Then we cut along line AB. The greatest common measure of the remaining rectangle $ABDE$ is the same as the greatest common measure of the original (this is essentially just Theorem 2). We continue this process—obtaining smaller and smaller rectangles—until we obtain a square. If the sides of the original rectangle are not commensurate (i.e., if their ratio is not a rational number), then the process will continue forever.

31. We use the bisection method, constantly narrowing an interval in which the desired solution lies. Note that throughout the following algorithm, $\sqrt{2} \in (l, u)$. Thus when the algorithm terminates (as it surely does, since $u - l$ decreases by a factor of 2 on every iteration), $|\sqrt{2} - l| < 10^{-n}$. Thus the algorithm computes $\sqrt{2}$ with an error of at most 1 in the nth decimal place.

1. Set l equal to 1, and set u equal to 2.

2. If $u - l < 10^{-n}$, then output l and stop. Otherwise continue with step 3.

3. Set x equal to $(u + l)/2$. If $x^2 < 2$, then replace l by x. Otherwise replace u by x. In either case, return to step 2 and continue from there.

33. We need to check for reflexivity and transitivity in both cases, symmetry in part (a), and antisymmetry in part (b). Form the matrix representing R, i.e., the n by n matrix $M_R = (m_{ij})$ in which $m_{ij} = 1$ if iRj and $m_{ij} = 0$ if not. To check for reflexivity, simply check to see if $m_{ii} = 1$ for all i. To check for transitivity, compute $M_{R \circ R}$, defined in Section 3.3 to be the Boolean matrix product of M with itself. The relation is transitive if and only if $M_{R \circ R} = M$ (given that R is reflexive). To check for symmetry, we need to see that each below-the-diagonal entry, m_{ij} for $i < j$, has the same value as the above-the-diagonal entry symmetrically located across the main diagonal, namely m_{ji}. Finally, to check for antisymmetry, we need to check that for each below-the-diagonal entry (m_{ij} for $i < j$) that has the value 1, the above-the-diagonal entry symmetrically located across the main diagonal (namely m_{ji}) has the value 0.

SECTION 4.2 Pseudocode Description of Algorithms

1.

```
procedure small_sum(A : list of integers)
    { A = (a_1, a_2, ..., a_n); sum is the sum of the elements of A less than 100 }
    sum ← 0
    for i ← 1 to n do
        if a_i < 100 then sum ← sum + a_i
    return(sum)
```

3. (a) **procedure** *middle*(a, b, c : distinct natural numbers)
{finds the middle value among a, b, and c}
if $a < b$ **then**
if $c < a$ **then return**(a)
else if $b < c$ **then return**(b)
else return(c)
else if $c < b$ **then return**(b)
else if $a < c$ **then return**(a)
else return(c)

(b) **procedure** *add_fractions*(a, b, c, d : integers)
{assume that $b > 0$ and $d > 0$; returns the numerator and denominator of $(a/b) + (c/d)$}
return($ad + bc, bd$)

(c) **procedure** *large_square*(n : positive integer)
{computes largest perfect square $\leq n$}
$i \leftarrow 1$
while $i^2 \leq n$ **do**
$i \leftarrow i + 1$
return($(i-1)^2$)

(d) **procedure** *big_word*($w_1, w_2, \ldots, w_n$: strings)
{finds a longest word in the list, and its length}
maxword $\leftarrow w_1$
maxlength $\leftarrow$ length(w_1)
for $i \leftarrow 2$ **to** n **do**
if length(w_i) $>$ *maxlength* **then**
begin
maxword $\leftarrow w_i$
maxlength $\leftarrow$ length(w_i)
end
return(*maxlength*, *maxword*)

5. (a)
$i \leftarrow m$
while $i \leq n$ **do**
begin
statement
$i \leftarrow i + 1$
end

(b)
$i \leftarrow n$
while $i \geq 1$ **do**
begin
statement
$i \leftarrow i - 1$
end

(c)
```
i ← m
while i ≤ n do
   begin
      statement
      i ← i + k
   end
```

7. Suppose that d is a common divisor of x, y, and z. Then d is a common divisor of x and y, so by Theorem 1 in Section 4.1, d is a common divisor of $\gcd(x,y)$ and z. Conversely, if d is a divisor of $\gcd(x,y)$, then by Exercise 15 in Section 4.1, d divides both x and y. Hence d is a common divisor of x, y, and z. Thus we have shown that the sets of divisors about which the two sides of the equation speak are identical, and so their greatest elements are equal.

