LECTURE 1_2
CHAPTER 9: RECURSION

PROBLEM SOLVING AND PROGRAM DESIGN IN C
7TH EDITION

FIGURE 9.1 SPLITTING A PROBLEM INTO SMALLER PROBLEMS

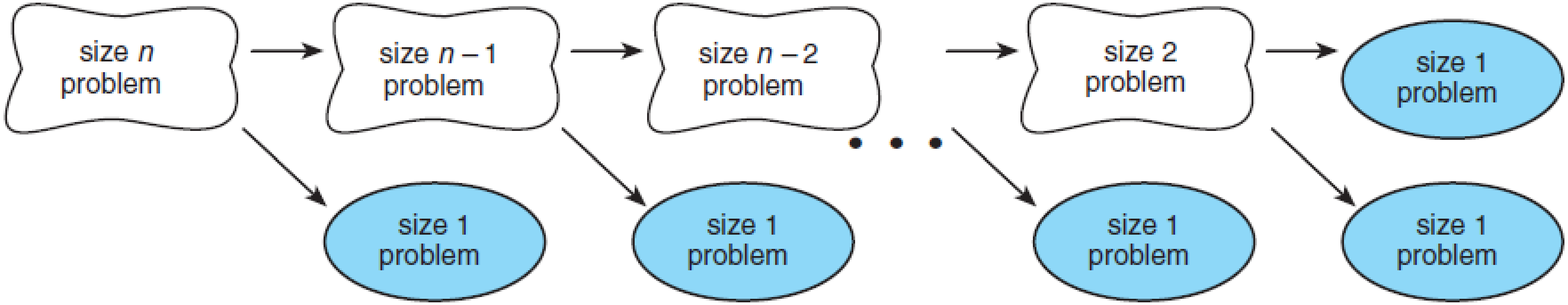
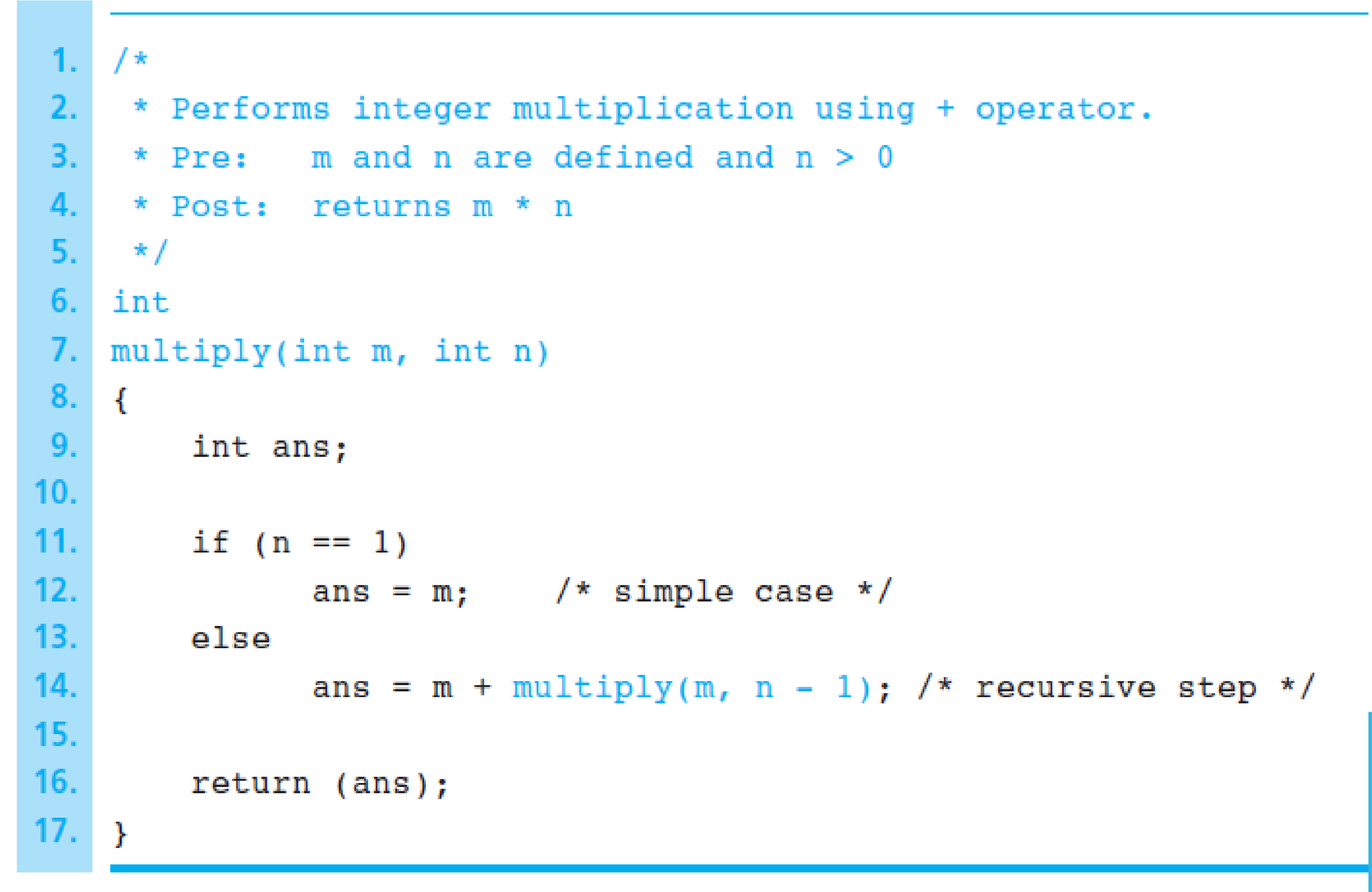


