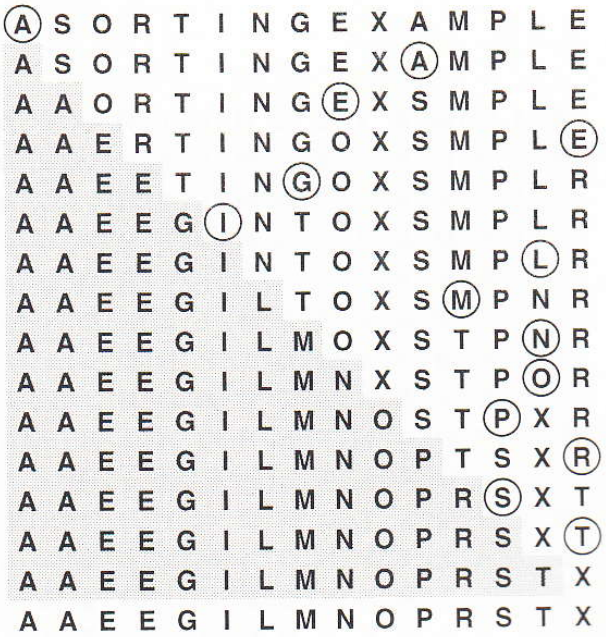


Figure 6.3
Selection sort example

The first pass has no effect in this example, because there is no element in the array smaller than the A *at the left. On the second pass, the other* A *is the smallest remaining element, so it is exchanged with the* S *in the second position. Then, the* E *near the middle is exchanged with the* O *in the third position on the third pass; then, the other* E *is exchanged with the* R *in the fourth position on the fourth pass; and so forth.*

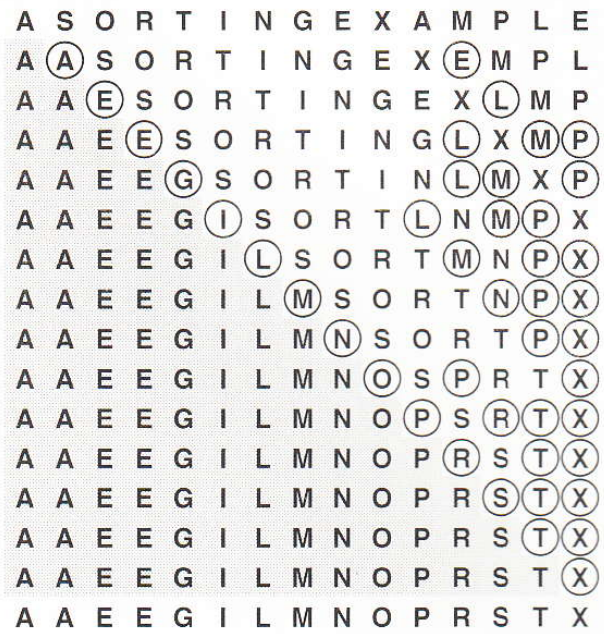


Figure 6.5
Bubble sort example

Small keys percolate over to the left in bubble sort. As the sort moves from right to left, each key is exchanged with the one on its left until a smaller one is encountered. On the first pass, the E *is exchanged with the* L*, the* P*, and the* M *before stopping at the* A *on the right; then the* A *moves to the beginning of the file, stopping at the other* A*, which is already in position. The* i*th smallest key reaches its final position after the* i*th pass, just as in selection sort, but other keys are moved closer to their final position as well.*

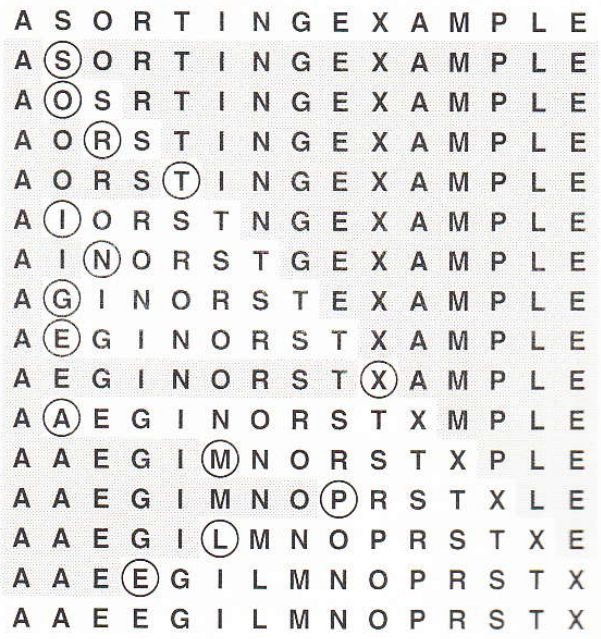


Figure 6.4
Insertion sort example

During the first pass of insertion sort, the S *in the second position is larger than the* A*, so it does not have to be moved. On the second pass, when the* O *in the third position is encountered, it is exchanged with the* S *to put* A O S *in sorted order, and so forth. Unshaded elements that are not circled are those that were moved one position to the right.*

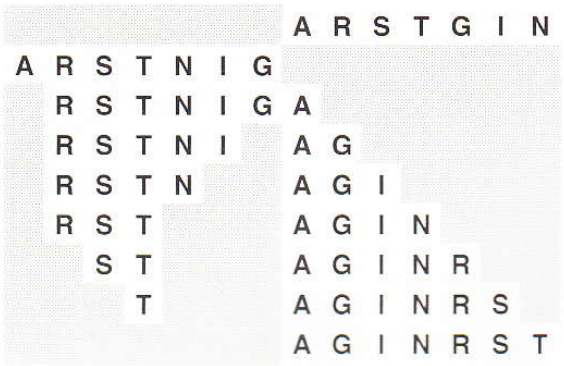


Figure 8.1
Merging without sentinels

To merge two ascending files, we copy into an auxiliary array, with the second file in reverse order immediately following the first. Then, we follow thIs simple rule: Move the left or right item, whichever has the smaller key, to the output. The largest key serves as a sentinel for the other file, no matter in which file the key is. This figure illustrates how the files A R S T *and* G I N *are merged.*