9.
```
procedure naive_gcd(x, y : positive integers)
   {computes gcd(x, y) by brute force}
      d ← min(x, y)
      while ¬(d | x ∧ d | y) do
         d ← d − 1
      return(d)
```

11. This algorithm finds the smallest factor of N greater than 1. It will necessarily be prime.

```
procedure small_prime(N : integer > 1)
   {finds smallest prime factor of N}
      d ← 2
      while d ∤ N do
         d ← d + 1
      return(d)
```

13. The smallest nontrivial factor at each stage will necessarily be prime. This procedure finds the highest power of this prime which divides N, then goes on to the next smallest prime factor.

```
procedure factorization(N : integer > 1)
    {the output will be a set of pairs, L = {(p_1, e_1), (p_2, e_2), ..., (p_k, e_k)},
     where N = p_1^{e_1} p_2^{e_2} ... p_k^{e_k} is the prime factorization}
      k ← 0 {k will be the number of factors}
      p ← 2 {potential prime factor}
      while N > 1 do
         begin
            e ← 0 {e will be the largest power of p dividing N}
            while p | N do
               begin
                  e ← e + 1
                  N ← N/p
               end
            if e > 0 then {found a new prime factor}
               begin
                  k ← k + 1
                  p_k ← p; e_k ← e
               end
            p ← p + 1 {go on to next potential prime factor}
         end
      return(L)
```

15.

```
procedure mode(s : string of decimal digits)
    {finds the digit that occurs most often in s; we assume that the
     answer is unique; uses procedure tabulate from this section}
      count ← tabulate(s)
      maxdigit ← 0
      for i ← 1 to 9 do
         if count(i) > count(maxdigit) then maxdigit ← i
      return(maxdigit)
```

17.

```
procedure order_4(a, b, c, d : distinct numbers)
    {returns the same numbers, put into increasing order}
      if a > b then interchange a and b {now a < b}
      if b > c then interchange b and c {now a, b < c}
      if c > d then interchange c and d {now d is the largest}
      if a > b then interchange a and b {now a < b}
      if b > c then interchange b and c {now a, b < c}
      if a > b then interchange a and b {now a < b < c < d}
      return(a, b, c, d)
```

19.

```
procedure exercise_19(A : list of nonzero real numbers)
    {assume that A = (a_1, a_2, ..., a_n) has at least
     one positive and one negative number}
      pos_ ← count0; neg_ ← count0
      pos_ ← sum0; neg_ ← sum0
      for i ← 1 to n do
          if a_i > 0 then
                  begin
                      pos_ ← countpos_count + 1
                      pos_ ← sumpos_sum + a_i
                  end
              else
                  begin
                      neg_ ← countneg_count + 1
                      neg_ ← sumneg_sum − a_i  {since |a_i| = −a_i }
                  end
      return((pos_sum/pos_count) − (neg_sum/neg_count))
```

21.

```
procedure circulant(a_1, a_2, ..., a_n)
    {constructs circulant matrix (circ) whose first row is (a_1, a_2, ..., a_n)}
      for i ← 1 to n do
          for j ← 1 to n do
              begin
                  k ← i − 1 + j  {this does the shifting}
                  if k > n then k ← k − n
                  circ(i, j) ← a_k
              end
      return(circ)
```

23.

```
procedure phi(n : positive integer)
    {computes the number of numbers less than n
     that are relatively prime to n}
      count ← 0
      for i ← 1 to n do {does i meet the condition gcd(i, n) = 1?}
          if euclid(i, n) = 1 then count ← count + 1
      return(count)
```