FIGURE 9.2 RECURSIVE FUNCTION MULTIPLY



```
/*
 * Performs integer multiplication using + operator.
 * Pre:   m and n are defined and n > 0
 * Post:  returns m * n
 */
int
multiply(int m, int n)
{
    int ans;

    if (n == 1)
          ans = m;    /* simple case */
    else
          ans = m + multiply(m, n - 1); /* recursive step */

    return (ans);
}
```

FIGURE 9.5 TRACE OF FUNCTION MULTIPLY

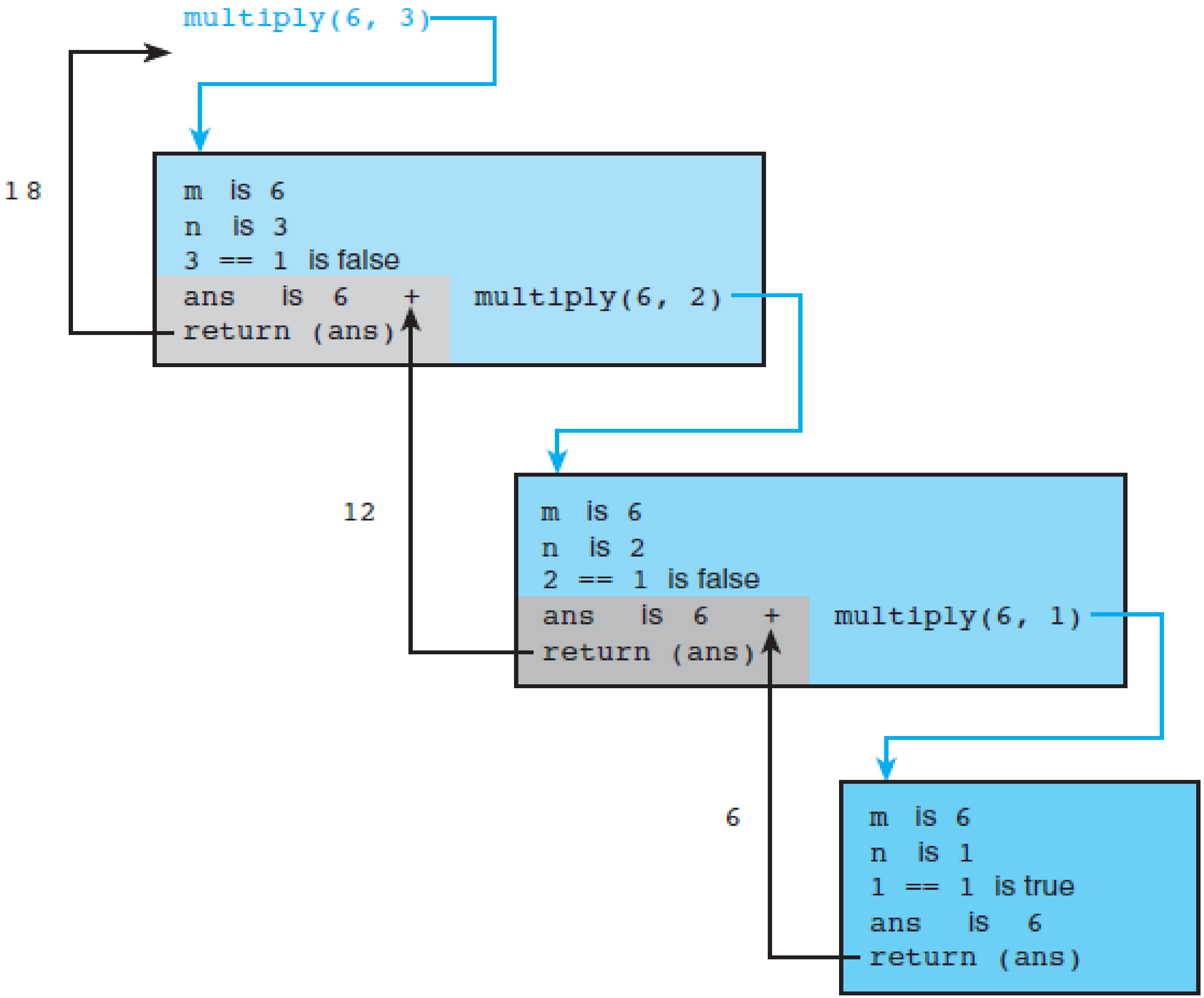
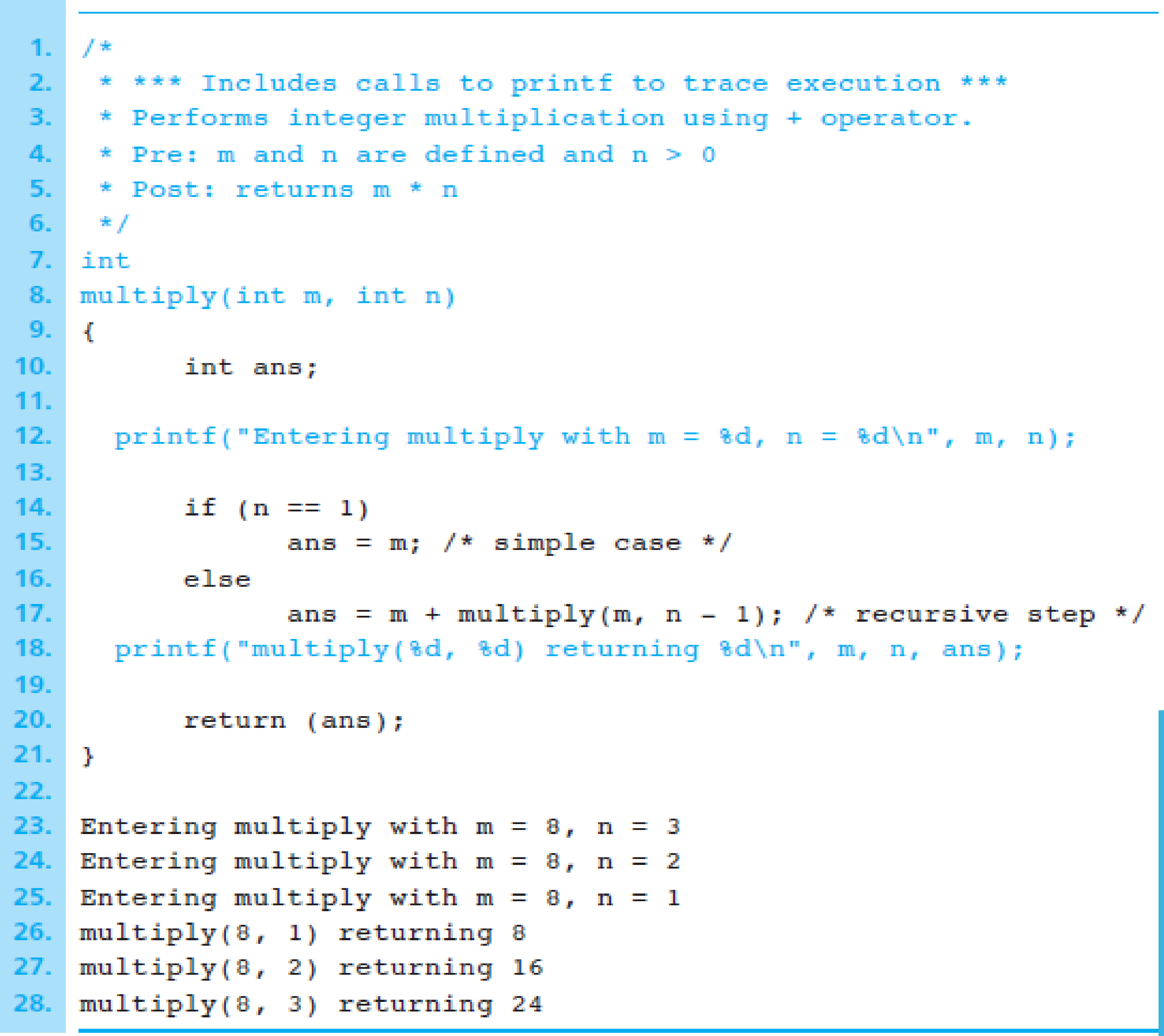


