

$$\text{Max. no. of comparisons} = n - 1 \quad (3.7)$$

In terms of either copies or comparisons, the merging algorithm has time complexity  $O(n_1 + n_2)$  or  $O(n)$ .

Program 3.26 shows a Java method implementing the merging algorithm. This method takes all three arrays and their bounds as parameters. It is coded tightly, using the post-increment operator ‘++’ to increment the counters  $i$ ,  $j$ , and  $k$  ‘on the fly’.

## 3.6 Sorting

If we have to maintain a large collection of data such as a dictionary or telephone directory, it makes sense to maintain the data in sorted order. We can efficiently search a sorted collection, and we can efficiently merge two sorted collections. But what if the initial collection is unsorted? Or what if we are given an unsorted collection to add to the main collection?

Situations like these give rise to the *sorting* problem: given a collection of data, rearrange the data into ascending order. Sorting is a classical problem in computer science. Numerous algorithms have been devised, varying widely in subtlety and efficiency.

In this section we shall study several array sorting algorithms. In each case, the problem is to sort the (sub)array  $a[\text{left} \dots \text{right}]$ .

### 3.6.1 Selection sort

Consider the following idea. First, find the least value in the array, and swap it into the leftmost component where it belongs. Henceforth ignore the leftmost component, and repeat the process on the unsorted components to its right. When only one component remains to be sorted, we are finished.

This simple idea is the basis of *selection sort*, and is captured by Algorithm 3.27. Step 1.1 employs an auxiliary algorithm to find the least of a sequence of components. (The auxiliary algorithm is omitted here, but see Exercise 3.10(d).)

Figure 3.28 shows the loop invariant for the selection sort algorithm. The sorted subarray,  $a[\text{left} \dots l - 1]$ , contains the lesser values. The unsorted subarray,  $a[l \dots \text{right}]$ , contains the greater values. The algorithm finds the least value in the unsorted subarray, and swaps it into  $a[l]$ . Now we can claim that the subarray  $a[\text{left} \dots l]$  is sorted, so the sorted subarray has expanded by one component at the expense of the unsorted subarray.

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

1. For  $l = \text{left}, \dots, \text{right} - 1$ , repeat:
  - 1.1. Set  $p$  such that  $a[p]$  is the least of  $a[l \dots \text{right}]$ .
  - 1.2. If  $p \neq l$ , swap  $a[p]$  and  $a[l]$ .
2. Terminate.

**Algorithm 3.27** Array selection sort algorithm.