8. A bubble sort can sort an array of n entries into ascending order by making n – 1 passes through the array. On each pass, it compares adjacent entries and swaps them if they are out of order. For example, on the first pass, it compares the first and second entries, then the second and third entries, and so on. At the end of the first pass, the largest entry is in its proper position at the end of the array. We say that it has bubbled to its correct spot. Each subsequent pass ignores the entries at the end of the array, since they are sorted and are larger than any of the remaining entries. Thus, each pass makes one fewer comparison than the previous pass. Figure 8-17 gives an example of a bubble sort.

Implement the bubble sort a. Iteratively

a. Iteratively

b. Recursively

10. The bubble sort in Exercise 8 always makes n passes. However, it is possible for the array to become sorted before

all n passes are complete. For example, a bubble sort of the array 9216478

is sorted after only two passes:

2 1 6 4 7 8 9(endofpass1) 1 2 4 6 7 8 9(endofpass2)

But since a swap occurred during the second pass, the sort needs to make one more pass to check that the array is in order. Additional passes, such as the ones that the algorithm in Exercise 8 would make, are unnecessary.

You can skip these unnecessary passes and even do less work by remembering where the last swap occurred. During the first pass, the last swap is of the 9 and 8. The second pass checks up to the 8. But during the second pass, the last swap is of the 6 and 4. You now know that 6, 7, 8, and 9 are sorted. The third pass needs only to check up to the 4, instead of the 7, as an ordinary bubble sort would do. No swaps occur during the third pass, so the index of the last swap during this pass is taken as zero, indicating that no further passes are necessary. Implement this revised bubble sort.