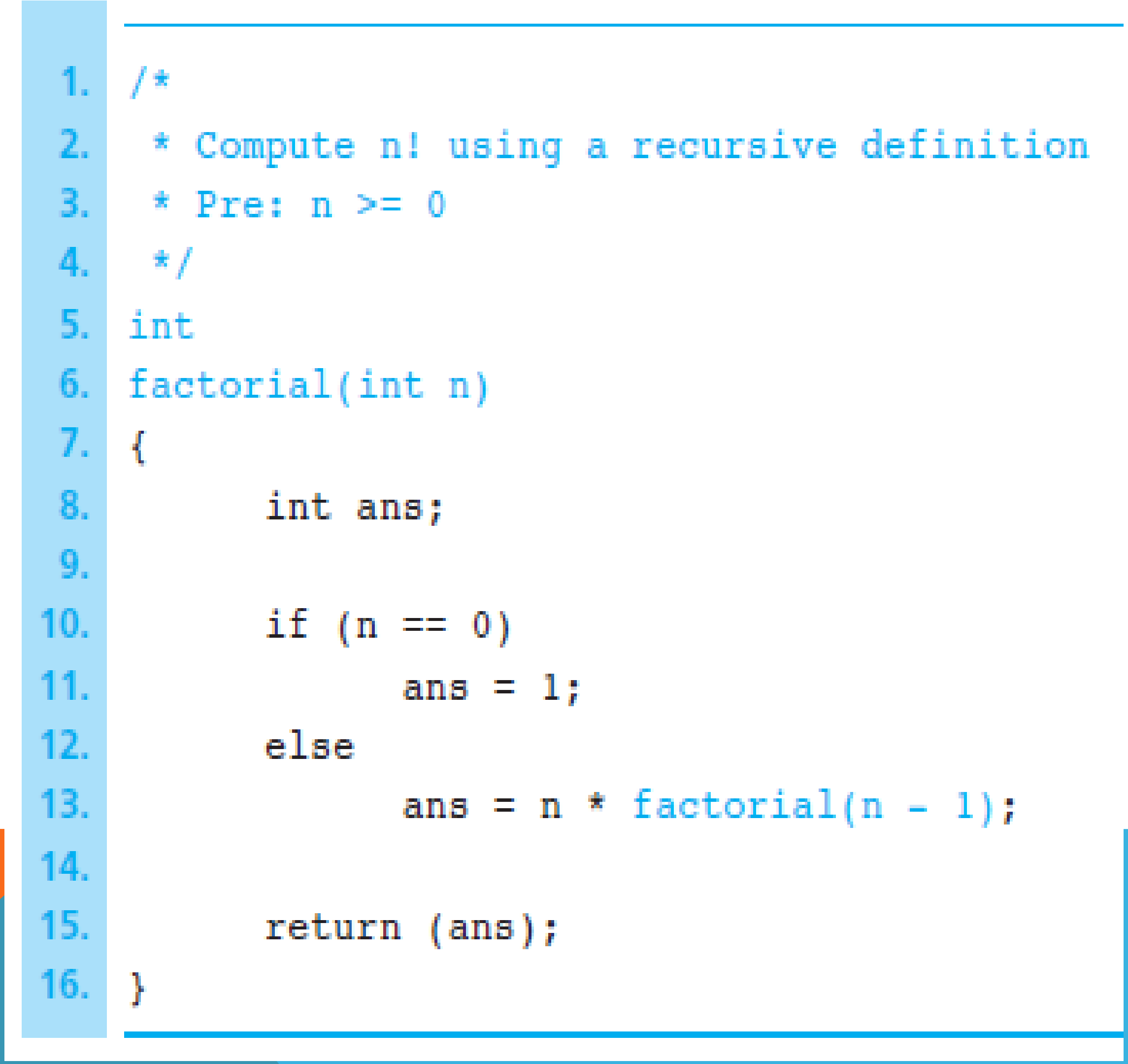
FIGURE 9.9 RECURSIVE FUNCTION MULTIPLY WITH PRINT STATEMENTS TO CREATE TRACE AND OUTPUT FROM MULTIPLY(8, 3)



```c
/*
 * *** Includes calls to printf to trace execution ***
 * Performs integer multiplication using + operator.
 * Pre: m and n are defined and n > 0
 * Post: returns m * n
 */
int
multiply(int m, int n)
{
      int ans;

  printf("Entering multiply with m = %d, n = %d\n", m, n);

      if (n == 1)
            ans = m; /* simple case */
      else
            ans = m + multiply(m, n - 1); /* recursive step */
  printf("multiply(%d, %d) returning %d\n", m, n, ans);

      return (ans);
}
```

```
Entering multiply with m = 8, n = 3
Entering multiply with m = 8, n = 2
Entering multiply with m = 8, n = 1
multiply(8, 1) returning 8
multiply(8, 2) returning 16
multiply(8, 3) returning 24
```

