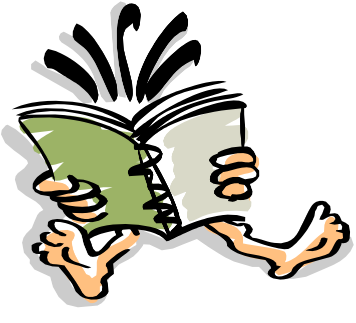
CS1101
Discrete Structures 1

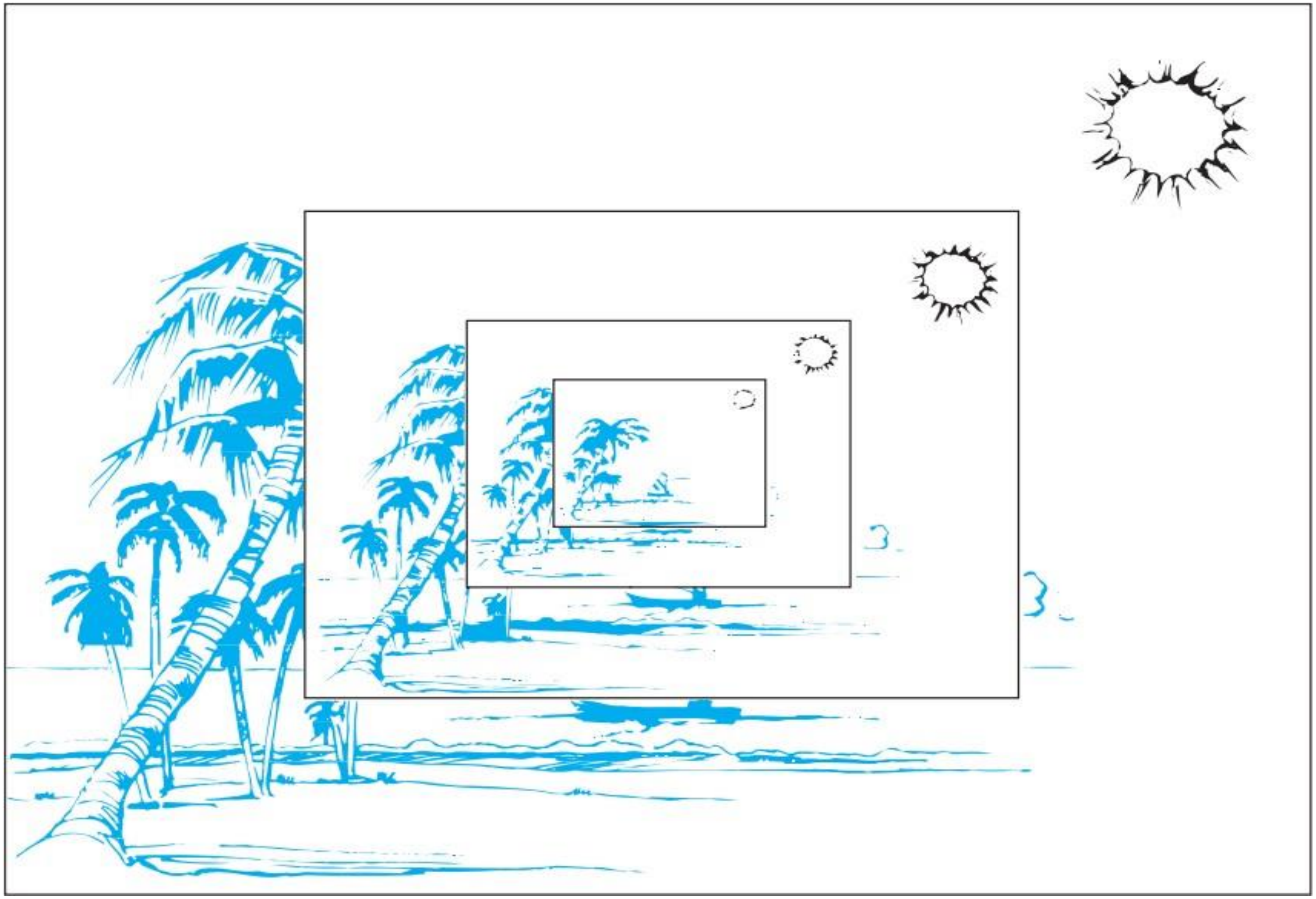
Chapter 05
Induction and Recursion



Chapter 5: Induction and Recursion

- Mathematical Induction.
- Strong Induction.
- Recursive Definitions.
- Recursive Algorithms.

