offer little help in overcoming the complexity of algorithms of exponential or factorial time complexity. Because of the increased speed of computation, increases in computer memory, and the use of algorithms that take advantage of parallel processing, many problems that were considered impossible to solve five years ago are now routinely solved, and certainly five years from now this statement will still be true. This is even true when the algorithms used are intractable.

Exercises

1. Give a big-O estimate for the number of operations (where an operation is an addition or a multiplication) used in this segment of an algorithm.

```
t := 0
for i := 1 to 3
  for j := 1 to 4
    t := t + ij
```

2. Give a big-O estimate for the number additions used in this segment of an algorithm.

```
t := 0
for i := 1 to n
  for j := 1 to n
    t := t + i + j
```

3. Give a big-O estimate for the number of operations, where an operation is a comparison or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the **for** loops, where $a_1, a_2, ..., a_n$ are positive real numbers).

```
m := 0
for i := 1 to n
  for j := i + 1 to n
    m := max(a_i a_j, m)
```

4. Give a big-O estimate for the number of operations, where an operation is an addition or a multiplication, used in this segment of an algorithm (ignoring comparisons used to test the conditions in the **while** loop).

```
i := 1
t := 0
while i ≤ n
    t := t + i
    i := 2i
```

5. How many comparisons are used by the algorithm given in Exercise 16 of Section 3.1 to find the smallest natural number in a sequence of n natural numbers?

6. a) Use pseudocode to describe the algorithm that puts the first four terms of a list of real numbers of arbitrary length in increasing order using the insertion sort.

b) Show that this algorithm has time complexity $O(1)$ in terms of the number of comparisons used.

7. Suppose that an element is known to be among the first four elements in a list of 32 elements. Would a linear search or a binary search locate this element more rapidly?

8. Given a real number x and a positive integer k, determine the number of multiplications used to find x^{2^k} starting with x and successively squaring (to find x^2, x^4, and so on). Is this a more efficient way to find x^{2^k} than by multiplying x by itself the appropriate number of times?

9. Give a big-O estimate for the number of comparisons used by the algorithm that determines the number of 1s in a bit string by examining each bit of the string to determine whether it is a 1 bit (see Exercise 25 of Section 3.1).

***10. a)** Show that this algorithm determines the number of 1 bits in the bit string S:

```
procedure bit count(S: bit string)
count := 0
while S ≠ 0
  count := count + 1
  S := S ∧ (S − 1)
return count {count is the number of 1s in S}
```

Here $S - 1$ is the bit string obtained by changing the rightmost 1 bit of S to a 0 and all the 0 bits to the right of this to 1s. [Recall that $S \wedge (S - 1)$ is the bitwise *AND* of S and $S - 1$.]

b) How many bitwise *AND* operations are needed to find the number of 1 bits in a string S using the algorithm in part (a)?

11. a) Suppose we have n subsets $S_1, S_2, \ldots, S_n$ of the set $\{1, 2, \ldots, n\}$. Express a brute-force algorithm that determines whether there is a disjoint pair of these subsets. [*Hint:* The algorithm should loop through the subsets; for each subset S_i, it should then loop through all other subsets; and for each of these other subsets S_j, it should loop through all elements k in S_i to determine whether k also belongs to S_j.]

b) Give a big-O estimate for the number of times the algorithm needs to determine whether an integer is in one of the subsets.

12. Consider the following algorithm, which takes as input a sequence of n integers $a_1, a_2, \ldots, a_n$ and produces as output a matrix $\mathbf{M} = \{m_{ij}\}$ where m_{ij} is the minimum term in the sequence of integers $a_i, a_{i+1}, \ldots, a_j$ for $j \geq i$ and $m_{ij} = 0$ otherwise.

```
initialize M so that m_ij = a_i if j ≥ i and m_ij = 0
  otherwise
for i := 1 to n
  for j := i + 1 to n
    for k := i + 1 to j
      m_ij := min(m_ij, a_k)
return M= {m_ij} {m_ij is the minimum term of
  a_i, a_{i+1}, ..., a_j}
```

a) Show that this algorithm uses $O(n^3)$ comparisons to compute the matrix $\mathbf{M}$.

b) Show that this algorithm uses $\Omega(n^3)$ comparisons to compute the matrix $\mathbf{M}$. Using this fact and part (a), conclude that the algorithms uses $\Theta(n^3)$ comparisons. [*Hint:* Only consider the cases where $i \leq n/4$ and $j \geq 3n/4$ in the two outer loops in the algorithm.]