FIGURE 9.10 RECURSIVE FACTORIAL FUNCTION



```
/*
 * Compute n! using a recursive definition
 * Pre: n >= 0
 */
int
factorial(int n)
{
        int ans;

        if (n == 0)
                ans = 1;
        else
                ans = n * factorial(n - 1);

        return (ans);
}
```

FIGURE 9.11 TRACE OF FACT = FACTORIAL(3);

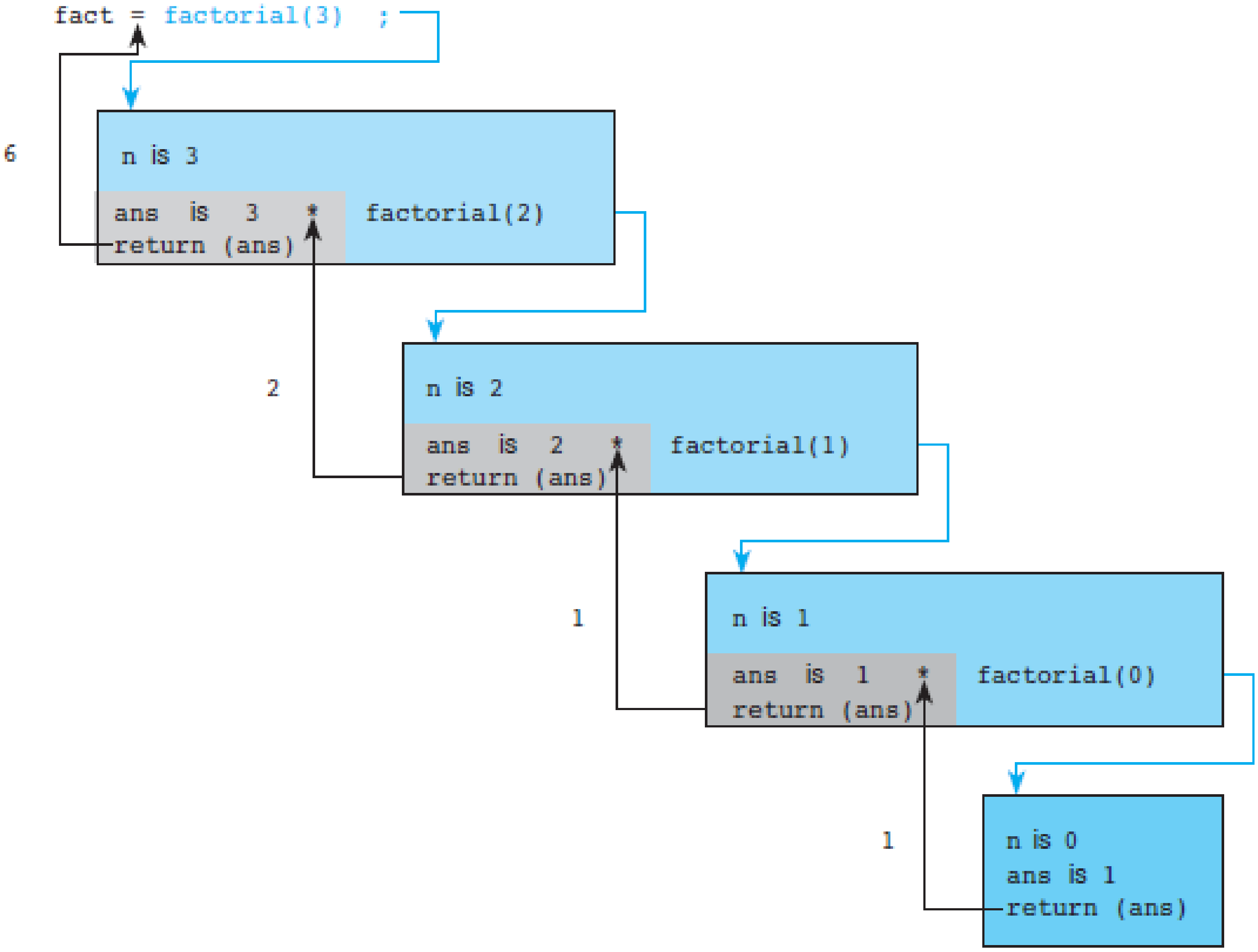
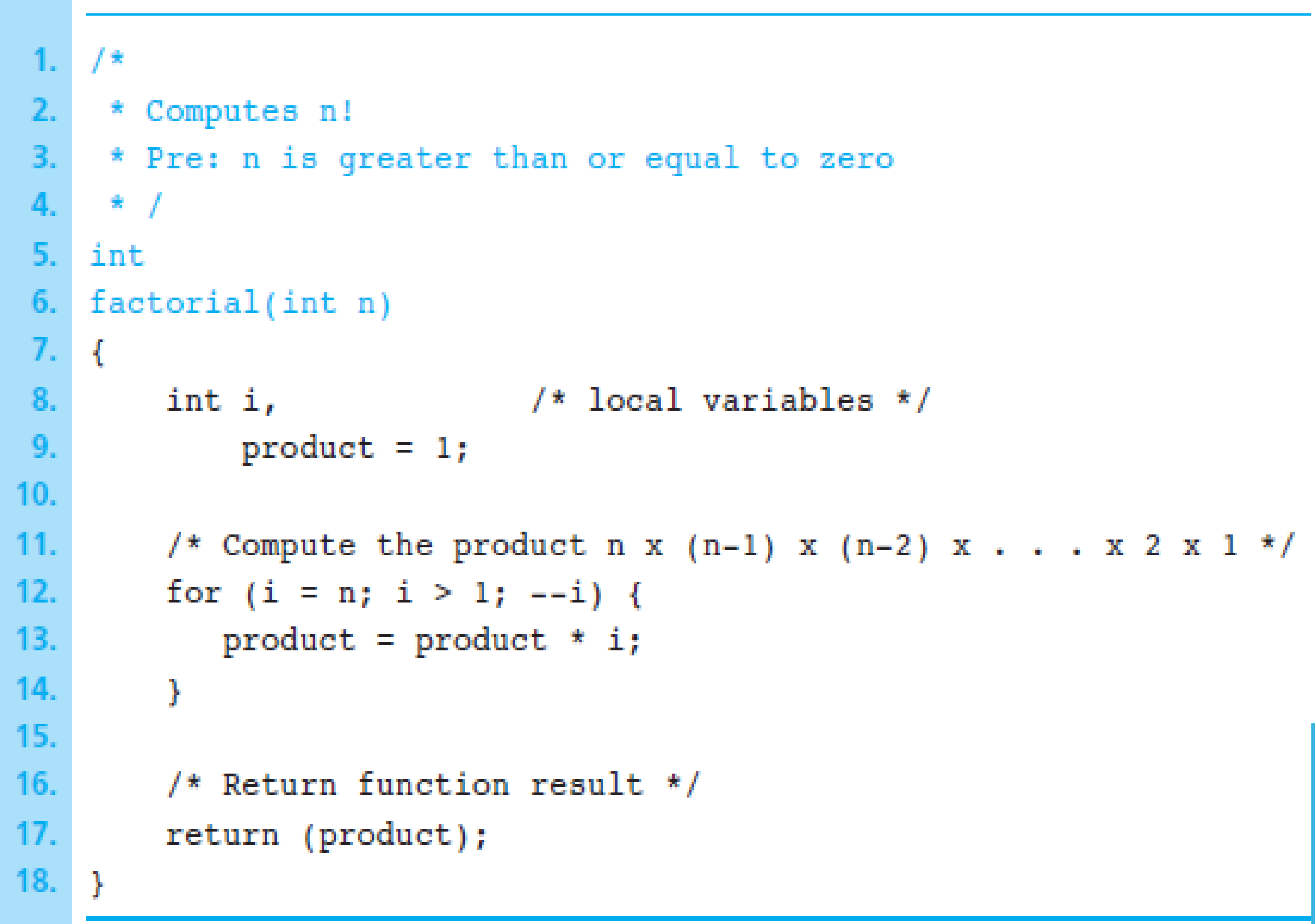
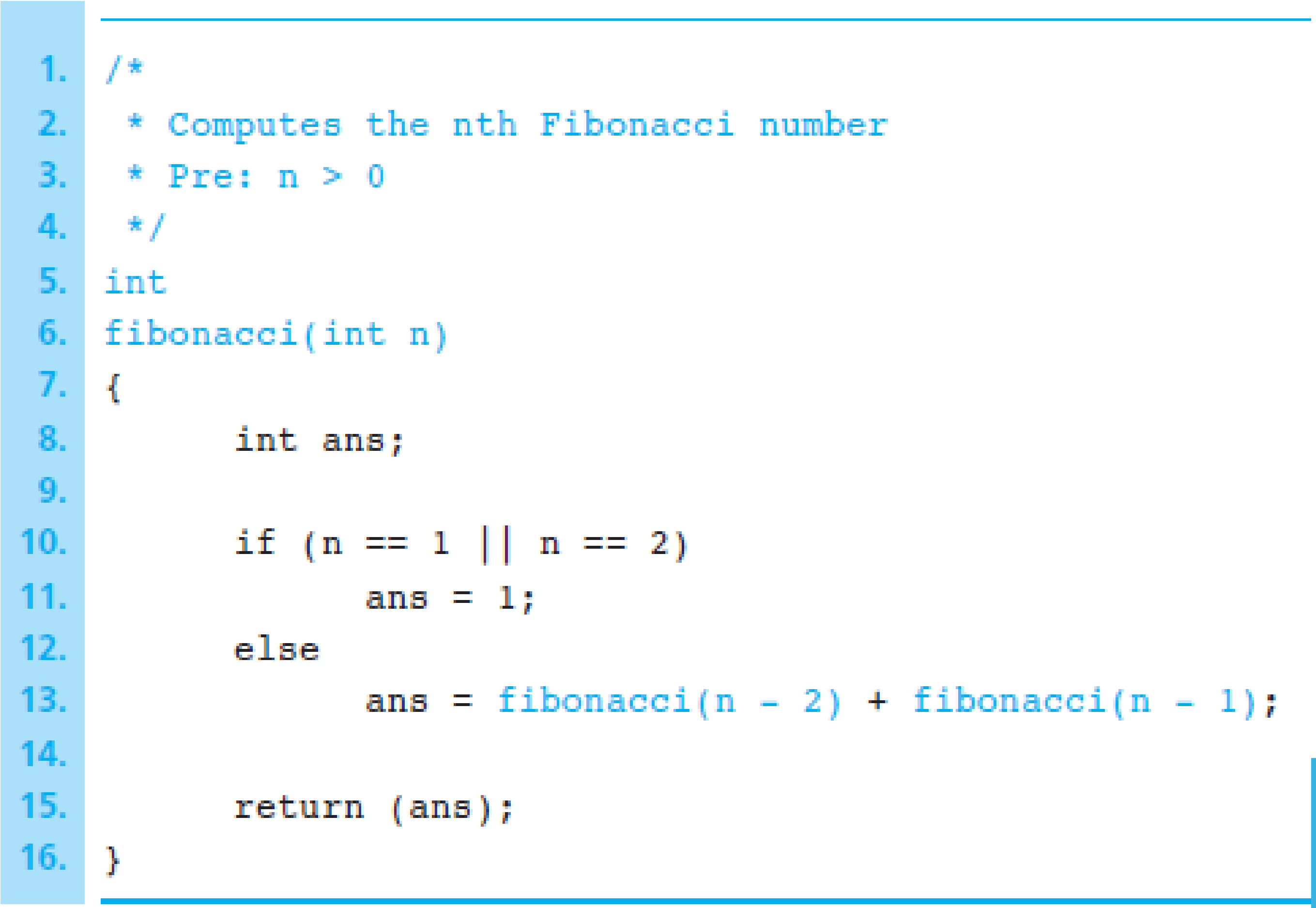