25.
```
procedure modes(s : string of decimal digits)
    {finds the digits that occur most often in s; answer is given as a
     set L of digits; uses procedure tabulate from this section}
      count ← tabulate(s)
      maxdigit ← 0
      L ← {0} {the tentative answer}
      for i ← 1 to 9 do
          if count(i) = count(maxdigit) then
                  L ← L ∪ {maxdigit} {i occurs as often as maxdigit}
              else if count(i) > count(maxdigit) then
                  begin {i alone is the new tentative answer}
                      maxdigit ← i
                      L ← {i}
                  end
      return(L)
```

27.
```
procedure substring(s,t : strings)
    {s = s_1 s_2 ... s_n and t = t_1 t_2 ... t_m; determines whether s is a substring of t}
      for j ← 0 to m − n do
          begin {see whether s occurs starting at t_{j+1}}
              matched ← true {until we find a difference}
              for i ← 1 to n do
                  if s_i ≠ t_{j+i} then matched ← false
              if matched then return(true) {otherwise try next location}
          end
      return(false) {no match found}
```

29.
```
procedure length_of_repeat(n : positive integer)
    {finds the length of the repeat in the decimal expansion of 1/n;
     the array R holds the successive remainders in the long division
     process; when a remainder recurs, we are done}
      R(1) ← 1 {the numerator is 1; it is the first remainder}
      l ← 1
      while true do
          begin
              l ← l + 1
              R(l) ← 10 · R(l − 1) mod n
                      {bring down next 0 and find new remainder}
              if R(l) = 0 then return(0) {decimal is finite}
              for i ← 1 to l − 1 do {has this remainder occurred before?}
                  if R(i) = R(l) then return(l − i)
          end
```

SECTION 4.3 Efficiency of Algorithms

1. **(a)** $i \leftarrow 1$; *eating* $\neq a$; $i \leftarrow 2$; *eating* $\neq$ *curds*; $i \leftarrow 3$; *eating* $=$ *eating*; return 3

(b) $i \leftarrow 1$; *more* $\neq a$; $i \leftarrow 2$; *more* $\neq$ *curds*; $i \leftarrow 3$; *more* $\neq$ *eating*; $i \leftarrow 4$; *more* $\neq$ *her*; $i \leftarrow 5$; *more* $\neq$ *little*; $i \leftarrow 6$; *more* $\neq$ *miss*; $i \leftarrow 7$; *more* $\neq$ *muffet*; $i \leftarrow 8$; *more* $\neq$ *on*; $i \leftarrow 9$; *more* $\neq$ *sat*; $i \leftarrow 10$; *more* $\neq$ *tuffet*; loop finished, return 0

(c) *low* $\leftarrow 1$, *high* $\leftarrow 10$, *middle* $\leftarrow 5$; *eating* $\preceq$ *little*, so *high* $\leftarrow 5$, *middle* $\leftarrow 3$; *eating* $\preceq$ *eating*, so *high* $\leftarrow 3$, *middle* $\leftarrow 2$; *eating* $\not\preceq$ *curds*, so *low* $\leftarrow 3$, loop finished; *eating* $=$ *eating*, so return 3

(d) *low* $\leftarrow 1$, *high* $\leftarrow 10$, *middle* $\leftarrow 5$; *more* $\not\preceq$ *little*, so *low* $\leftarrow 6$, *middle* $\leftarrow 8$; *more* $\preceq$ *on*, so *high* $\leftarrow 8$, *middle* $\leftarrow 7$; *more* $\preceq$ *muffet*, so *high* $\leftarrow 7$, *middle* $\leftarrow 6$; *more* $\not\preceq$ *miss*, so *low* $\leftarrow 7$, loop finished; *more* $\neq$ *muffet*, so return 0

3. **(a)** The $4n$ term and the coefficient 10 do not matter when we are dealing with big-oh, so this function is in $O(n^3)$.