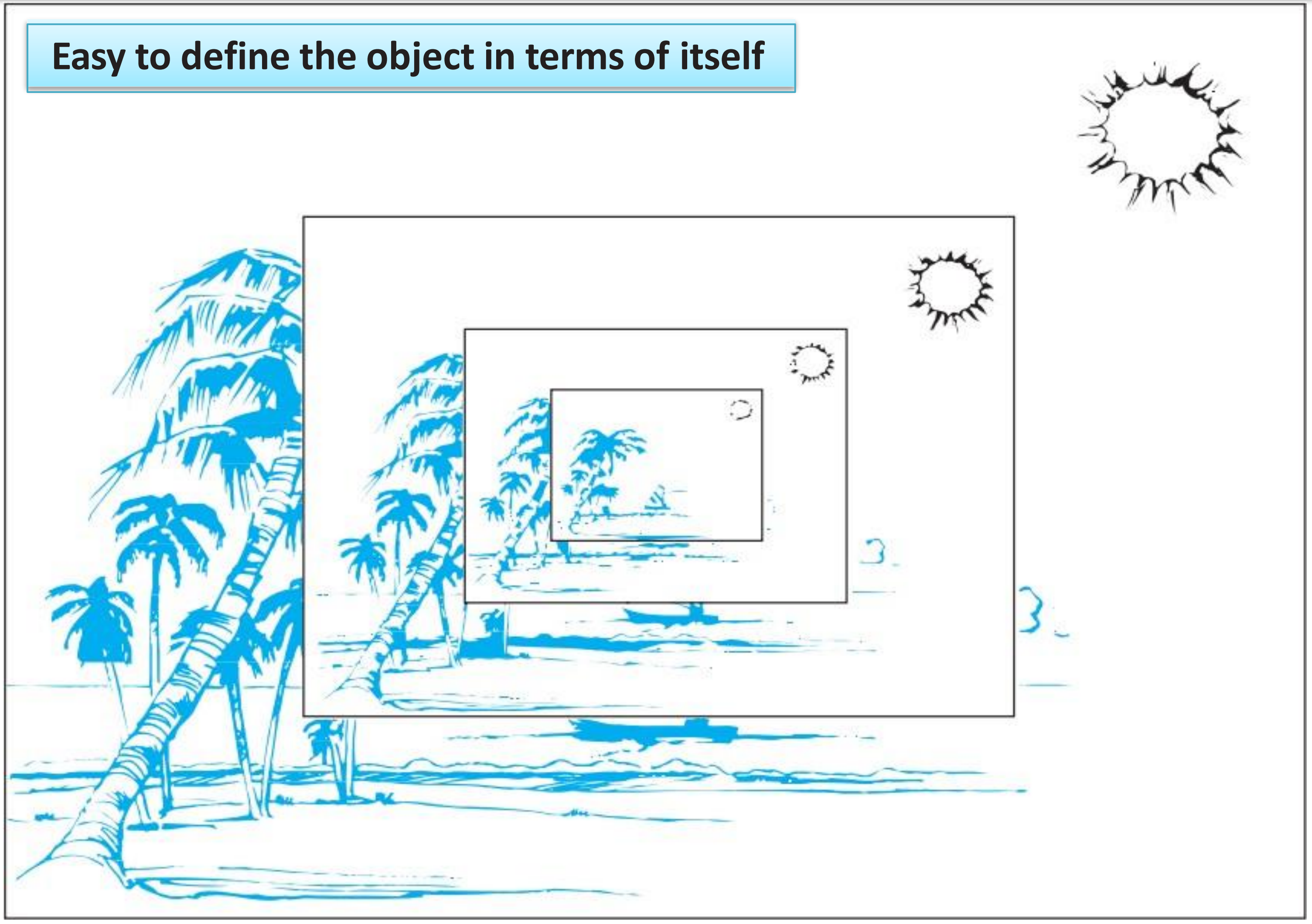
Recursive Definitions (1/13)



Recursive Definitions (1/13)

Recursion

Easy to define the object in terms of itself



Recursive Definitions (2/13)

Recursion:

The process of defining an object in terms of itself.