FIGURE 9.12 ITERATIVE FUNCTION FACTORIAL



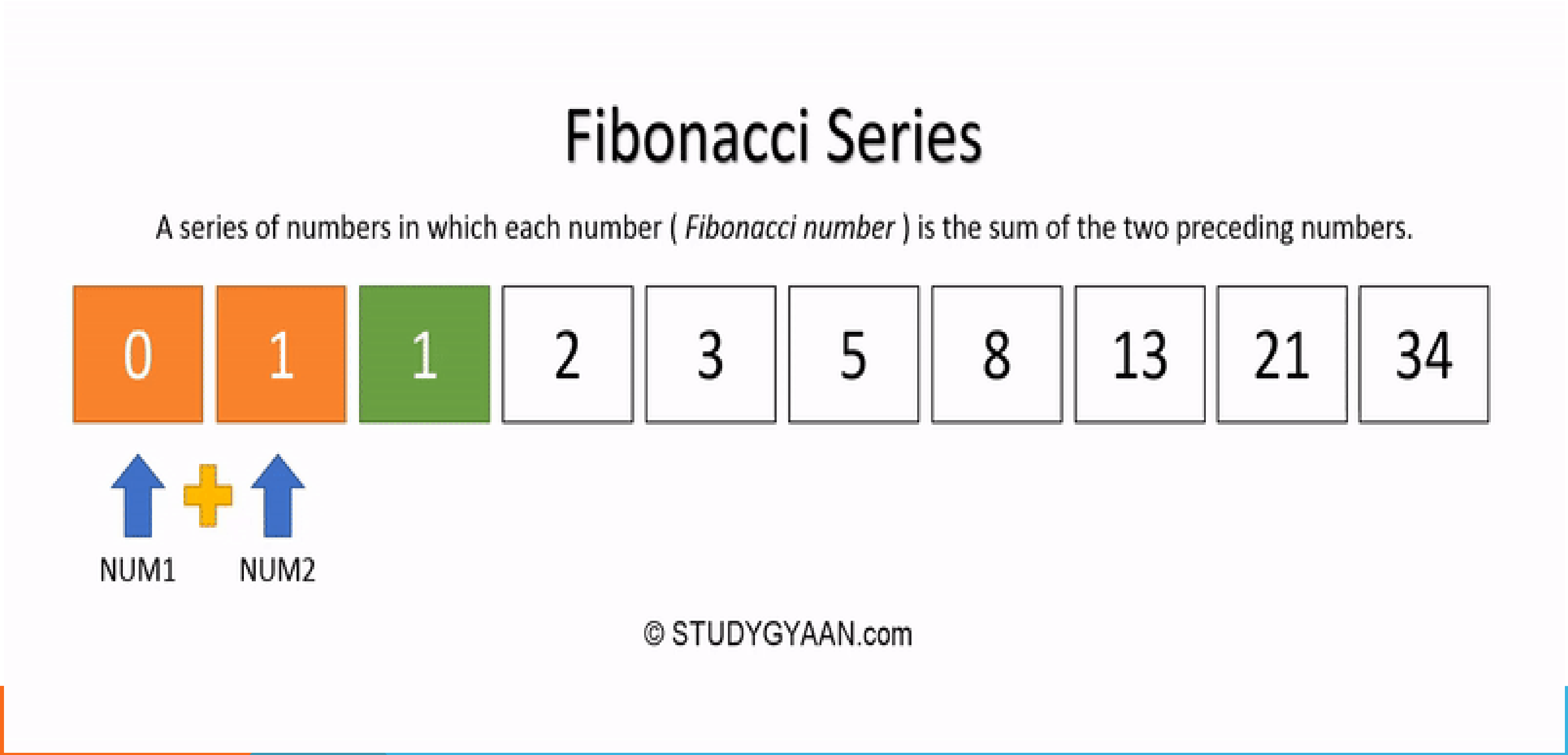
```
/*
 * Computes n!
 * Pre: n is greater than or equal to zero
 * /
int
factorial(int n)
{
    int i,                  /* local variables */
        product = 1;

    /* Compute the product n x (n-1) x (n-2) x . . . x 2 x 1 */
    for (i = n; i > 1; --i) {
       product = product * i;
    }

    /* Return function result */
    return (product);
}
```

FIGURE 9.13 RECURSIVE FUNCTION FIBONACCI



```c
/*
 * Computes the nth Fibonacci number
 * Pre: n > 0
 */
int
fibonacci(int n)
{
      int ans;

      if (n == 1 || n == 2)
            ans = 1;
      else
            ans = fibonacci(n - 2) + fibonacci(n - 1);

      return (ans);
}
```