(b) Since $\log n < n$, the first term dominates the second, so this function is in $O(n^2)$.

(c) $O(n^{1.5}/(n+1)) = O(n^{1.5}/n) = O(\sqrt{n})$

(d) $O(2^n)$, since $2^n > n^2$ for large n

5. **(a)** $O(n/2) = O(n)$

(b) $O(\log_3 n) = O(\log n)$

(c) $O(1)$, since the loop is iterated only once

7. **(a)** If $(high - low)$ is even, then $(high + low)$ is also even, so $middle = (high + low)/2$. There are now two cases. If *low* is replaced by $middle + 1$, then the new value of $(high - low)$ is

$$high - \left(\frac{high + low}{2} + 1\right) = \frac{high - low - 2}{2} = \frac{high - low}{2} - 1.$$

If *high* is replaced by *middle*, then the new value of $(high - low)$ is

$$\frac{high + low}{2} - low = \frac{high - low}{2}.$$

Hence either $(high - low)$ is cut in half, or else it is cut in half and 1 more is subtracted. On the other hand, if $(high - low)$ is odd, then $(high + low)$ is also odd, so $middle = (high + low - 1)/2$. Again there are two cases. If *low* is replaced by $middle + 1$, then the new value of $(high - low)$ is

$$high - \left(\frac{high + low - 1}{2} + 1\right) = \frac{high - low - 1}{2}.$$

If *high* is replaced by *middle*, then the new value of $(high - low)$ is

$$\frac{high + low - 1}{2} - low = \frac{high - low - 1}{2}.$$

Hence in either case, 1 is subtracted from $(high - low)$ and then the value is cut in half.

(b) We need to show that $(high - low)$ never jumps from a positive number to a negative number. If $(high - low)$ is odd, then $(high - low) \geq 1$, so that the new value of $(high - low)$ is still greater than or equal to 0 by our formula from part (a). If $(high - low)$ is even and not yet 0, then $(high - low) \geq 2$, so again our formula from part (a) guarantees that $(high - low)$ remains nonnegative. Thus $(high - low)$ is a constantly decreasing integer that never becomes negative, so it must eventually equal 0, thereby terminating the loop.

9. From algebra we know that $\log_a n = (\log_b n)/(\log_b a)$. Since $1/(\log_b a)$ is a positive constant, $\log_a n \in O(\log_b n)$.

11. **(a)** Let $C_1 = m^m/m!$, where $m = \lceil k \rceil$. Then for $n > m$ we have

$$\begin{aligned} k^n = k^{n-m} \cdot k^m &\leq m^{n-m} \cdot m^m \\ &\leq n(n-1)\cdots(m+1) \cdot m^m \\ &= n(n-1)\cdots(m+1) \cdot C_1 \cdot m! = C_1 n!, \end{aligned}$$

so $k^n \in O(n!)$.

(b) Since $n! = n(n-1)\cdots 2 \cdot 1 \leq n \cdot n \cdots n \cdot n = n^n$, clearly $n! \in O(n^n)$.

(c) Let $m = \lceil k \rceil$. Let $C_1 = m^m/m!$. Then if $n > m$,

$$\begin{aligned} mC_1(n+1)! &= mC_1(n+1)n(n-1)\cdots(m+1)m! = m(n+1)n(n-1)\cdots(m+1)m^m \\ &\geq m(n+1)m \cdot m \cdots m \cdot m^m = (n+1)m^{n+1} \geq (n+1)k^{n+1}. \end{aligned}$$

Thus $(n+1)! \geq (n+1)\dfrac{1}{mC_1}k^{n+1}$. In particular, $\dfrac{(n+1)!}{k^{n+1}} \geq (n+1)\dfrac{1}{mC_1} \to \infty$ as $n \to \infty$, so $n! \notin O(k^n)$.

(d) $n^n = n \cdot n \cdots n \geq n(n-1)\cdots 2 \cdot n \geq n! \cdot n$. Thus $n^n/n! \geq n \to \infty$ as $n \to \infty$. Therefore $n^n \notin O(n!)$.

