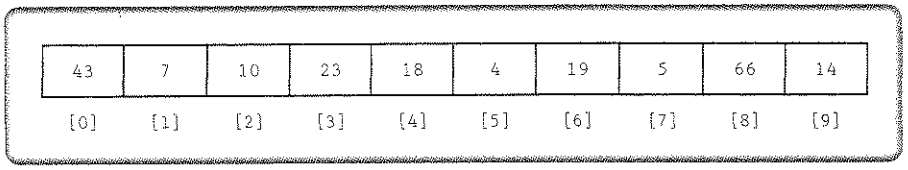
imation—that helps us determine how much work is required by a particular algorithm. Both building and measuring tools are needed to construct sound program solutions.

Exercises

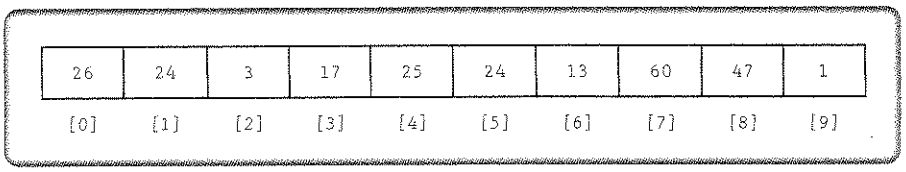
1. Show the contents of the array



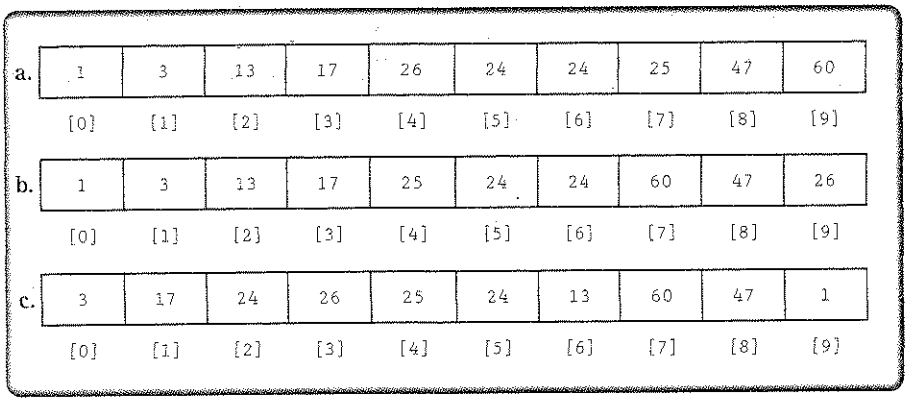
after the fourth iteration of

   a. `BubbleSort`
   b. `SelectionSort`
   c. `InsertionSort`

2. a. Show how the values in the array in Exercise 1 would have to be rearranged to satisfy the heap property.
   b. Show how the array would look with four values in the sorted portion after reheaping.
3. a. Show how the values in the array in Exercise 1 would be arranged immediately before the execution of the function `Merge` in the original (nonrecursive) call to `MergeSort`.
   b. Show how the values in the array in Exercise 1 would be arranged immediately before the first recursive call to `QuickSort`.
4. Given the array



tell which sorting algorithm would produce the following results after four iterations:



5. How many comparisons would be needed to sort an array containing 100 elements using `ShortBubble`
   a. in the worst case?
   b. in the best case?
6. A sorting function is called to sort a list of 100 integers that have been read from a file. If all 100 values are zero, what would the execution requirements (in terms of Big-O notation) be if the sort used was
   a. `QuickSort`, with the first element used as the split value?
   b. `ShortBubble`?
   c. `SelectionSort`?
   d. `HeapSort`?
   e. `InsertionSort`?
   f. `MergeSort`?
7. How many comparisons would be needed to sort an array containing 100 elements using `SelectionSort` if the original array values were already sorted?
   a. 10,000
   b. 9,900
   c. 4,950
   d. 99
   e. None of the above
8. A merge sort is used to sort an array of 1,000 test scores in descending order. Which of the following statements is true?
   a. The sort is fastest if the original test scores are sorted from smallest to largest.
   b. The sort is fastest if the original test scores are in completely random order.
   c. The sort is fastest if the original test scores are sorted from largest to smallest.
   d. The sort is the same, no matter what the order of the original elements.