RECURSIVE FUNCTION FIBONACCI



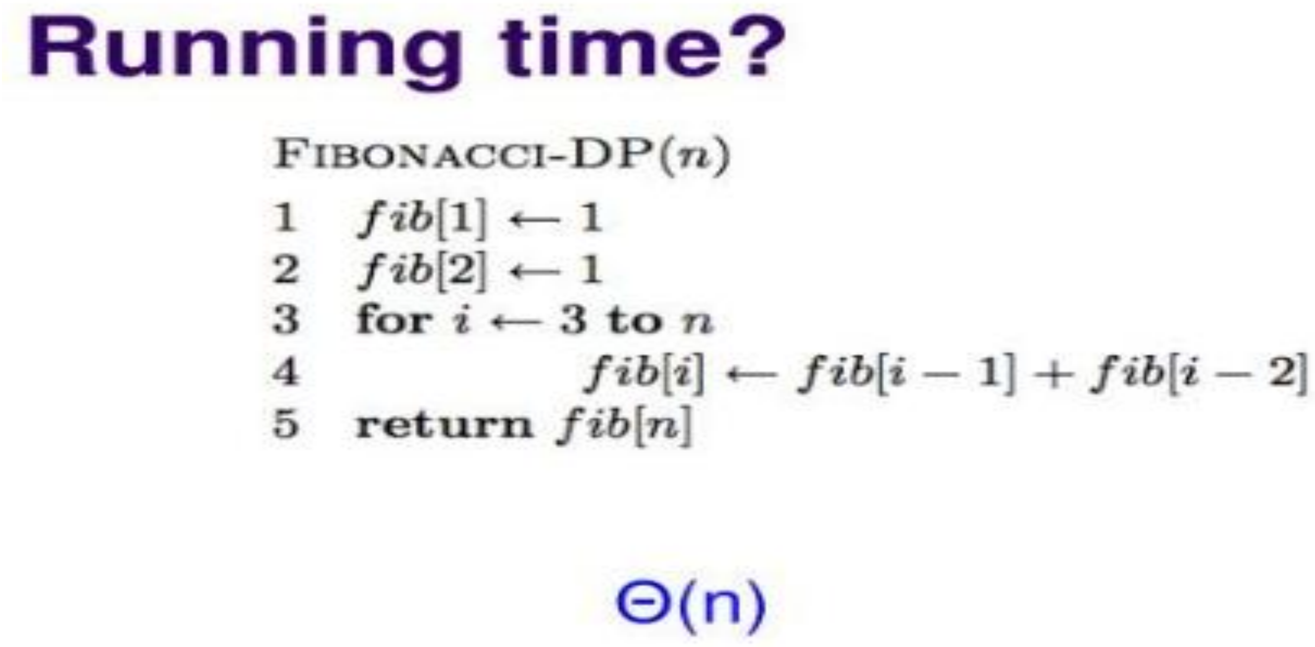
Fibonacci Series

A series of numbers in which each number (*Fibonacci number*) is the sum of the two preceding numbers.

