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.

\[
I := 0
\]
\[
\text{for } j := 1 \text{ to } 3
\]
\[
I := I + j
\]

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

\[
I := 0
\]
\[
\text{for } j := 1 \text{ to } 3
\]
\[
I := I + j
\]

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).

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 if statement).

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. 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.

7. Suppose that an element is known to be among the first four elements of 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 $x^k$ starting with $x$ and successively squaring (to find $x^2$, $x^4$, and so on). Is this a more efficient way to find $x^k$ 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 $S$.

\[
\text{function} \quad \text{count}(S, n)
\]

\[
\text{return} \quad \text{count}_n(S)
\]

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 $S_j$ to determine whether $k$ also belongs to $S_i$.]

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 $M = (m_{ij})$ where $m_{ij}$ is the minimum term in the sequence of integers $a_i, a_{i+1}, \ldots, a_j$ and $m_{ij} = 0$ otherwise.

\[
\text{initialize } M \text{ so that } m_{ij} = a_i \text{ if } j \geq i \text{ and } m_{ij} = 0 \text{ otherwise}
\]

\[
\text{for } i := 1 \text{ to } n
\]

\[
\text{for } j := i \text{ to } n
\]

\[
\text{for } k := i \text{ to } j
\]

\[
M_{ij} := \min(M_{ij}, a_k)
\]

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

b) Show that this algorithm uses $O(n^3)$ comparisons to compute the matrix $M$. Using this fact and part (a), conclude that the algorithms uses $8n^3$ comparisons.

13. The conventional algorithm for evaluating a polynomial $a_0x^n + a_1x^{n-1} + \cdots + a_n$ at $x = c$ can be expressed in pseudocode by

procedure polynomial ($c, a_0, a_1, \ldots, a_n$: real numbers)
  $\text{power} := 1$
  $y := a_0$
  for $i := 1$ to $n$
    $y := y + c \cdot \text{power}$
    $\text{power} := \text{power} \cdot x$
  return $y$

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 $a_0x^n + a_1x^{n-1} + \cdots + a_i x + a_0$ at $x = c$.

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

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 $10^{-12}$ seconds, with these functions $f(n)$?

a) $\log n$

b) $1000n$

c) $n^2$

d) $1000n^2$

e) $n^3$

f) $2n$

g) $2n^2$

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) $\log n$

b) $\log n$

c) $(\log n)^2$

d) $1000000n$

e) $n^2$

f) $2^n$

g) $2^{10}$

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 $10^{-12}$ seconds, with these values of $n$?

a) 10

b) 20

c) 50

d) 100

e) 1000

19. How much time does an algorithm using $2^{10}$ operations need if each operation takes these amounts of time?

a) $10^{-6}$ s

b) $10^{-9}$ s

c) $10^{-12}$ s

d) $10^{-10}$ s

e) $10^{-3}$ s

f) $10^{-1}$ s

g) $2^n$

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 using a binary search.

c) used to locate an element in a list of $n$ terms using a linear 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. [Hint: 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 (described in the preamble to Exercise 41 in Section 3.1) use to sort a list of nondecreasing integers?

35. 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

36. 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 is halved from \( n \) to \( n/2 \), where \( n \) is a positive integer.

37. Find the complexity of the greedy 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 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) 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 \( i > j \).

40. 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.

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

42. 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 \( pr \).

43. What is the best order to form the product \( ABC \) if \( A, B, \) and \( C \) are matrices with dimensions \( 3 \times 9, 9 \times 4, \) and \( 4 \times 2 \), respectively?

44. What is the best order to form the product \( ABCD \) if \( A, B, \) \( C, \) and \( D \) are matrices with dimensions \( 30 \times 30, 10 \times 40, 40 \times 50, \) and \( 50 \times 30 \), 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.
**Key Terms and Results**

**TERMS**
- **algorithm**: a finite sequence of precise instructions for performing a computation or solving a problem
- **searching algorithm**: the problem of locating an element in a list
- **linear search algorithm**: a procedure for searching a list element by element
- **binary search algorithm**: a procedure for searching an ordered list by successively splitting the list in half
- **sorting**: the reordering of the elements of a list into prescribed order
- **time complexity**: a witness to the relationship between the complexity of an algorithm and the size of the input
- **worst-case time complexity**: the greatest amount of time required for an algorithm to solve a problem
- **average-case time complexity**: the average amount of time required for an algorithm to solve a problem
- **space complexity**: the amount of space in computer memory required for an algorithm to solve a problem
- **algorithmic paradigm**: a general approach for constructing algorithms based on a particular concept
- **brute force**: the algorithmic paradigm based on constructing algorithms for solving problems in a naive manner from the statement of the problem and definitions

**greedy algorithm**: an algorithm that makes the best choice at each step according to some specified condition
**tractable problem**: a problem for which there is a worst-case polynomial-time algorithm that solves it
**intractable problem**: a problem for which no worst-case polynomial-time algorithm exists for solving it
**solvable problem**: a problem that can be solved by an algorithm
**unsolvable problem**: a problem that cannot be solved by an algorithm

**RESULTS**
- **linear and binary search algorithms**: (given in Section 3.1)
  - **bubble sort**: a sorting that uses passes where successive items are interchanged if they in the wrong order
  - **insertion sort**: a sorting that at the jth step inserts the jth element into the correct position in the list, when the first j - 1 elements of the list are already sorted

The linear search has $O(n)$ worst case time complexity. The binary search has $O(log n)$ worst case time complexity. The bubble and insertion sorts have $O(n^2)$ worst case time complexity.

**Review Questions**

1. a) Define the term algorithm.
   b) What are the different ways to describe algorithms?
   c) What is the difference between an algorithm for solving a problem and a computer program that solves this problem?