Recursively Defined Functions:

Basis Step

Specify the value of the function at the first point.

Recursive Step

Specifying how terms in the function are found from previous terms.

Example 1:

We use two steps to define a function with the set of *nonnegative integers* as its domain:

1) <u>Basis Step:</u>

Specify the value of the function at *zero*.

$f(0) = 0$

2) <u>Recursive Step:</u>

Give a rule for finding its value at an integer from its values at smaller integers.

$f(n+1) = f(n) + 1,$ for integer $n \geq 0$ (i.e., nonnegative integers)

Recursive Definitions (4/13)

Example 2:

The sequence of powers of 2 is given by $a_n = 2^n$ for $n = 0, 1, 2, \ldots$

Example 2:

Explicit Formula

The sequence of powers of 2 is given by $\boxed{a_n = 2^n}$ for $n = 0, 1, 2, \dots$

Recursive Definitions (4/13)

Example 2 – Answer:

Explicit Formula

- The sequence of powers of 2 is given by $\boxed{a_n = 2^n}$ for $n = 0, 1, 2, \ldots$
- **1) Basis Step:**
- Specify the value of the sequence at *zero*.
- $a_0 = 2^0 = 1$

2) Recursive Step:

Give a rule for finding a term of the sequence from the previous one.

$a_{n+1} = 2a_n$, for $n = 0, 1, 2, \ldots$

Example 2 – Answer:

Explicit Formula

- The sequence of powers of 2 is given by $\boxed{a_n = 2^n}$ for $n = 0, 1, 2, \ldots$
- **<u>1) Basis Step:</u>**
- Specify the value of the sequence at *zero*.
- $a_0 = 2^0 = 1$

<u>2) Recursive Step:</u>

Give a rule for finding a term of the sequence from the previous one.

$\boxed{a_{n+1} = 2a_n}$, for $n = 0, 1, 2, \ldots$

Recursive Formula

Example 3:

Suppose that f is defined recursively by

$$f(0) = 3,$$
$$f(n+1) = 2f(n) + 3.$$

Find $f(1)$, $f(2)$, $f(3)$, and $f(4)$.

Recursive Definitions (5/13)

Example 3 – Answer:

Suppose that f is defined recursively by

$$f(0) = 3,$$

$$f(n+1) = 2f(n) + 3.$$

Find $f(1)$, $f(2)$, $f(3)$, and $f(4)$.

Solution: From the recursive definition it follows that

$$f(1) = 2f(0) + 3 = 2 \cdot 3 + 3 = 9,$$

$$f(2) = 2f(1) + 3 = 2 \cdot 9 + 3 = 21,$$

$$f(3) = 2f(2) + 3 = 2 \cdot 21 + 3 = 45,$$

$$f(4) = 2f(3) + 3 = 2 \cdot 45 + 3 = 93.$$

Example 4:

Give a recursive definition of the factorial function $n!$

Recursive Definitions (6/13)

Example 4 – Answer:

Give a recursive definition of the factorial function $n!$

1) **<u>Basis Step:</u>**

Specify the value of the function at *zero*.

$f(0) = 1$

2) **<u>Recursive Step:</u>**

Give a rule for finding its value at an integer from its values at smaller integers.

