9.4 Generating a Random Permutation

Consider the problem of simulating a card game. The deck consists of 52 distinct cards. In the course of a deal, we must generate cards from the deck, without duplicates. In effect, we need to shuffle the cards and then iterate through the deck. We want the shuffle to be fair. That is, each of the 52! possible orderings of the deck should be equally likely as a result of the shuffle.

Random permutations can be generated in linear time using one random number per item.

This type of problem involves *random permutations*. A random permutation uses one distinct random number per item. In general, the problem is the following: Generate a random permutation of 1, 2, …, *N*. All permutations should be equally likely. The randomness of the random permutation is, of course, limited by the randomness of the pseudorandom-number generator. Thus all permutations being equally likely is contingent on all random numbers' being uniformly distributed and independent. Random permutations can be generated in linear time.

A routine, `permute`, to generate a random permutation is shown in Figure 9.7. Here is how the `permute` routine works. We assume the items in the array are present in some unshuffled order. The loop performs a random shuffling. In each iteration of the loop, we switch `a[j]` with some array element in positions 0 to `j` (it is possible to perform no swap). A random permutation of 1, 2, …, *N* is generated by passing an array containing these items to `permute`.

The correctness of `permute` is subtle.

It is clear that `permute` generates shuffled permutations. But are all permutations equally likely? The answer is both yes and no. The answer, based on the algorithm, is yes. There are *N*! possible permutations, and the number of different possible outcomes of the $N - 1$ calls to `randomInt` at line 11 is also *N*! This is because the first call produces either 0 or 1, so it has two outcomes. The second call produces either 0, 1, or 2, so it has three outcomes. The last call has *N* outcomes. The total number of outcomes is the product of all these possibilities because each random number is independent of the previous one. All we have to show is that each sequence of random numbers corresponds to one and only one permutation. This can be established by working backward (see Exercise 9.6).

```
1     /**
2      * Randomly rearrange an array.
3      * The random numbers used depend on the time and day.
4      * @param a the array.
5      */
6     public static final void permute( Object [ ] a )
7     {
8         Random r = new Random( );
9
10        for( int j = 1; j < a.length; j++ )
11            Sort.swapReferences( a, j, r.randomInt( 0, j ) );
12    }
```

Figure 9.7 Permutation routine