9. A list is sorted from smallest to largest when a sort algorithm is called. Which of the following sorts would take the longest time to execute, and which would take the shortest time?
   a. `QuickSort`, with the first element used as the split value
   b. `ShortBubble`
   c. `SelectionSort`
   d. `HeapSort`
   e. `InsertionSort`
   f. `MergeSort`
10. a. In what case(s), if any, is the bubble sort $O(N)$?
    b. In what case(s), if any, is the selection sort $O(\log_2 N)$?
    c. In what case(s), if any, is quick sort $O(N^2)$?
11. A very large array of elements is to be sorted. The program will be run on a personal computer with limited memory. Which sort would be a better choice: a heap sort or a merge sort? Why?
12. Use the Three-Question Method to verify `MergeSort`.
13. True or false? Correct the false statements.
    a. `MergeSort` requires more space to execute than `HeapSort`.
    b. `QuickSort` (using the first element as the split value) is better for nearly sorted data than `HeapSort`.
    c. The efficiency of `HeapSort` is not affected by the order of the elements on entrance to the function.
14. Which of the following is true about `QuickSort`?
    a. A recursive version executes faster than a nonrecursive version.
    b. A recursive version has fewer lines of code than a nonrecursive version.
    c. A nonrecursive version takes more space on the run-time stack than a recursive version.
    d. It can be programmed only as a recursive function.
15. What is meant by the statement that "programmer time is an efficiency consideration"? Give an example of a situation in which programmer time is used to justify the choice of an algorithm, possibly at the expense of other efficiency considerations.
16. Identify one or more correct answers: Reordering an array of pointers to list elements, rather than sorting the elements themselves, is a good idea when
    a. the number of elements is very large.
    b. the individual elements are large in size.
    c. the sort is recursive.
    d. there are multiple keys on which to sort the elements.

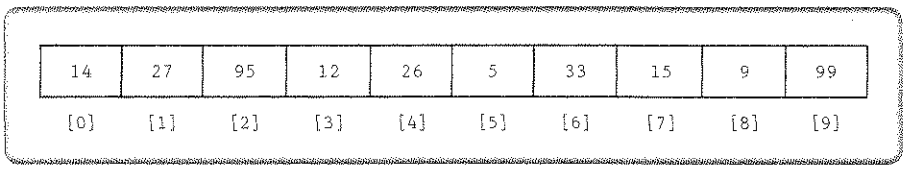
17. Go through the sorting algorithms coded in this chapter and determine which ones are stable as coded. If there are unstable algorithms (other than `HeapSort`), make them stable.
18. Give arguments for and against using functions (such as `Swap`) to encapsulate frequently used code in a sorting routine.
19. Write a version of the bubble sort algorithm that sorts a list of integers in descending order.
20. We said that `HeapSort` is inherently unstable. Explain why.
21. Sooey County is about to have its annual Big Pig Contest. Because the sheriff's son, Wilbur, is majoring in computer science, the county hires him to computerize the Big Pig judging. Each pig's name (string) and weight (integer) are to be read in from the keyboard. The county expects 500 entries this year.

    The output needed is a listing of the ten heaviest pigs, sorted from biggest to smallest. Because Wilbur has just learned some sorting methods in school, he feels up to the task of writing this "pork-gram." He writes a program to read in all the entries into an array of records, then uses a selection sort to put the entire array in order based on the `pigWeight` member. He then prints the ten largest values from the array.

    Can you think of a more efficient way to write this program? If so, write the algorithm.
22. State University needs a listing of the overall SAT percentiles of the 14,226 students it has accepted in the past year. The data are in a text file, with one line per student. That line contains the student's ID number, SAT overall percentile, math score, English score, and high school grade point average. (At least one blank separates each two fields.) The output needed is a listing of all the percentile scores, one per line, sorted from highest to lowest. Duplicates should be printed. Outline an $O(N)$ algorithm to produce the listing.
23. Which sorting algorithm would you *not* use under the following conditions?
    a. The sort must be stable.
    b. Data are in descending order by key.
    c. Data are in ascending order by key.
    d. Space is very limited.
24. Determine the Big-O measure for `SelectionSort` based on the number of elements moved rather than the number of comparisons,
    a. for the best case.
    b. for the worst case.
25. Determine the Big-O measure for `BubbleSort` based on the number of elements moved rather than the number of comparisons,
    a. for the best case.
    b. for the worst case.

