each of size $n/2^j$, and hence the total work at this level is bounded by $2^j c(\frac{n}{2^j})^2 = cn^2/2^j$.
- *Summing over all levels of recursion:* Having gotten this far in the calculation, we've arrived at almost exactly the same sum that we had for the case $q = 1$ in the previous recurrence. We have

$$T(n) \leq \sum_{j=0}^{\log_2 n-1} \frac{cn^2}{2^j} = cn^2 \sum_{j=0}^{\log_2 n-1} \left(\frac{1}{2^j}\right) \leq 2cn^2 = O(n^2),$$

where the second inequality follows from the fact that we have a convergent geometric sum.

In retrospect, our initial guess of $T(n) = O(n^2 \log n)$, based on the analogy to (5.1), was an overestimate because of how quickly n^2 decreases as we replace it with $(\frac{n}{2})^2$, $(\frac{n}{4})^2$, $(\frac{n}{8})^2$, and so forth in the unrolling of the recurrence. This means that we get a geometric sum, rather than one that grows by a fixed amount over all n levels (as in the solution to (5.1)).

5.3 Counting Inversions

We've spent some time discussing approaches to solving a number of common recurrences. The remainder of the chapter will illustrate the application of divide-and-conquer to problems from a number of different domains; we will use what we've seen in the previous sections to bound the running times of these algorithms. We begin by showing how a variant of the Mergesort technique can be used to solve a problem that is not directly related to sorting numbers.

The Problem

We will consider a problem that arises in the analysis of *rankings*, which are becoming important to a number of current applications. For example, a number of sites on the Web make use of a technique known as *collaborative filtering*, in which they try to match your preferences (for books, movies, restaurants) with those of other people out on the Internet. Once the Web site has identified people with "similar" tastes to yours—based on a comparison

of how you and they rate various things—it can recommend new things that these other people have liked. Another application arises in *meta-search tools* on the Web, which execute the same query on many different search engines and then try to synthesize the results by looking for similarities and differences among the various rankings that the search engines return.

A core issue in applications like this is the problem of comparing two rankings. You rank a set of n movies, and then a collaborative filtering system consults its database to look for other people who had "similar" rankings. But what's a good way to measure, numerically, how similar two people's rankings are? Clearly an identical ranking is very similar, and a completely reversed ranking is very different; we want something that interpolates through the middle region.

Let's consider comparing your ranking and a stranger's ranking of the same set of n movies. A natural method would be to label the movies from 1 to n according to your ranking, then order these labels according to the stranger's ranking, and see how many pairs are "out of order." More concretely, we will consider the following problem. We are given a sequence of n numbers $a_1, \ldots, a_n$; we will assume that all the numbers are distinct. We want to define a measure that tells us how far this list is from being in ascending order; the value of the measure should be 0 if $a_1 < a_2 < \ldots < a_n$, and should increase as the numbers become more scrambled.

A natural way to quantify this notion is by counting the number of *inversions*. We say that two indices $i < j$ form an inversion if $a_i > a_j$, that is, if the two elements a_i and a_j are "out of order." We will seek to determine the number of inversions in the sequence $a_1, \ldots, a_n$.

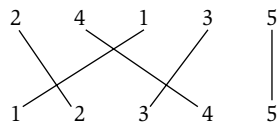

Figure 5.4 Counting the number of inversions in the sequence $2, 4, 1, 3, 5$. Each crossing pair of line segments corresponds to one pair that is in the opposite order in the input list and the ascending list—in other words, an inversion.

Just to pin down this definition, consider an example in which the sequence is $2, 4, 1, 3, 5$. There are three inversions in this sequence: $(2, 1)$, $(4, 1)$, and $(4, 3)$. There is also an appealing geometric way to visualize the inversions, pictured in Figure 5.4: we draw the sequence of input numbers in the order they're provided, and below that in ascending order. We then draw a line segment between each number in the top list and its copy in the lower list. Each crossing pair of line segments corresponds to one pair that is in the opposite order in the two lists—in other words, an inversion.

Note how the number of inversions is a measure that smoothly interpolates between complete agreement (when the sequence is in ascending order, then there are no inversions) and complete disagreement (if the sequence is in descending order, then every pair forms an inversion, and so there are $\binom{n}{2}$ of them).