procedure fastpower(n: positive integer, a: real number)
if n = 1 then return a
else if n is even then return fastpower(n/2, a)²
else return a · fastpower((n − 1)/2, a)²

28. To compute \( f_7 \), Algorithm 7 requires \( f_8 − 1 = 20 \) additions, and Algorithm 8 requires \( 7 − 1 = 6 \) additions.

30. This is essentially just Algorithm 8, with a different operation and different initial conditions.

procedure iterative(n: nonnegative integer)
if n = 0 then y := 1
else
    x := 1
    y := 2
    for i := 1 to n − 1
        z := x · y
        x := y
        y := z
    return y \{ the \( n \)th term of the sequence \}

32. This is very similar to the recursive procedure for computing the Fibonacci numbers. Note that we can combine the three base cases (stopping rules) into one.

procedure sequence(n: nonnegative integer)
if n < 3 then return n + 1
else return sequence(n − 1) + sequence(n − 2) + sequence(n − 3)

34. The iterative algorithm is much more efficient here. If we compute with the recursive algorithm, we end up computing the small values (early terms in the sequence) over and over and over again (try it for \( n = 5 \)).

36. We obtain the answer by computing \( P(m, m) \), where \( P \) is the following procedure, which we obtain simply by copying the recursive definition from Exercise 47 in Section 5.3 into an algorithm.

procedure P(m, n: positive integers)
if m = 1 then return 1
else if n = 1 then return 1
else if m < n then return P(m, m)
else if m = n then return 1 + P(m, m − 1)
else return P(m, n − 1) + P(m − n, n)

38. The following algorithm practically writes itself.

procedure power(w: bit string, i: nonnegative integer)
if i = 0 then return \( \lambda \)
else return w concatenated with power(w, i − 1)

40. If \( i = 0 \), then by definition \( w^i \) is no copies of \( w \), so it is correct to output the empty string. Inductively, if the algorithm correctly returns the \( i \)th power of \( w \), then it correctly returns the \( (i + 1) \)th power of \( w \) by concatenating one more copy of \( w \).

42. If \( n = 3 \), then the polygon is already triangulated. Otherwise, by Lemma 1 in Section 5.2, the polygon has a diagonal; draw it. This diagonal splits the polygon into two polygons, each of which has fewer than \( n \) vertices. Recursively apply this algorithm to triangulate each of these polygons. The result is a triangulation of the original polygon.