the first heads.
c random vari-
d the memory-
nds to the num-
bability p. Let
ne identity from

$[X \mid Y = 1]$.

In this case, let
be Z. Then, by

a geometric ran-
n have the same

ectation is often
ometric random

ippose that each
ne of every type
ch box is chosen
that you do not

collaborate with others to collect coupons, how many boxes of cereal must you buy before you obtain at least one of every type of coupon? This simple problem arises in many different scenarios and will reappear in several places in the book.

Let X be the number of boxes bought until at least one of every type of coupon is obtained. We now determine $\mathbf{E}[X]$. If X_i is the number of boxes bought while you had exactly $i-1$ different coupons, then clearly $X = \sum_{i=1}^{n} X_i$.

The advantage of breaking the random variable X into a sum of n random variables X_i, $i = 1, \ldots, n$, is that each X_i is a geometric random variable. When exactly $i-1$ coupons have been found, the probability of obtaining a new coupon is

$$p_i = 1 - \frac{i-1}{n}.$$

Hence, X_i is a geometric random variable with parameter p_i, and

$$\mathbf{E}[X_i] = \frac{1}{p_i} = \frac{n}{n-i+1}.$$

Using the linearity of expectations, we have that

$$\begin{aligned}
\mathbf{E}[X] &= \mathbf{E}\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} \mathbf{E}[X_i] \\
&= \sum_{i=1}^{n} \frac{n}{n-i+1} \\
&= n \sum_{i=1}^{n} \frac{1}{i}.
\end{aligned}$$

The summation $\sum_{i=1}^{n} 1/i$ is known as the *harmonic number* $H(n)$, and as we show next, $H(n) = \ln n + \Theta(1)$. Thus, for the coupon collector's problem, the expected number of random coupons required to obtain all n coupons is $n \ln n + \Theta(n)$.

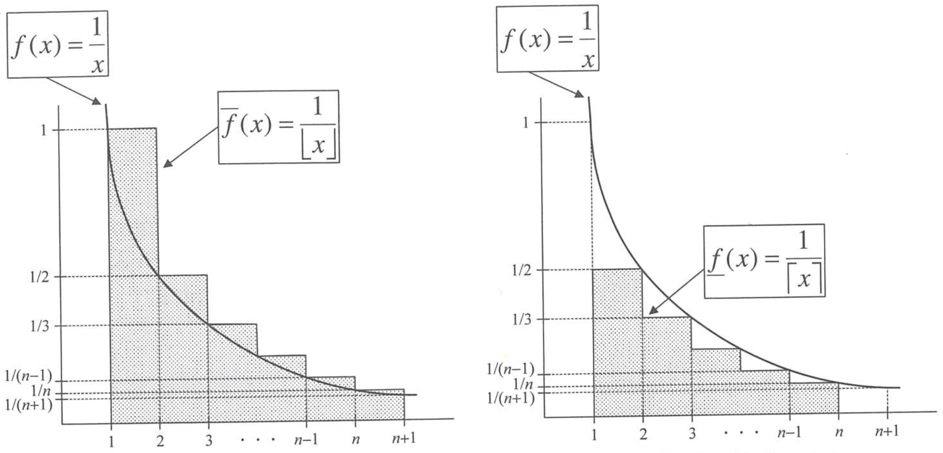
Lemma 2.10: *The harmonic number* $H(n) = \sum_{i=1}^{n} 1/i$ *satisfies* $H(n) = \ln n + \Theta(1)$.

Proof: Since $1/x$ is monotonically decreasing, we can write

$$\ln n = \int_{x=1}^{n} \frac{1}{x}\, dx \le \sum_{k=1}^{n} \frac{1}{k}$$

and

$$\sum_{k=2}^{n} \frac{1}{k} \le \int_{x=1}^{n} \frac{1}{x}\, dx = \ln n.$$



(a) Approximating $1/x$ from above.

(b) Approximating $1/x$ from below.

Figure 2.1: Approximating the area below $f(x) = 1/x$.

This is clarified in Figure 2.1, where the area below the curve $f(x) = 1/x$ corresponds to the integral and the areas of the shaded regions correspond to the summations $\sum_{k=1}^{n} 1/k$ and $\sum_{k=2}^{n} 1/k$.

Hence $\ln n \le H(n) \le \ln n + 1$, proving the claim. ■

As a simple application of the coupon collector's problem, suppose that packets are sent in a stream from a source host to a destination host along a fixed path of routers. The host at the destination would like to know which routers the stream of packets has passed through, in case it finds later that some router damaged packets that it processed. If there is enough room in the packet header, each router can append its identification number to the header, giving the path. Unfortunately, there may not be that much room available in the packet header.

Suppose instead that each packet header has space for exactly one router identification number, and this space is used to store the identification of a router chosen uniformly at random from all of the routers on the path. This can actually be accomplished easily; we consider how in Exercise 2.18. Then, from the point of view of the destination host, determining all the routers on the path is like a coupon collector's problem. If there are n routers along the path, then the expected number of packets in the stream that must arrive before the destination host knows all of the routers on the path is $nH(n) = n \ln n + \Theta(n)$.

