

### 3.6.4 Quick-sort

**Quick-sort** is another sorting algorithm based on the divide-and-conquer strategy. The idea is as follows. Choose any value from the array, and call that value the **pivot**. Then **partition** the array into three subarrays, such that the left subarray contains values known to be less than (or equal to) the pivot, the middle subarray is a single component containing the pivot itself, and the right subarray contains values known to be greater than (or equal to) the pivot. Now all that remains to be done is to sort the left and right subarrays.

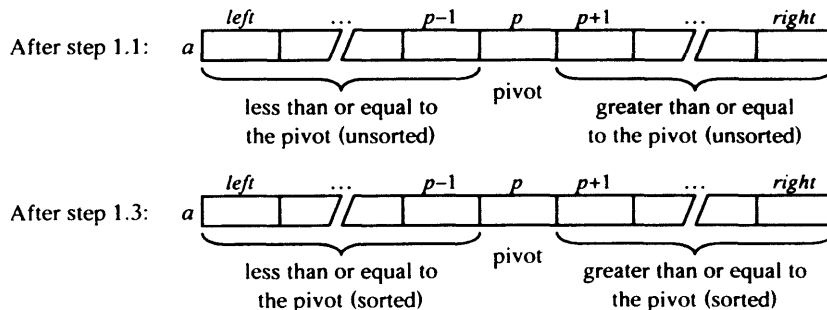
This idea is captured by Algorithm 3.38. Step 1.1 is the partitioning step; this is delegated to an auxiliary partitioning algorithm, which we shall discuss shortly. Steps 1.2 and 1.3 sort the left and right subarrays; each is just a recursive call to the quick-sort algorithm itself. The easy (non-recursive) case is when the array has fewer than two components.

Figure 3.39 shows the state of the array immediately after the partitioning step, and after the two sorting steps. We can see that the partitioning step has moved the pivot into its final position,  $a[p]$ . It has also moved all values less than the pivot into the left subarray,  $a[\text{left} \dots p - 1]$ , and all values greater than the pivot into the right subarray,  $a[p + 1 \dots \text{right}]$ . (Any values that happen to be equal to the pivot could end up in either subarray – we do not care which.) Now the left and right subarrays can be sorted separately. Figure 3.39 shows that this makes the whole array sorted.

To sort  $a[\text{left} \dots \text{right}]$ :

1. If  $\text{left} < \text{right}$ :
  - 1.1. 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  $a[p]$ .
  - 1.2. Sort  $a[\text{left} \dots p-1]$ .
  - 1.3. Sort  $a[p+1 \dots \text{right}]$ .
2. Terminate.

**Algorithm 3.38** Array quick-sort algorithm.



**Figure 3.39** Invariants for the array quick-sort algorithm.