

Figure 3.40 illustrates the quick-sort algorithm's behavior as it sorts an array of words. This illustration assumes, for simplicity, that the leftmost word is chosen as the pivot.

Now let us study the *partitioning* problem. It is very important to understand that we *need not sort* the array while partitioning it. (That would make steps 1.2 and 1.3 of the quick-sort algorithm redundant!) Nor do we need to pre-select the pivot's final position  $p$ . The deliberately loose specification of the partitioning problem turns out to be important, because it allows us to develop an efficient partitioning algorithm whose time complexity is  $O(n)$ .

Numerous partitioning algorithms have been developed. One idea is as follows. First choose some value as the pivot. Then scan the array from the left looking for a value greater than the pivot, and 'simultaneously' scan the array from the right looking for a value less than the pivot; if we find such values, swap them. Repeat the simultaneous scans until they meet. Finally, put the pivot into the position where the scans met.

We shall adopt a simpler idea, which is captured in Algorithm 3.41. Step 1 chooses the value of the leftmost component as the pivot. Step 2 scans from left to right. Whenever step 2.1 finds a value less than the pivot, step 2.1.1 'rotates' three values: the pivot itself, its right neighbor, and the value just found to be less than the pivot. (This rotation is just a three-way swap.)

Figure 3.42 shows the partitioning algorithm's loop invariant:  $a[p]$  contains the pivot,  $a[\text{left} \dots p - 1]$  contain values known to be less than  $a[p]$ , and  $a[p + 1 \dots r - 1]$  contain

	0	1	2	3	4	5	6	7	8
Initially:	fox	cow	pig	cat	rat	lion	tiger	goat	dog
After step 1.1:	cow	cat	dog	fox	rat	lion	tiger	goat	pig
After step 1.2:	cat	cow	dog	fox	rat	lion	tiger	goat	pig
After step 1.3:	cat	cow	dog	fox	goat	lion	pig	rat	tiger

**Figure 3.40** Illustration of the array quick-sort algorithm (assuming that the leftmost component is chosen as the pivot).

To partition  $a[\text{left} \dots \text{right}]$  such that  $a[\text{left} \dots p - 1]$  are all less than or equal to  $a[p]$ , and  $a[p + 1 \dots \text{right}]$  are all greater than or equal to  $a[p]$ :

1. Let *pivot* be the value of  $a[\text{left}]$ , and set  $p = \text{left}$ .
2. For  $r = \text{left} + 1, \dots, \text{right}$ , repeat:
  - 2.1. If  $a[r]$  is less than *pivot*:
    - 2.1.1. Copy  $a[r]$  into  $a[p]$ ,  $a[p + 1]$  into  $a[r]$ , and *pivot* into  $a[p + 1]$ .
    - 2.1.2. Increment  $p$ .
3. Terminate with answer  $p$ .

**Algorithm 3.41** Quick-sort partitioning algorithm. (See the text for variations on step 1.)