13. The conventional algorithm for evaluating a polynomial $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ at $x = c$ can be expressed in pseudocode by

procedure *polynomial*$(c, a_0, a_1, \ldots, a_n$: real numbers)
  power := 1
  $y := a_0$
  for $i := 1$ **to** n
    power := *power* $*$ c
    $y := y + a_i * power$
  return y $\{y = a_n c^n + a_{n-1}c^{n-1} + \cdots + a_1 c + a_0\}$

where the final value of y is the value of the polynomial at $x = c$.

a) Evaluate $3x^2 + x + 1$ at $x = 2$ by working through each step of the algorithm showing the values assigned at each assignment step.

b) Exactly how many multiplications and additions are used to evaluate a polynomial of degree n at $x = c$? (Do not count additions used to increment the loop variable.)

14. There is a more efficient algorithm (in terms of the number of multiplications and additions used) for evaluating polynomials than the conventional algorithm described in the previous exercise. It is called **Horner's method**. This pseudocode shows how to use this method to find the value of $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$ at $x = c$.

procedure *Horner*$(c, a_0, a_1, a_2, \ldots, a_n$: real numbers)
  $y := a_n$
  for $i := 1$ **to** n
    $y := y * c + a_{n-i}$
  return y $\{y = a_n c^n + a_{n-1}c^{n-1} + \cdots + a_1 c + a_0\}$

a) Evaluate $3x^2 + x + 1$ at $x = 2$ by working through each step of the algorithm showing the values assigned at each assignment step.

b) Exactly how many multiplications and additions are used by this algorithm to evaluate a polynomial of degree n at $x = c$? (Do not count additions used to increment the loop variable.)

15. What is the largest n for which one can solve within one second a problem using an algorithm that requires $f(n)$ bit operations, where each bit operation is carried out in 10^{-9} seconds, with these functions $f(n)$?

a) $\log n$ **b)** n **c)** $n \log n$
d) n^2 **e)** 2^n **f)** $n!$

16. What is the largest n for which one can solve within a day using an algorithm that requires $f(n)$ bit operations, where each bit operation is carried out in 10^{-11} seconds, with these functions $f(n)$?

a) $\log n$ **b)** $1000n$ **c)** n^2
d) $1000n^2$ **e)** n^3 **f)** 2^n
g) 2^{2n} **h)** 2^{2^n}

17. What is the largest n for which one can solve within a minute using an algorithm that requires $f(n)$ bit operations, where each bit operation is carried out in 10^{-12} seconds, with these functions $f(n)$?

a) $\log \log n$ **b)** $\log n$ **c)** $(\log n)^2$
d) $1000000n$ **e)** n^2 **f)** 2^n
g) 2^{n^2}

18. How much time does an algorithm take to solve a problem of size n if this algorithm uses $2n^2 + 2^n$ operations, each requiring 10^{-9} seconds, with these values of n?

a) 10 **b)** 20 **c)** 50 **d)** 100

19. How much time does an algorithm using 2^{50} operations need if each operation takes these amounts of time?

a) 10^{-6} s **b)** 10^{-9} s **c)** 10^{-12} s

20. What is the effect in the time required to solve a problem when you double the size of the input from n to $2n$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these function? [Express your answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

a) $\log \log n$ **b)** $\log n$ **c)** $100n$
d) $n \log n$ **e)** n^2 **f)** n^3
g) 2^n

21. What is the effect in the time required to solve a problem when you increase the size of the input from n to $n + 1$, assuming that the number of milliseconds the algorithm uses to solve the problem with input size n is each of these function? [Express your answer in the simplest form possible, either as a ratio or a difference. Your answer may be a function of n or a constant.]

a) $\log n$ **b)** $100n$ **c)** n^2
d) n^3 **e)** 2^n **f)** 2^{n^2}
g) $n!$

22. Determine the least number of comparisons, or best-case performance,

a) required to find the maximum of a sequence of n integers, using Algorithm 1 of Section 3.1.

b) used to locate an element in a list of n terms with a linear search.

c) used to locate an element in a list of n terms using a binary search.

23. Analyze the average-case performance of the linear search algorithm, if exactly half the time the element x is not in the list and if x is in the list it is equally likely to be in any position.

24. An algorithm is called **optimal** for the solution of a problem with respect to a specified operation if there is no algorithm for solving this problem using fewer operations.

a) Show that Algorithm 1 in Section 3.1 is an optimal algorithm with respect to the number of comparisons of integers. [*Note:* Comparisons used for bookkeeping in the loop are not of concern here.]

b) Is the linear search algorithm optimal with respect to the number of comparisons of integers (not including comparisons used for bookkeeping in the loop)?

