DS-4

Recursion

Suppose P is a procedure containing either a Call statement to itself or a Call statement to a second procedure that may eventually result in a Call statement back to the original procedure P. Then P is called a recursive procedure. So that the program will not continue to run indefinitely, a recursive procedure must have the following two properties:

1. There must be certain criteria, called base case criteria, for which the procedure does not call itself.

2. Each time the procedure does call itself(directly or indirectly), it must be closer to the base criteria.

A recursive procedure with these two properties is to said to be well-defined.

Similarly, a function is said to be recursively define if the function definition refers to itself. Again, in order for the definition not to be circular, it must have the following two properties:

(1) There must be certain arguments, called base values, for which the function does not refer to itself.

(2) Each time the function does refer to itself, the argument of the function must be closer to a base value

Factorial Function

The product of the positive integers from 1 to n, inclusive, is called "n factorial" and is usually denoted by $n!$:

$$n! = 1 \cdot 2 \cdot 3 \cdots (n-2)(n-1)n$$

It is also convenient to define $0! = 1$, so that the function is defined for all nonnegative integers. Thus we have

$$0! = 1 \qquad 1! = 1 \qquad 2! = 1 \cdot 2 = 2 \qquad 3! = 1 \cdot 2 \cdot 3 = 6 \qquad 4! = 1 \cdot 2 \cdot 3 \cdot 4 = 24$$
$$5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120 \qquad 6! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 = 720$$

and so on. Observe that

$$5! = 5 \cdot 4! = 5 \cdot 24 = 120 \quad \text{and} \quad 6! = 6 \cdot 5! = 6 \cdot 120 = 720$$

This is true for every positive integer n; that is,

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

Accordingly, the factorial function may also be defined as follows:

Definition 6.1: (Factorial Function)

(*a*) If $n = 0$, then $n! = 1$.
(*b*) If $n > 0$, then $n! = n \cdot (n-1)!$

Recursive procedure for calculating factorial

This procedure calculates N! and returns the value:

FACTORIAL(N)

1. If N=0, then: return 1.
2. Return N*FACTORIAL(N-1)

Fibonacci Sequence

The celebrated Fibonacci sequence (usually denoted by $F_0, F_1, F_2, \ldots$) is as follows:

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

That is, $F_0 = 0$ and $F_1 = 1$ and each succeeding term is the sum of the two preceding terms. For example, the next two terms of the sequence are

$$34 + 55 = 89 \quad \text{and} \quad 55 + 89 = 144$$

A formal definition of this function follows:

Definition 6.2: (Fibonacci Sequence)

(*a*) If $n = 0$ or $n = 1$, then $F_n = n$.

(*b*) If $n > 1$, then $F_n = F_{n-2} + F_{n-1}$

The procedure for finding the nth term Fn of the Fibonacci sequence follows:

FIBONACCI(N)

1. If N=0 or N=1, then return N
2. Return FIBONACCI(N-2)+FIBONACCI(N-1)

6.7 TOWERS OF HANOI

The preceding section gave examples of some recursive definitions and procedures. This section shows how recursion may be used as a tool in developing an algorithm to solve a particular problem. The problem we pick is known as the Towers of Hanoi problem.

Suppose three pegs, labeled A, B and C, are given, and suppose on peg A there are placed a finite number n of disks with decreasing size. This is pictured in Fig. 6-10 for the case $n = 6$. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary. The rules of the game are as follows:

(*a*) Only one disk may be moved at a time. Specifically, only the top disk on any peg may be moved to any other peg.

(*b*) At no time can a larger disk be placed on a smaller disk.