2. a) Describe, using English, an algorithm for finding the largest integer in a list of n integers.
   b) Express this algorithm in pseudocode.
   c) How many comparisons does the algorithm use?

3. a) State the definition of the fact that $f(n)$ is $O(g(n))$.
   b) Prove or disprove that $n^2 + 18n + 107$ is $O(n^3)$.
   c) Use the definition of the fact that $f(n)$ is $O(g(n))$ directly to prove or disprove that $n^3$ is $O(n^2 + 18n + 107)$.

4. List these functions so that each function is big-$O$ of the next function in the list: $(log n)^2, n^3, 1000000, \sqrt{n}, 100n + 101, 3^n, n!n^2$.

5. a) How can you produce a big-$O$ estimate for a function that is the sum of different terms where each term is the product of several functions?
   b) Give a big-$O$ estimate for the function $f(n) = |a^n - 1| + 1 + (a^{n^2} - 10000n^3)$ for the function $g$ in your estimate $f(n) = O(g(n))$ use a simple function of smallest possible order.

6. a) Define what the worst-case time complexity, average-case time complexity, and best-case time complexity (in terms of comparisons) mean for an algorithm that finds the smallest integer in a list of n integers.
   b) What are the worst-case, average-case, and best-case time complexities, in terms of comparisons, of the algorithm that finds the smallest integer in a list of n integers by comparing each of the integers with the smallest integer found so far?
Supplementary Exercises

7. a) Describe the linear search and binary search algorithm for finding an integer in a list of integers in increasing order.
   b) Compare the worst-case time complexities of these two algorithms.
   c) Is one of these algorithms always faster than the other (measured in terms of comparisons)?

8. a) Describe the bubble sort algorithm.
   b) Use the bubble sort algorithm to sort the list 5, 2, 4, 1, 3.
   c) Give a big-O estimate for the number of comparisons used by the bubble sort.
   d) Use the insertion sort algorithm to sort the list 2, 5, 1, 4, 3.
   e) Give a big-O estimate for the number of comparisons used by the insertion sort.

9. a) Describe the insertion sort algorithm.

10. a) Devise an algorithm that finds the closest pair of integers in a list of integers, and determine the worst-case complexity of your algorithm.
    b) Devise an efficient algorithm for finding the second largest element in a sequence of n elements and determine the worst-case complexity of your algorithm.
    c) Devise an algorithm that finds all equal pairs of sums of two terms of a sequence of n numbers, and determine the worst-case complexity of your algorithm.

11. Describe the concept of a greedy algorithm.

12. Explain what it means for a problem to be tractable and what it means for a problem to be solvable.

Supplementary Exercises

1. a) Describe an algorithm for locating the last occurrence of the largest number in a list of integers.
   b) Estimate the number of comparisons used.

2. a) Describe an algorithm for finding the first and second largest elements in a list of integers.
   b) Estimate the number of comparisons used.

3. a) Give an algorithm to determine whether a bit string contains a pair of consecutive zeros.
   b) How many comparisons does the algorithm use?

4. a) Suppose that a list contains integers that are in order of largest to smallest and an integer can appear repeatedly in this list. Devise an algorithm that locates all occurrences of an integer x in the list.
   b) Estimate the number of comparisons used.

5. a) Adapt Algorithm 1 in Section 3.1 to find the maximum and the minimum of a sequence of n elements by employing a temporary maximum and a temporary minimum that is updated as each successive element is examined.
   b) Describe the algorithm from part (a) in pseudocode.
   c) How many comparisons of elements of the sequence are carried out by this algorithm? (Do not count comparisons used to determine whether the end of the sequence has been reached.) How does this compare to the number of comparisons used by the algorithm in Exercise 5?

6. a) Describe in detail (and in English) the steps of an algorithm that finds the maximum and minimum of a sequence of n elements by examining pairs of successive elements, keeping track of a temporary maximum and a temporary minimum. If n is odd, both the temporary maximum and temporary minimum should initially equal the first term, and if n is even, the temporary minimum and temporary maximum should be found by comparing the initial two elements. The temporary maximum and temporary minimum should be updated by comparing them with the maximum and minimum of the pair of elements being examined.
   b) Express the algorithm described in part (a) in pseudocode.

7. Devise an efficient algorithm for finding the second largest element in a sequence of n elements and determine the worst-case complexity of your algorithm. Hint: Sort the sequence. Use the fact that sorting can be done with worst-case time complexity $O(n \log n)$.

The shaker sort (or bidirectional bubble sort) successively compares pairs of adjacent elements, exchanging them if they are out of order, and alternately passing through the list from the beginning to the end and then from the end to the beginning until no exchanges are needed.

8. Devide the shaker sort to sort the list 3, 5, 1, 4, 6, 2.

9. Express the shaker sort in pseudocode.

10. Show that $(n \log n + n^2)^3$ is $O(n^8)$.

11. Show that $8x^3 + 12x + 100 \log x$ is $O(x^3)$.

12. Give a big-O estimate for $(x^2 + x)(\log x^2)$.

13. Find a big-O estimate for $\sum_{j=1}^{n} j(j + 1)$.

14. Explain why the shaker sort is efficient for sorting lists that are already in close to the correct order.

15. Show that $(n \log n + n^2)^3$ is $O(n^8)$.

16. Show that $8x^3 + 12x + 100 \log x$ is $O(x^3)$.

17. Give a big-O estimate for $(x^2 + x)(\log x^2)$.

18. Find a big-O estimate for $\sum_{j=1}^{n} j(j + 1)$.

19. Show that $n!$ is not $O(2^n)$.

20. Show that $n^n$ is not $O(n!)$. 