2.5. Application: The Expected Run-Time of Quicksort

Quicksort is a simple – and, in practice, very efficient – sorting algorithm. The input is a list of n numbers $x_1, x_2, \ldots, x_n$. For convenience, we will assume that the numbers are distinct. A call to the Quicksort function begins by choosing a *pivot* element from the set. Let us assume the pivot is x. The algorithm proceeds by comparing every

```
Quicksort Algorithm:

Input: A list $S = \{x_1, \ldots, x_n\}$ of $n$ distinct elements over a totally ordered universe.
Output: The elements of $S$ in sorted order.

1. If $S$ has one or zero elements, return $S$. Otherwise continue.
2. Choose an element of $S$ as a pivot; call it $x$.
3. Compare every other element of $S$ to $x$ in order to divide the other elements into two sublists:
    (a) $S_1$ has all the elements of $S$ that are less than $x$;
    (b) $S_2$ has all those that are greater than $x$.
4. Use Quicksort to sort $S_1$ and $S_2$.
5. Return the list $S_1, x, S_2$.
```

Algorithm 2.1: Quicksort.

other element to x, dividing the list of elements into two sublists: those that are less than x and those that are greater than x. Notice that if the comparisons are performed in the natural order, from left to right, then the order of the elements in each sublist is the same as in the initial list. Quicksort then recursively sorts these sublists.

In the worst case, Quicksort requires $\Omega(n^2)$ comparison operations. For example, suppose our input has the form $x_1 = n$, $x_2 = n-1$, $\ldots$, $x_{n-1} = 2$, $x_n = 1$. Suppose also that we adopt the rule that the pivot should be the first element of the list. The first pivot chosen is then n, so Quicksort performs $n-1$ comparisons. The division has yielded one sublist of size 0 (which requires no additional work) and another of size $n-1$, with the order $n-1, n-2, \ldots, 2, 1$. The next pivot chosen is $n-1$, so Quicksort performs $n-2$ comparisons and is left with one group of size $n-2$ in the order $n-2, n-3, \ldots, 2, 1$. Continuing in this fashion, Quicksort performs

$$(n-1) + (n-2) + \cdots + 2 + 1 = \frac{n(n-1)}{2} \text{ comparisons.}$$

This is not the only bad case that leads to $\Omega(n^2)$ comparisons; similarly poor performance occurs if the pivot element is chosen from among the smallest few or the largest few elements each time.

We clearly made a bad choice of pivots for the given input. A reasonable choice of pivots would require many fewer comparisons. For example, if our pivot always split the list into two sublists of size at most $\lceil n/2 \rceil$, then the number of comparisons $C(n)$ would obey the following recurrence relation:

$$C(n) \le 2C(\lceil n/2 \rceil) + \Theta(n).$$

The solution to this equation yields $C(n) = O(n \log n)$, which is the best possible result for comparison-based sorting. In fact, any sequence of pivot elements that always split the input list into two sublists each of size at least cn for some constant c would yield an $O(n \log n)$ running time.

This discussion provides some intuition for how we would like pivots to be chosen. In each iteration of the algorithm there is a good set of pivot elements that split the input list into two almost equal sublists; it suffices if the sizes of the two sublists are within a constant factor of each other. There is a also a bad set of pivot elements that do not split up the list significantly. If good pivots are chosen sufficiently often, Quicksort will terminate quickly. How can we guarantee that the algorithm chooses good pivot elements sufficiently often? We can resolve this problem in one of two ways.

First, we can change the algorithm to choose the pivots randomly. This makes Quicksort a randomized algorithm; the randomization makes it extremely unlikely that we repeatedly choose the wrong pivots. We demonstrate shortly that the expected number of comparisons made by a simple randomized Quicksort is $2n \ln n + O(n)$, matching (up to constant factors) the $\Omega(n \log n)$ bound for comparison-based sorting. Here, the expectation is over the random choice of pivots.

A second possibility is that we can keep our deterministic algorithm, using the first list element as a pivot, but consider a probabilistic model of the inputs. A *permutation* of a set of n distinct items is just one of the $n!$ orderings of these items. Instead of looking for the worst possible input, we assume that the input items are given to us in a random order. This may be a reasonable assumption for some applications; alternatively, this could be accomplished by ordering the input list according to a randomly chosen permutation before running the deterministic Quicksort algorithm. In this case, we have a deterministic algorithm but a *probabilistic analysis* based on a model of the inputs. We again show in this setting that the expected number of comparisons made is $2n \ln n + O(n)$. Here, the expectation is over the random choice of inputs.

The same techniques are generally used both in analyses of randomized algorithms and in probabilistic analyses of deterministic algorithms. Indeed, in this application the analysis of the randomized Quicksort and the probabilistic analysis of the deterministic Quicksort under random inputs are essentially the same.

Let us first analyze Random Quicksort, the randomized algorithm version of Quicksort.

Theorem 2.11: *Suppose that, whenever a pivot is chosen for Random Quicksort, it is chosen independently and uniformly at random from all possibilities. Then, for any input, the expected number of comparisons made by Random Quicksort is* $2n \ln n + O(n)$.