$f(n+1) = (n+1) \cdot f(n)$, for $n = 0, 1, 2, \ldots$

Recursive Definitions (7/13)

Example 5:

Recall from Chapter 2 that the Fibonacci numbers, $f_0, f_1, f_2, \ldots$, are defined by the equations $f_0 = 0$, $f_1 = 1$, and

$$f_n = f_{n-1} + f_{n-2}$$

Find:

f_2

f_3

f_4

f_5

Example 5 – Answer:

Recall from Chapter 2 that the Fibonacci numbers, $f_0, f_1, f_2, \ldots$, are defined by the equations $f_0 = 0$, $f_1 = 1$, and

$$f_n = f_{n-1} + f_{n-2}$$

<u>Find:</u>

$$f_2 = f_1 + f_0 = 1 + 0 = 1$$

$$f_3 = f_2 + f_1 = 1 + 1 = 2$$

$$f_4 = f_3 + f_2 = 2 + 1 = 3$$

$$f_5 = f_4 + f_3 = 3 + 2 = 5$$

Recursive Definitions (8/13)

Example 6:

Give a recursive definition of

$$\sum_{k=0}^{n} a_k.$$

Recursive Definitions (8/13)

Example 6 – Answer :

Solution: The first part of the recursive definition is

$$\sum_{k=0}^{0} a_k = a_0.$$

The second part is

$$\sum_{k=0}^{n+1} a_k = \left(\sum_{k=0}^{n} a_k\right) + a_{n+1}.$$

Definition:

Recursive definitions play an important role in the study of strings. (More information: *The theory of formal languages*).

The set Σ^* of strings over the alphabet Σ is defined recursively by

1) Basis Step:

$\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

2) Recursive Step:

If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Recursive Definitions (10/13)

Example 7:

If $\Sigma = \{0, 1\}$, the strings found to be in Σ^*, the set of all bit strings, are λ, specified to be in Σ^*in the basis step, 0 and 1 formed during the first application of the recursive step, 00, 01, 10, and 11 formed during the second application of the recursive step, and so on.

$$\lambda, 0, 1, 00, 01, 10, 11, \ldots$$

Example 8:

If $\Sigma = \{a, b\}$, show that aab is in Σ^*.

Example 8 – Answer:

If $\Sigma = \{a, b\}$, show that aab is in Σ^*.

<u>1) Basis Step:</u>

$\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

<u>2) Recursive Step:</u>

If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Since $\lambda \in \Sigma^*$ and $a \in \Sigma$ then $\lambda a \in \Sigma^*$ (i.e., $a \in \Sigma^*$)

Example 8 – Answer:

If $\Sigma = \{a, b\}$, show that aab is in Σ^*.

1) Basis Step:

$\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

2) Recursive Step:

If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Since $\lambda \in \Sigma^*$ and $a \in \Sigma$ then $\lambda a \in \Sigma^*$ (i.e., $a \in \Sigma^*$)

Since $a \in \Sigma^*$ and $a \in \Sigma$ then $aa \in \Sigma^*$

Example 8 – Answer:

If $\Sigma = \{a, b\}$, show that aab is in Σ^*.

1) Basis Step:

$\lambda \in \Sigma^*$ (where λ is the empty string containing no symbols).

2) Recursive Step:

If $w \in \Sigma^*$ and $x \in \Sigma$, then $wx \in \Sigma^*$.

Since $\lambda \in \Sigma^*$ and $a \in \Sigma$ then $\lambda a \in \Sigma^*$ (i.e., $a \in \Sigma^*$)

Since $a \in \Sigma^*$ and $a \in \Sigma$ then $aa \in \Sigma^*$

Since $aa \in \Sigma^*$ and $b \in \Sigma$ then $aab \in \Sigma^*$

Recursive Definitions (12/13)

Definition:

Two strings can be combined via the operation of concatenation. Let Σ be a set of symbols and Σ^* the set of strings formed from symbols in Σ. We can define the concatenation of two strings, denoted by $\cdot$, recursively as follows.

1) <u>Basis Step:</u>

If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

2) <u>Recursive Step:</u>

If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

Definition:

1) <u>Basis Step:</u>

If $w \in \Sigma^*$, then $w \cdot \lambda = w$, where λ is the empty string.

2) <u>Recursive Step:</u>

If $w_1 \in \Sigma^*$ and $w_2 \in \Sigma^*$ and $x \in \Sigma$, then $w_1 \cdot (w_2 x) = (w_1 \cdot w_2)x$.