26. Determine the Big-O measure for `QuickSort` based on the number of elements moved rather than the number of comparisons,
    a. for the best case.
    b. for the worst case.
27. Determine the Big-O measure for `MergeSort` based on the number of elements moved rather than the number of comparisons,
    a. for the best case.
    b. for the worst case.
28. Fill in the following table, showing the number of comparisons needed either to find the value or to determine that the value is not in the array, given the following array of values.

`dataValues`



14	27	95	12	26	5	33	15	9	99
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Values	Search dataValues Sequentially	Search sortedValues Sequentially	Binary Search sortedValues	Search Tree
15				
17				
14				
5				
99				
100				
0				

For Exercises 29–32, use the following values:

66 47 87 90 126 140 145 153 177 285 393 395 467 566 620 735

29. Store the values into a hash table with 20 positions, using the division method of hashing and the linear probing method of resolving collisions.
30. Store the values into a hash table with 20 positions, using rehashing as the method of collision resolution. Use `key % tableSize` as the hash function, and `(key + 3) % tableSize` as the rehash function.
31. Store the values into a hash table with ten buckets, each containing three slots. If a bucket is full, use the next (sequential) bucket that contains a free slot.
32. Store the values into a hash table that uses the hash function `key % 10` to determine into which of ten chains to put the value.
33. Fill in the following table, showing the number of comparisons needed to find each value using the hashing representations given in Exercises 29–32.

Number of Comparisons

Value	Exercise 29	Exercise 30	Exercise 31	Exercise 32
66				
467				
566				
735				
285				
87				

34. If you know the index of an element stored in an array of N unsorted elements, which of the following best describes the order of the algorithm to retrieve the element?
    a. $O(1)$
    b. $O(N)$
    c. $O(\log_2 N)$
    d. $O(N^2)$
    e. $O(0.5N)$
35. The element being searched for is *not* in an array of 100 elements. What is the *average* number of comparisons needed in a sequential search to determine that the element is not present
    a. if the elements are completely unsorted?
    b. if the elements are sorted from smallest to largest?
    c. if the elements are sorted from largest to smallest?

36. The element being searched for is *not* in an array of 100 elements. What is the *maximum* number of comparisons needed in a sequential search to determine that the element is not present
    a. if the elements are completely unsorted?
    b. if the elements are sorted from smallest to largest?
    c. if the elements are sorted from largest to smallest?
37. The element being searched for *is* in an array of 100 elements. What is the *average* number of comparisons needed in a sequential search to determine the position of the element
    a. if the elements are completely unsorted?
    b. if the elements are sorted from smallest to largest?
    c. if the elements are sorted from largest to smallest?
38. Choose the answer that correctly completes the following sentence: The elements in an array may be sorted by highest probability of being requested so as to reduce
    a. the average number of comparisons needed to find an element in the list.
    b. the maximum number of comparisons needed to detect that an element is not in the list.
    c. the average number of comparisons needed to detect that an element is not in the list.
    d. the maximum number of comparisons needed to find an element that is in the list.
39. True or false? Correct any false statements.
    a. A binary search of a sorted set of elements in an array is always faster than a sequential search of the elements.
    b. A binary search is an $O(N\log_2 N)$ algorithm.
    c. A binary search of elements in an array requires that the elements be sorted from smallest to largest.
    d. A high-probability ordering scheme would be a poor choice for arranging an array of elements that are equally likely to be requested.
    e. When a hash function is used to determine the placement of elements in an array, the order in which the elements are added does not affect the resulting array.
    f. When hashing is used, increasing the size of the array always reduces the number of collisions.
    g. If we use buckets in a hashing scheme, we do not have to worry about collision resolution.
    h. If we use chaining in a hashing scheme, we do not have to worry about collision resolution.

i. The functions in this chapter are used only for external searching (i.e., not for disk searching).

j. The goal of a successful hashing scheme is an O(1) search.

40. Choose the answer that correctly completes the following sentence: The number of comparisons required to find an element in a hash table with N buckets, of which M are full,

    a. is always 1.

    b. is usually only slightly less than N.

    c. may be large if M is only slightly less than N.

    d. is approximately $\log_2 M$.

    e. is approximately $\log_2 N$.

41. How might you order the elements in a list of C++'s reserved words to use the idea of high-probability ordering?

42. How would you modify the radix sort algorithm to sort the list in descending order?

43. The radix sort algorithm uses an array of queues. Would an array of stacks work just as well?

44. On the Web, the file `Sorts.in` contains a minimal test plan for the sorting algorithms we have studied. Design a more comprehensive test plan and apply it using `SortDr.cpp`.