Proof: Let $y_1, y_2, \ldots, y_n$ be the same values as the input values $x_1, x_2, \ldots, x_n$ but sorted in increasing order. For $i < j$, let X_{ij} be a random variable that takes on the value 1 if y_i and y_j are compared at any time over the course of the algorithm, and 0 otherwise. Then the total number of comparisons X satisfies

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij},$$

and

$$\begin{aligned}\mathbf{E}[X] &= \mathbf{E}\left[\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} X_{ij}\right]\\ &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \mathbf{E}[X_{ij}]\end{aligned}$$

by the linearity of expectations.

Since X_{ij} is an indicator random variable that takes on only the values 0 and 1, $\mathbf{E}[X_{ij}]$ is equal to the probability that X_{ij} is 1. Hence all we need to do is compute the probability that two elements y_i and y_j are compared. Now, y_i and y_j are compared if and only if either y_i or y_j is the first pivot selected by Random Quicksort from the set $Y^{ij} = \{y_i, y_{i+1}, \dots, y_{j-1}, y_j\}$. This is because if y_i (or y_j) is the first pivot selected from this set, then y_i and y_j must still be in the same sublist, and hence they will be compared. Similarly, if neither is the first pivot from this set, then y_i and y_j will be separated into distinct sublists and so will not be compared.

Since our pivots are chosen independently and uniformly at random from each sublist, it follows that, the first time a pivot is chosen from Y^{ij}, it is equally likely to be any element from this set. Thus the probability that y_i or y_j is the first pivot selected from Y^{ij}, which is the probability that $X_{ij} = 1$, is $2/(j - i + 1)$. Using the substitution $k = j - i + 1$ then yields

$$\begin{aligned}\mathbf{E}[X] &= \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \frac{2}{j-i+1}\\ &= \sum_{i=1}^{n-1}\sum_{k=2}^{n-i+1} \frac{2}{k}\\ &= \sum_{k=2}^{n}\sum_{i=1}^{n+1-k} \frac{2}{k}\\ &= \sum_{k=2}^{n}(n+1-k)\frac{2}{k}\\ &= \left((n+1)\sum_{k=2}^{n}\frac{2}{k}\right) - 2(n-1)\\ &= (2n+2)\sum_{k=1}^{n}\frac{1}{k} - 4n.\end{aligned}$$

Notice that we used a rearrangement of the double summation to obtain a clean form for the expectation.

Recalling that the summation $H(n) = \sum_{k=1}^{n} 1/k$ satisfies $H(n) = \ln n + \Theta(1)$, we have $\mathbf{E}[X] = 2n \ln n + \Theta(n)$. ■

Next we consider the deterministic version of Quicksort, on random input. We assume that the order of the elements in each recursively constructed sublist is the same as in the initial list.

Theorem 2.12: *Suppose that, whenever a pivot is chosen for Quicksort, the first element of the sublist is chosen. If the input is chosen uniformly at random from all possible permutations of the values, then the expected number of comparisons made by Deterministic Quicksort is $2n \ln n + O(n)$.*

Proof: The proof is essentially the same as for Random Quicksort. Again, y_i and y_j are compared if and only if either y_i or y_j is the first pivot selected by Quicksort from the set Y^{ij}. Since the order of elements in each sublist is the same as in the original list, the first pivot selected from the set Y^{ij} is just the first element from Y^{ij} in the input list, and since all possible permutations of the input values are equally likely, every element in Y^{ij} is equally likely to be first. From this, we can again use linearity of expectations in the same way as in the analysis of Random Quicksort to obtain the same expression for $\mathbf{E}[X]$. ■

2.6. Exercises

Exercise 2.1: Suppose we roll a fair k-sided die with the numbers 1 through k on the die's faces. If X is the number that appears, what is $\mathbf{E}[X]$?

Exercise 2.2: A monkey types on a 26-letter keyboard that has lowercase letters only. Each letter is chosen independently and uniformly at random from the alphabet. If the monkey types 1,000,000 letters, what is the expected number of times the sequence "proof" appears?

Exercise 2.3: Give examples of functions f and random variables X where $\mathbf{E}[f(X)] \leq f(\mathbf{E}[X])$, $\mathbf{E}[f(X)] = f(\mathbf{E}[X])$, and $\mathbf{E}[f(X)] \geq f(\mathbf{E}[X])$.

Exercise 2.4: Prove that $\mathbf{E}[X^k] \geq \mathbf{E}[X]^k$ for any even integer $k \geq 1$.

Exercise 2.5: If X is a $B(n, 1/2)$ random variable with $n \geq 1$, show that the probability that X is even is $1/2$.

Exercise 2.6: Suppose that we independently roll two standard six-sided dice. Let X_1 be the number that shows on the first die, X_2 the number on the second die, and X the sum of the numbers on the two dice.

(a) What is $\mathbf{E}[X \mid X_1 \text{ is even}]$?
(b) What is $\mathbf{E}[X \mid X_1 = X_2]$?
(c) What is $\mathbf{E}[X_1 \mid X = 9]$?
(d) What is $\mathbf{E}[X_1 - X_2 \mid X = k]$ for k in the range $[2, 12]$?