The concatenation of the strings w_1 and w_2 is often written as $w_1 w_2$ rather than $w_1 \cdot w_2$.

<u>Ex.</u> If $w_1 = discrete$ and $w_2 = mathematics$

Then $w_1 w_2 = discretemathematics$

Example 9:

Give a recursive definition of $l(w)$, the length of the string w.

Recursive Definitions (13/13)

Example 9 – Answer:

Give a recursive definition of $l(w)$, the length of the string w.

Solution: The length of a string can be recursively defined by

$$l(\lambda) = 0;$$
$$l(wx) = l(w) + 1 \text{ if } w \in \Sigma^* \text{ and } x \in \Sigma.$$

Recursive Algorithms (1/4)

Definition:

An algorithm is called ***recursive*** if it solves a problem by reducing it to an instance of the same problem with smaller input.

Example 1:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

$0! = 1$

$1! = 1$

$2! = 1 \cdot 2 = 2$

$3! = 1 \cdot 2 \cdot 3 = 6$

$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$

.

.

.

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

$0! = 1$
$1! = 1$
$2! = 1 \cdot 2 = 2$
$3! = 1 \cdot 2 \cdot 3 = 6$
$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$
.
.
.

$0! = 1$
$1! = (0!) \cdot 1$
$2! = (1!) \cdot 2 = 2$
$3! = (2!) \cdot 3 = 6$
$4! = (3!) \cdot 4 = 24$
.
.
.

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

$$0! = 1$$
$$1! = 1$$
$$2! = 1 \cdot 2 = 2$$
$$3! = 1 \cdot 2 \cdot 3 = 6$$
$$4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$

.
.
.

$$0! = 1$$
$$1! = (0!) \cdot 1$$
$$2! = (1!) \cdot 2 = 2$$
$$3! = (2!) \cdot 3 = 6$$
$$4! = (3!) \cdot 4 = 24$$

.
.
.

$$0! = 1$$
$$n! = (n-1)! \cdot n$$

Where $n = 1, 2, 3, \ldots$

Recursive Algorithms (2/4)

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

$0! = 1$

$4! = ?$

$0! = 1$

$n! = (n-1)! \cdot n$

Where $n = 1, 2, 3, \ldots$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

$0! = 1$

$4! = 3! \cdot 4$
$4! = 2! \cdot 3 \cdot 4$
$4! = 1! \cdot 2 \cdot 3 \cdot 4$
$4! = 0! \cdot 1 \cdot 2 \cdot 3 \cdot 4$
$4! = 1 \cdot 1 \cdot 2 \cdot 3 \cdot 4 = 24$

$0! = 1$

$n! = (n-1)! \cdot n$

Where $n = 1, 2, 3, \ldots$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

$$0! = 1$$

$$n! = (n-1)! \cdot n$$

Where $n = 1, 2, 3, \ldots$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
4	

Recursive Algorithms (2/4)

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
4	

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot \mathit{factorial}(n - 1)$
{output is $n!$}

n	return
4	$4 \cdot f(3)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
3	$4 \cdot f(3)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing *n*!.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
3	$4 \cdot f(3)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot$ *factorial*$(n - 1)$
{output is $n!$}

n	return
3	$4 \cdot 3 \cdot f(2)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
2	$4 \cdot 3 \cdot f(2)$

Recursive Algorithms (2/4)

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
2	$4 \cdot 3 \cdot f(2)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
2	$4 \cdot 3 \cdot 2 \cdot f(1)$

Recursive Algorithms (2/4)

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n-1)$
{output is $n!$}

n	return
1	$4 \cdot 3 \cdot 2 \cdot f(1)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
1	$4 \cdot 3 \cdot 2 \cdot f(1)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
1	$4 \cdot 3 \cdot 2 \cdot 1 \cdot f(0)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n-1)$
{output is $n!$}

n	return
0	$4 \cdot 3 \cdot 2 \cdot 1 \cdot f(0)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing *n*!.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n-1)$
{output is $n!$}

n	return
0	$4 \cdot 3 \cdot 2 \cdot 1 \cdot f(0)$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
0	$4 \cdot 3 \cdot 2 \cdot 1 \cdot 1$

Example 1 – Answer:

Give a recursive algorithm for computing $n!$, where n is a nonnegative integer.

