Line 15 saves a reference to the original table. We then create a new, double-sized, empty hash table at lines 18 to 19. Finally, we scan through the original array and `insert` any active elements into the new table. The `insert` will use the new hash function (since the array has a new size) and will automatically resolve all collisions. We can be sure that the recursive call to `insert` (at line 24) does not force another rehash. Alternatively, we could replace line 24 with two lines of code surrounded by braces (see Exercise 19.14).

Quadratic probing is implemented in `findPos`. It uses the previously described trick to avoid multiplications and mods.

So far nothing we have done depends on quadratic probing. Figure 19.14 (page 571) implements quadratic probing by extending `ProbingHashTable` and providing `findPos`, which finally deals with the quadratic probing algorithm. We keep searching the table until we find either an empty cell or a match. Lines 12 through 14 directly implement the methodology described in Theorem 19.5. Note that if `x` is marked deleted, its position will be returned by `findPos`. `assertFound` will eventually throw an exception when it determines that `x` is no longer active.

19.4.2 Analysis of Quadratic Probing

In *secondary clustering*, elements that hash to the same position will probe the same alternative cells. Secondary clustering is a minor theoretical blemish.

Quadratic probing has not yet been mathematically analyzed. Although it eliminates primary clustering, elements that hash to the same position will probe the same alternative cells. This is known as *secondary clustering*. Once again, the independence of successive probes cannot be assumed. Secondary clustering is a slight theoretical blemish. Simulation results suggest that it generally causes less than an extra one-half probe per search and that this is true only for high load factors. Figure 19.5 illustrates the difference between linear probing and quadratic probing and shows that quadratic probing does not suffer from as much clustering as does linear probing.

Double hashing is a hashing technique that does not suffer from secondary clustering. A second hash function is used to drive the collision resolution.

There are techniques that eliminate secondary clustering. The most popular of these is *double hashing*, in which a second hash function is used to drive the collision resolution. Specifically, we probe at a distance $Hash_2(X)$, $2Hash_2(X)$, and so on. The second hash function must be carefully chosen (for example, it should *never* evaluate to 0), and we need to make sure that all cells can be probed. A function such as $Hash_2(X) = R - (X \bmod R)$, with R a prime smaller than M, will generally work well. Double hashing is theoretically interesting because it can be shown to use essentially the same number of probes as the purely random analysis of linear probing would imply. However, it is somewhat more complicated than quadratic probing to implement and, as we see, requires careful attention to some details.

There seems to be no good reason not to use a quadratic probing strategy, unless the overhead of maintaining a half-empty table is burdensome. This would be the case in other programming languages if the items being stored were very large.