13. Let C_1, C_2, k_1, and k_2 be such that $f_1(n) \leq C_1 g(n)$ for all $n \geq k_1$, and $f_2(n) \leq C_2 g(n)$ for all $n \geq k_2$. Let $C = C_1 + C_2$ and let $k = \max(k_1, k_2)$. Then for all $n \geq k$ we have $f_1(n) + f_2(n) \leq C_1 g(n) + C_2 g(n) = Cg(n)$.

15. **(a)** This follows from the fact that under the hypothesis, $f(n)+g(n) \leq g(n)+g(n) = 2g(n)$ for all large n.

(b) This follows from the definition (let $C = m$ and $k = 1$).

17. **(a)** $n + n + \cdots + n$ (n times) $= n^2 \in O(n^2)$

(b) $(n-1) + (n-2) + \cdots + 1 + 0 \leq n^2 \in O(n^2)$

(c) $n + n + \cdots + n$ ($1 + \lfloor \log n \rfloor$ times) $\in O(n \log n)$

(d) $\dfrac{n}{2} + \dfrac{n}{4} + \dfrac{n}{8} + \cdots \leq n \in O(n)$

Hence in either case, 1 is subtracted from $(high - low)$ and then the value is cut in half.

(b) We need to show that $(high - low)$ never jumps from a positive number to a negative number. If $(high - low)$ is odd, then $(high - low) \geq 1$, so that the new value of $(high - low)$ is still greater than or equal to 0 by our formula from part (a). If $(high - low)$ is even and not yet 0, then $(high - low) \geq 2$, so again our formula from part (a) guarantees that $(high - low)$ remains nonnegative. Thus $(high - low)$ is a constantly decreasing integer that never becomes negative, so it must eventually equal 0, thereby terminating the loop.

9. From algebra we know that $\log_a n = (\log_b n)/(\log_b a)$. Since $1/(\log_b a)$ is a positive constant, $\log_a n \in O(\log_b n)$.

11. **(a)** Let $C_1 = m^m/m!$, where $m = \lceil k \rceil$. Then for $n > m$ we have

$$\begin{aligned} k^n &= k^{n-m} \cdot k^m \leq m^{n-m} \cdot m^m \\ &\leq n(n-1)\cdots(m+1) \cdot m^m \\ &= n(n-1)\cdots(m+1) \cdot C_1 \cdot m! = C_1 n!, \end{aligned}$$

so $k^n \in O(n!)$.

(b) Since $n! = n(n-1)\cdots 2 \cdot 1 \leq n \cdot n \cdots n \cdot n = n^n$, clearly $n! \in O(n^n)$.

(c) Let $m = \lceil k \rceil$. Let $C_1 = m^m/m!$. Then if $n > m$,

$$\begin{aligned} mC_1(n+1)! &= mC_1(n+1)n(n-1)\cdots(m+1)m! = m(n+1)n(n-1)\cdots(m+1)m^m \\ &\geq m(n+1)m \cdot m \cdots m \cdot m^m = (n+1)m^{n+1} \geq (n+1)k^{n+1}. \end{aligned}$$

Thus $(n+1)! \geq (n+1)\dfrac{1}{mC_1}k^{n+1}$. In particular, $\dfrac{(n+1)!}{k^{n+1}} \geq (n+1)\dfrac{1}{mC_1} \to \infty$ as $n \to \infty$, so $n! \notin O(k^n)$.

(d) $n^n = n \cdot n \cdots n \geq n(n-1)\cdots 2 \cdot n \geq n! \cdot n$. Thus $n^n/n! \geq n \to \infty$ as $n \to \infty$. Therefore $n^n \notin O(n!)$.

