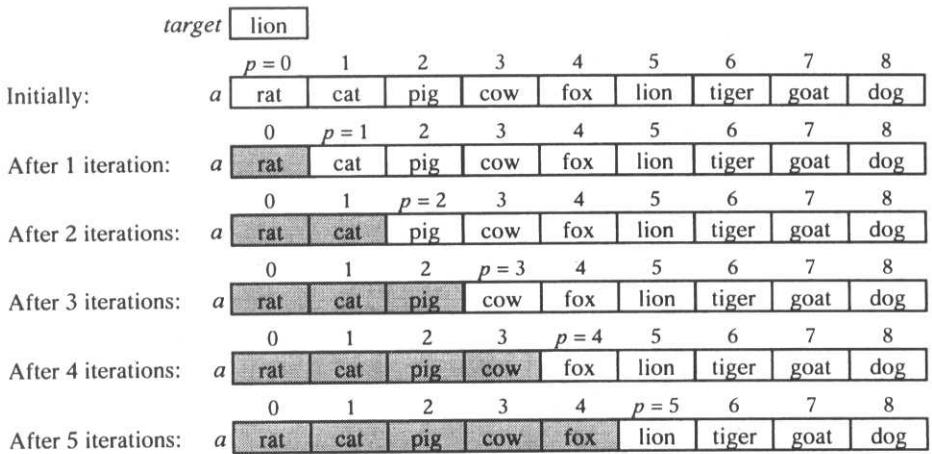


A *loop invariant* is a state that pertains before and after every iteration of a particular loop in an algorithm or program. Figure 3.13 is an example of a loop invariant. We shall use loop invariants frequently in this book to assist in understanding algorithms (some of which are rather more complicated than Algorithm 3.12).

Figure 3.14 illustrates linear search of an array of words. At each step, the components known not to equal *target* are highlighted. (Compare with Figure 3.13.)

Let us now analyze the linear search algorithm's time efficiency. For a searching algorithm the characteristic operations are comparisons. Let  $n = \text{right} - \text{left} + 1$  be the length of  $a[\text{left} \dots \text{right}]$ . There are two cases to consider. If the search is *successful*, step 1 could compare *target* with any number of components from 1 through  $n$ , so:

$$\text{Average no. of comparisons (successful search)} = (n + 1)/2 \tag{3.3}$$



**Figure 3.14** Illustration of the unsorted array linear search algorithm.

```

static final int NONE = -1; // ... distinct from any array index.
static int linearSearch (Object target,
                        Object[] a, int left, int right) {
// Find which (if any) component of a[left...right] equals target.
    for (int p = left; p <= right; p++) {
        if (target.equals(a[p])) return p;
    }
    return NONE;
}

```

**Program 3.15** Implementation of the unsorted array linear search algorithm as a Java method.