25. Describe the worst-case time complexity, measured in terms of comparisons, of the ternary search algorithm described in Exercise 27 of Section 3.1.

26. Describe the worst-case time complexity, measured in terms of comparisons, of the search algorithm described in Exercise 28 of Section 3.1.

27. Analyze the worst-case time complexity of the algorithm you devised in Exercise 29 of Section 3.1 for locating a mode in a list of nondecreasing integers.

28. Analyze the worst-case time complexity of the algorithm you devised in Exercise 30 of Section 3.1 for locating all modes in a list of nondecreasing integers.

29. Analyze the worst-case time complexity of the algorithm you devised in Exercise 31 of Section 3.1 for finding the first term of a sequence of integers equal to some previous term.

30. Analyze the worst-case time complexity of the algorithm you devised in Exercise 32 of Section 3.1 for finding all terms of a sequence that are greater than the sum of all previous terms.

31. Analyze the worst-case time complexity of the algorithm you devised in Exercise 33 of Section 3.1 for finding the first term of a sequence less than the immediately preceding term.

32. Determine the worst-case complexity in terms of comparisons of the algorithm from Exercise 5 in Section 3.1 for determining all values that occur more than once in a sorted list of integers.

33. Determine the worst-case complexity in terms of comparisons of the algorithm from Exercise 9 in Section 3.1 for determining whether a string of n characters is a palindrome.

34. How many comparisons does the selection sort (see preamble to Exercise 41 in Section 3.1) use to sort n items? Use your answer to give a big-O estimate of the complexity of the selection sort in terms of number of comparisons for the selection sort.

35. Find a big-O estimate for the worst-case complexity in terms of number of comparisons used and the number of terms swapped by the binary insertion sort described in the preamble to Exercise 47 in Section 3.1.

36. Show that the greedy algorithm for making change for n cents using quarters, dimes, nickels, and pennies has $O(n)$ complexity measured in terms of comparisons needed.

Exercises 37 and 38 deal with the problem of scheduling the most talks possible given the start and end times of n talks.

37. Find the complexity of a brute-force algorithm for scheduling the talks by examining all possible subsets of the talks. [*Hint:* Use the fact that a set with n elements has 2^n subsets.]

38. Find the complexity of the greedy algorithm for scheduling the most talks by adding at each step the talk with the earliest end time compatible with those already scheduled (Algorithm 7 in Section 3.1). Assume that the talks are not already sorted by earliest end time and assume that the worst-case time complexity of sorting is $O(n \log n)$.

39. Describe how the number of comparisons used in the worst case changes when these algorithms are used to search for an element of a list when the size of the list doubles from n to $2n$, where n is a positive integer.

a) linear search **b)** binary search

40. Describe how the number of comparisons used in the worst case changes when the size of the list to be sorted doubles from n to $2n$, where n is a positive integer when these sorting algorithms are used.

a) bubble sort **b)** insertion sort

c) selection sort (described in the preamble to Exercise 41 in Section 3.1)

d) binary insertion sort (described in the preamble to Exercise 47 in Section 3.1)

An $n \times n$ matrix is called **upper triangular** if $a_{ij} = 0$ whenever $i > j$.

41. From the definition of the matrix product, describe an algorithm in English for computing the product of two upper triangular matrices that ignores those products in the computation that are automatically equal to zero.

42. Give a pseudocode description of the algorithm in Exercise 41 for multiplying two upper triangular matrices.

43. How many multiplications of entries are used by the algorithm found in Exercise 41 for multiplying two $n \times n$ upper triangular matrices?

In Exercises 44–45 assume that the number of multiplications of entries used to multiply a $p \times q$ matrix and a $q \times r$ matrix is pqr.

44. What is the best order to form the product **ABC** if **A**, **B**, and **C** are matrices with dimensions 3×9, 9×4, and 4×2, respectively?

45. What is the best order to form the product **ABCD** if **A**, **B**, **C**, and **D** are matrices with dimensions 30×10, 10×40, 40×50, and 50×30, respectively?.

***46.** In this exercise we deal with the problem of **string matching**.

a) Explain how to use a brute-force algorithm to find the first occurrence of a given string of m characters, called the **target**, in a string of n characters, where $m \leq n$, called the **text**. [*Hint:* Think in terms of finding a match for the first character of the target and checking successive characters for a match, and if they do not all match, moving the start location one character to the right.]

b) Express your algorithm in pseudocode.

c) Give a big-O estimate for the worst-case time complexity of the brute-force algorithm you described.