13. Let C_1, C_2, k_1, and k_2 be such that $f_1(n) \leq C_1 g(n)$ for all $n \geq k_1$, and $f_2(n) \leq C_2 g(n)$ for all $n \geq k_2$. Let $C = C_1 + C_2$ and let $k = \max(k_1, k_2)$. Then for all $n \geq k$ we have $f_1(n) + f_2(n) \leq C_1 g(n) + C_2 g(n) = C g(n)$.

15. **(a)** This follows from the fact that under the hypothesis, $f(n)+g(n) \leq g(n)+g(n) = 2g(n)$ for all large n.

(b) This follows from the definition (let $C = m$ and $k = 1$).

17. **(a)** $n + n + \cdots + n$ (n times) $= n^2 \in O(n^2)$

(b) $(n-1) + (n-2) + \cdots + 1 + 0 \leq n^2 \in O(n^2)$

(c) $n + n + \cdots + n$ $(1 + \lfloor \log n \rfloor$ times$) \in O(n \log n)$

(d) $\dfrac{n}{2} + \dfrac{n}{4} + \dfrac{n}{8} + \cdots \leq n \in O(n)$

19. **(a)** First the algorithm finds that 12 is the largest element, so it interchanges 12 and 2, yielding the sequence (2,4,5,8,6,10,12). Next it finds that 10 is the largest element among the first six elements; interchanging 10 and 10 has no effect, so the sequence is unchanged. The next pass reveals that 8 is the largest element among the first five, and so 8 and 6 are interchanged, yielding (2,4,5,6,8,10,12). The next three passes again result in no change, and the sequence is now sorted.

(b) After the outer loop is executed for a given value of *end*, a_{end} is the *end*th largest element in the list, in its correct place. Thus a_n, a_{n-1}, ..., a_2 are all correctly placed, and hence so is a_1.

(c) The comparison step is executed $(n-1)+(n-2)+\cdots+2+1$ times, so the efficiency is $n(n-1)/2 \in O(n^2)$.

21.

```
procedure speedy(A, x)
    {searches A for x, as in linear_search }
        a_0 ← x {the process will find x here if not before}
        i ← n
        while true do
            begin {loop executed at most n + 1 times}
                if x = a_i then return(i) {the only comparison}
                i ← i − 1
            end
```

23. Since $g \notin O(f)$, there is an increasing, infinite sequence of numbers n_1, n_2, ..., such that $g(n_i)/f(n_i) \to \infty$ as $i \to \infty$. On the other hand, there exist positive constants C and k such that $f(n) \le Cg(n)$ for all $n \ge k$. Without loss of generality we may assume that $C \ge 1$. Let the function h be defined by

$$h(n) = \begin{cases} g(n) & \text{if } n = n_i \text{ for some even } i \\ f(n) & \text{otherwise.} \end{cases}$$

Then $h(n) \le \max(g(n), f(n)) \le \max(g(n), Cg(n)) = Cg(n)$ for all $n \ge k$, so $h \in O(g)$. Similarly $h(n) \ge \min(f(n), g(n)) \ge \min(f(n), f(n)/C) \ge f(n)/C$, so $f \in O(h)$. But $h \notin O(f)$ and $g \notin O(h)$, since as $i \to \infty$,

$$\frac{h(n_{2i})}{f(n_{2i})} = \frac{g(n_{2i})}{f(n_{2i})} \to \infty \qquad \text{and} \qquad \frac{g(n_{2i+1})}{h(n_{2i+1})} = \frac{g(n_{2i+1})}{f(n_{2i+1})} \to \infty.$$

25. **(a)** Using L'Hôpital's rule we have

$$\lim_{n\to\infty} \frac{n^a}{\log n} = \lim_{n\to\infty} \frac{an^{a-1}}{(1/n)\log e} = \lim_{n\to\infty} \left(\frac{a}{\log e}\right) n^a = \infty.$$

(b) We want to solve the equation $n^{0.01} = \log n$. Taking logs of both sides yields $0.01 \log n = \log(\log n)$, or $x = 100 \log x$, where $x = \log n$. Solving $x = 100 \log x$ iteratively on a calculator (start with any big guess and repeatedly take log of the display