A Recursive Algorithm for Computing $n!$.

procedure *factorial*(n: nonnegative integer)
if $n = 0$ **then return** 1
else return $n \cdot factorial(n - 1)$
{output is $n!$}

n	return
0	$4 \cdot 3 \cdot 2 \cdot 1 \cdot 1 = 24$

Example 2:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

$$a^0 = 1$$
$$a^1 = a$$
$$a^2 = a \cdot a$$
$$a^3 = a \cdot a \cdot a$$
$$a^4 = a \cdot a \cdot a \cdot a$$

.

.

.

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

$a^0 = 1$
$a^1 = a$
$a^2 = a \cdot a$
$a^3 = a \cdot a \cdot a$
$a^4 = a \cdot a \cdot a \cdot a$
.
.
.

$a^0 = 1$
$a^1 = a \cdot a^0$
$a^2 = a \cdot a^1$
$a^3 = a \cdot a^2$
$a^4 = a \cdot a^3$
.
.

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

$a^0 = 1$
$a^1 = a$
$a^2 = a \cdot a$
$a^3 = a \cdot a \cdot a$
$a^4 = a \cdot a \cdot a \cdot a$
.
.
.

$a^0 = 1$
$a^1 = a \cdot a^0$
$a^2 = a \cdot a^1$
$a^3 = a \cdot a^2$
$a^4 = a \cdot a^3$
.
.

$a^0 = 1$

$a^n = a \cdot a^{n-1}$

Where $n = 1, 2, 3, \ldots$

Recursive Algorithms (3/4)

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

$a^0 = 1$

$a^n = a \cdot a^{n-1}$

Where $n = 1, 2, 3, \ldots$

Recursive Algorithms (3/4)

a	n	return
2	3	

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	3	

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

Example 2 – Answer:

a	n	return
2	3	$2 \cdot p(2,2)$

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

Example 2 – Answer:

a	n	return
2	2	$2 \cdot p(2,2)$

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n-1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	2	$2 \cdot p(2,2)$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

Example 2 – Answer:

a	n	return
2	2	$2 \cdot 2 \cdot p(2,1)$

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	1	$2 \cdot 2 \cdot p\,(2, 1)$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	1	$2 \cdot 2 \cdot p(2,1)$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

Example 2 – Answer:

a	n	return
2	1	$2 \cdot 2 \cdot 2 \cdot p(2,0)$

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	0	$2 \cdot 2 \cdot 2 \cdot p(2,0)$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

Example 2 – Answer:

a	n	return
2	0	$2 \cdot 2 \cdot 2 \cdot p(2, 0)$

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	0	$2 \cdot 2 \cdot 2 \cdot 1$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

Recursive Algorithms (3/4)

a	n	return
2	0	$2 \cdot 2 \cdot 2 \cdot 1$

Example 2 – Answer:

Give a recursive algorithm for computing a^n, where a is a nonzero real number and n is a nonnegative integer.

A Recursive Algorithm for Computing a^n.

procedure *power*(a: nonzero real number, n: nonnegative integer)
if $n = 0$ **then return** 1
else return $a \cdot power(a, n - 1)$
{output is a^n}

$2^3 = 2 \cdot 2 \cdot 2 \cdot 1 = 8$

Example 3:

Construct a recursive version of a *binary search* algorithm.

Recursive Algorithms (4/4)

Example 3 – Answer:

A Recursive Binary Search Algorithm.

procedure *binary search*$(i, j, x$: integers, $1 \leq i \leq j \leq n)$
$m := \lfloor (i + j)/2 \rfloor$
if $x = a_m$ **then**
    return m
else if $(x < a_m$ and $i < m)$ **then**
    return *binary search*$(i, m - 1, x)$
else if $(x > a_m$ and $j > m)$ **then**
    return *binary search*$(m + 1, j, x)$
else return 0
{output is location of x in $a_1, a_2, \ldots, a_n$ if it appears; otherwise it is 0}