0	1	1	2	3	5	8	13	21	34

© STUDYGYAAN.com

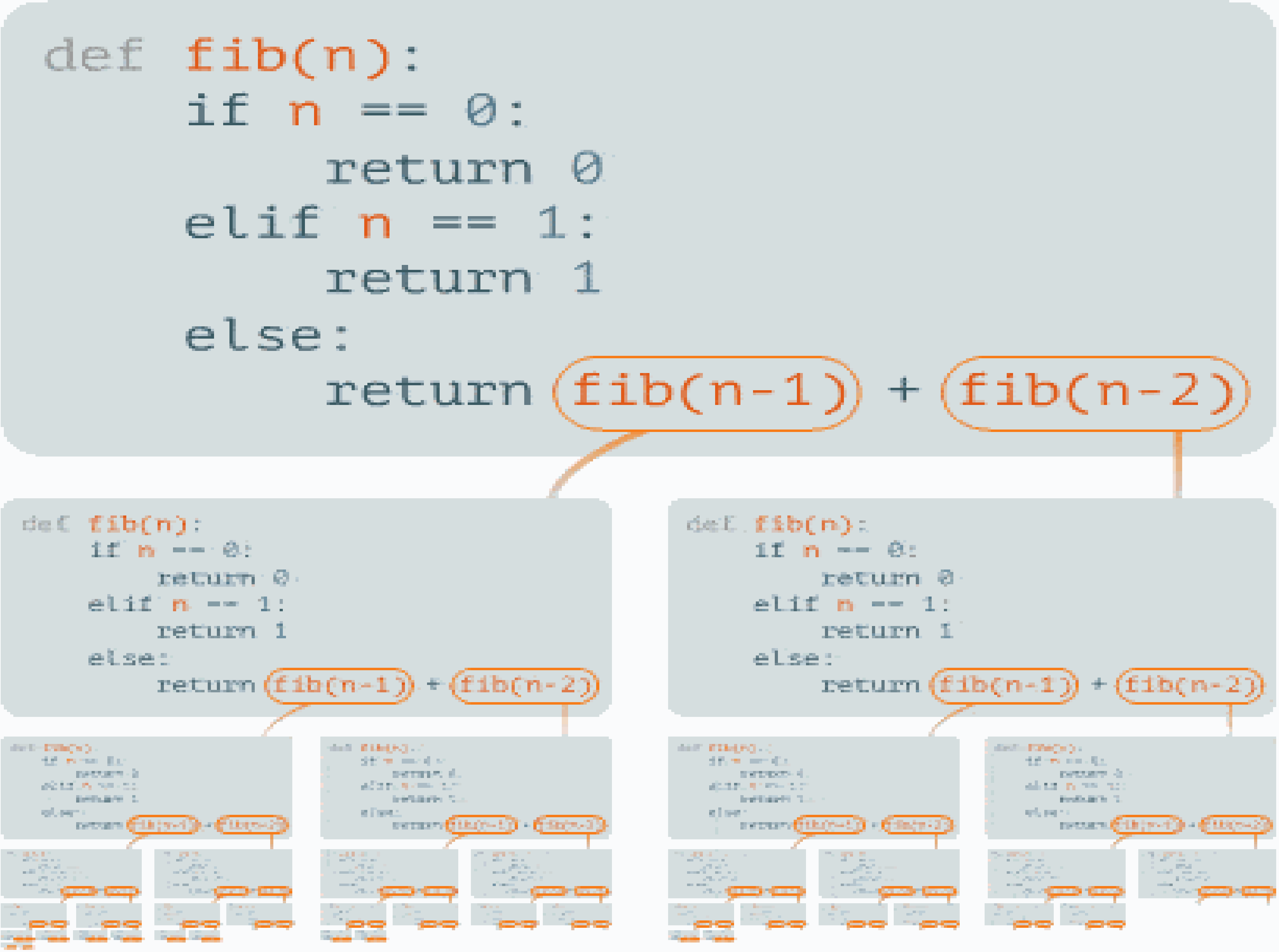
FIBONACCI FOR DYNAMIC PROGRAMMING



Running time?

```
FIBONACCI-DP($n$)
$fib[1] \leftarrow 1$
$fib[2] \leftarrow 1$
for $i \leftarrow 3$ to $n$
        $fib[i] \leftarrow fib[i-1] + fib[i-2]$
return $fib[n]$
```

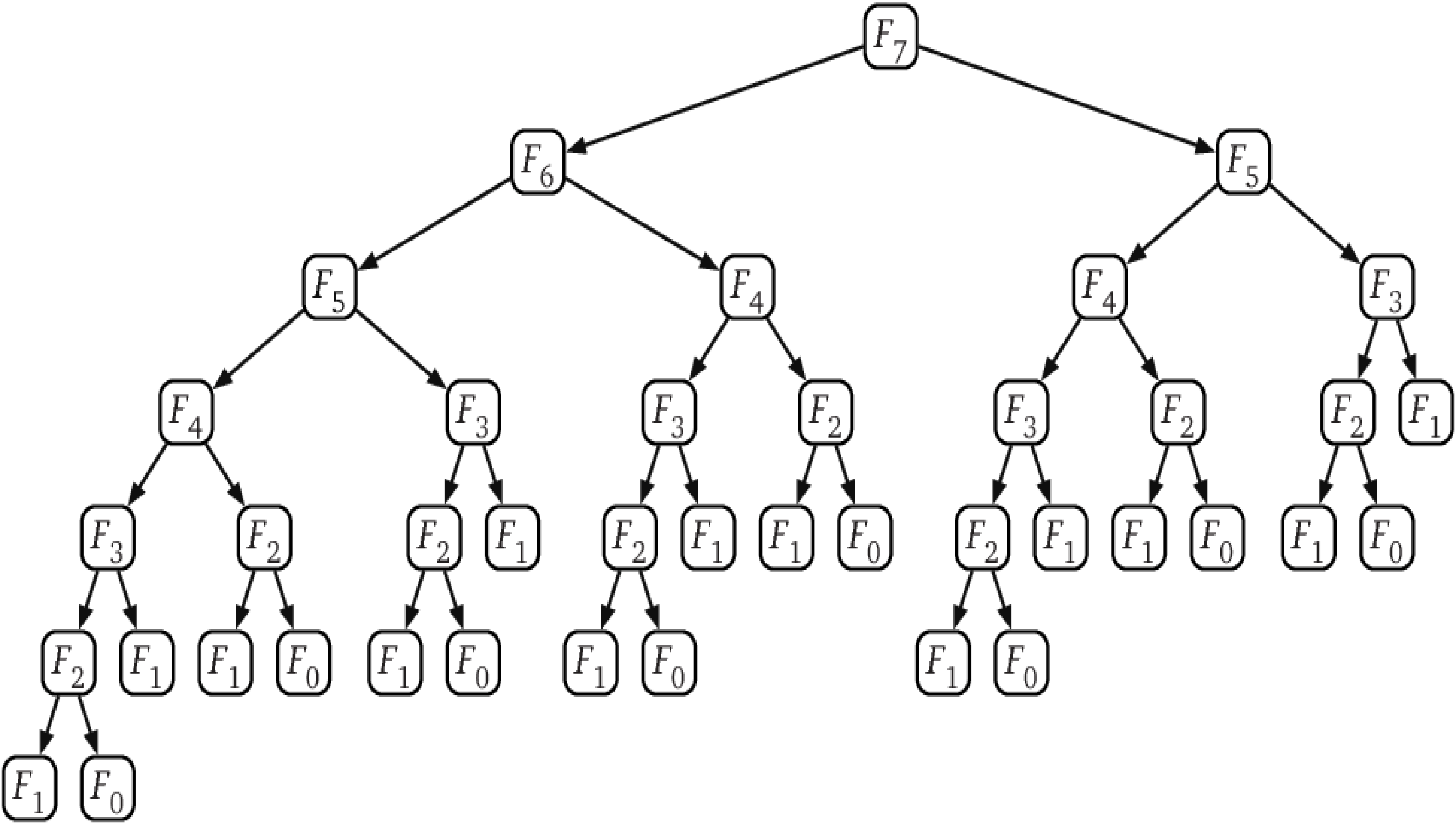
$\Theta(n)$



```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```


F_7
F_6
F_5
F_5
F_4
F_4
F_3
F_4
F_3
F_3
F_2
F_3
F_2
F_2
F_1
F_3
F_2
F_2
F_1
F_2
F_1
F_1
F_0
F_2
F_1
F_1
F_0
F_1
F_0
F_2
F_1
F_1
F_0
F_1
F_0
F_1
F_0
F_1
F_0
F_1
F_0