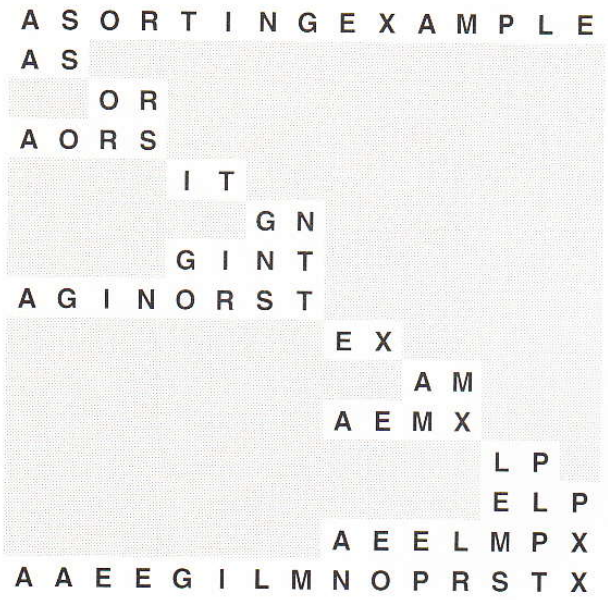
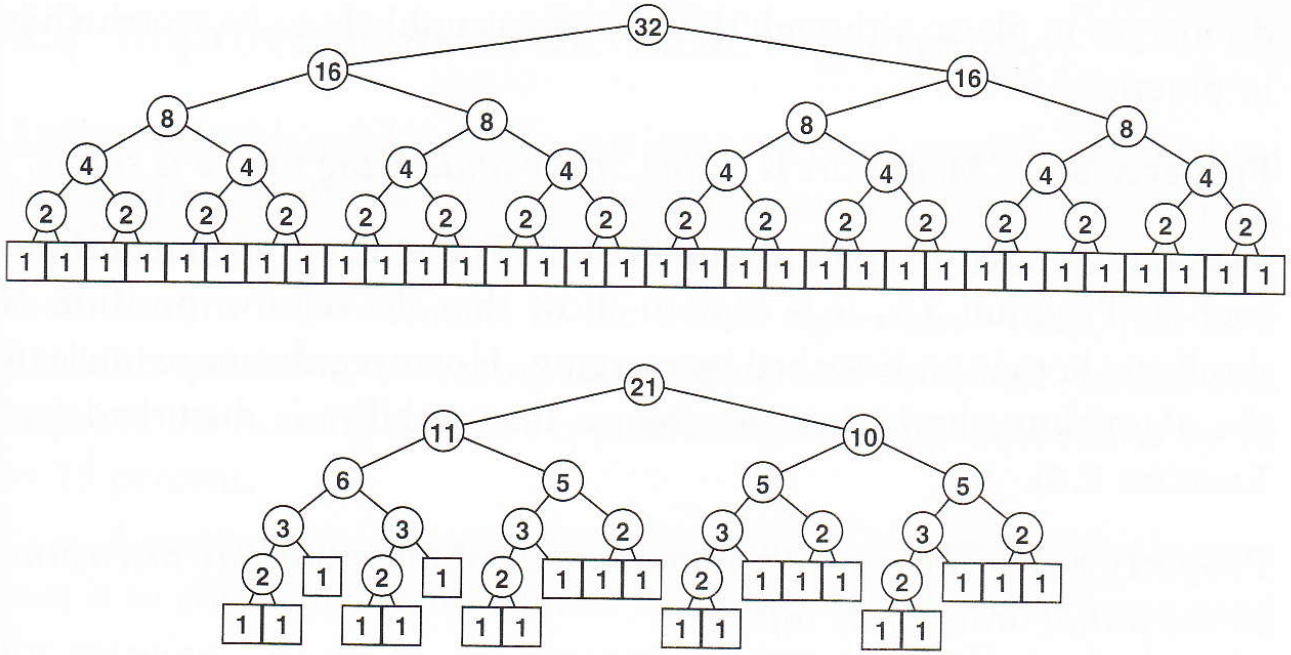


Figure 8.2
Top-down mergesort example

Each line shows the result of a call on `merge` *during top-down mergesort. First, we merge A and S to get A S; then, we merge O and R to get O R; then, we merge O R with A S to get A O R S. Later, we merge I T with G N to get G I N T, then merge this result with A O R S to get A G I N O R S T, and so on. The method recursively builds up small sorted files into larger ones.*



comparisons used by the algorithm is precisely the sum of all the node labels.

Figure 8.3
Divide-and-conquer trees

These tree diagrams depict the sizes of the subproblems created by top-down mergesort. Unlike the trees corresponding to quicksort, for example, these patterns are dependent on only the initial file size, rather than on the values of the keys in the file. The top diagram shows how a file of 32 elements is sorted. We (recursively) sort two files of 16 elements, then merge them. We sort the files of 16 elements by (recursively) sorting files of 8 elements, and so forth. For file sizes that are not a power of 2, the pattern is more intricate, as indicated by the bottom diagram.