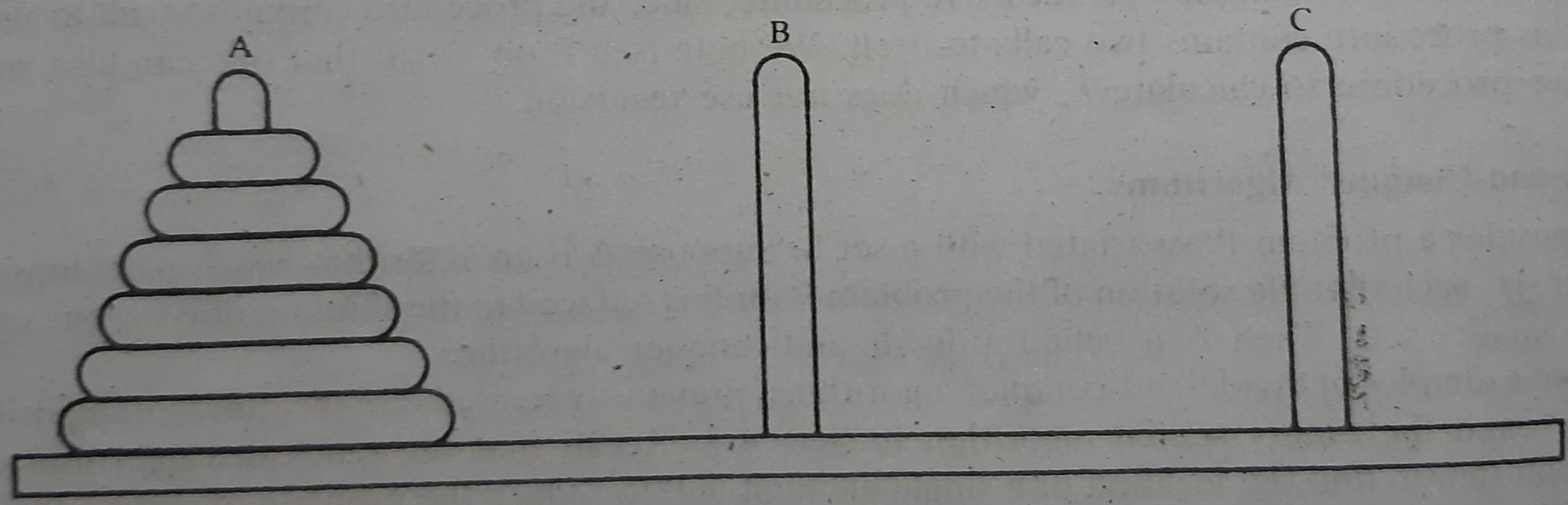
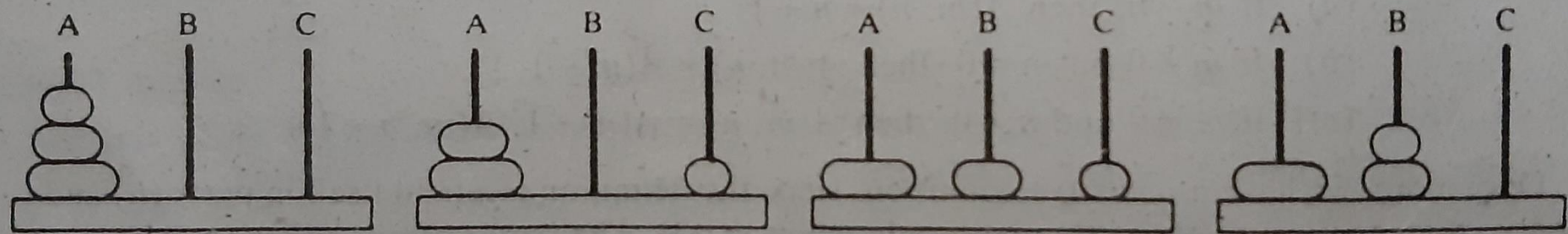


Fig. 6-10 Initial setup of Towers of Hanoi with $n = 6$.

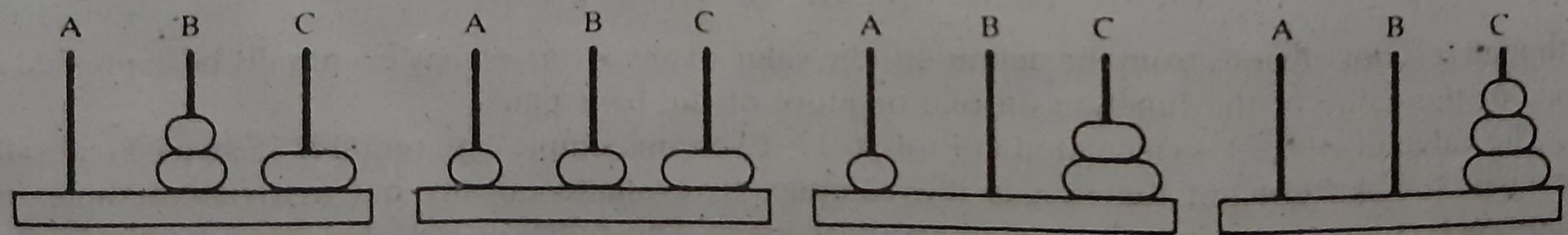
Sometimes we will write X → Y to denote the instruction "Move top disk from peg X to peg Y," where X and Y may be any of the three pegs.

The solution to the Towers of Hanoi problem for $n = 3$ appears in Fig. 6-11. Observe that it consists of the following seven moves:

$n = 3$: Move top disk from peg A to peg C.
Move top disk from peg A to peg B.
Move top disk from peg C to peg B.
Move top disk from peg A to peg C.
Move top disk from peg B to peg A.
Move top disk from peg B to peg C.
Move top disk from peg A to peg C.



(a) Initial. (1) A → C. (2) A → B. (3) C → B.



(4) A → C. (5) B → A. (6) B → C. (7) A → C.

In other words,

$n = 3$: $A \to C$, $A \to B$, $C \to B$, $A \to C$, $B \to A$, $B \to C$, $A \to C$

For completeness, we also give the solution to the Towers of Hanoi problem for $n = 1$ and $n = 2$:

$n = 1$: $A \to C$

$n = 2$: $A \to B$, $A \to C$, $B \to C$

Note that $n = 1$ uses only one move and that $n = 2$ uses three moves.

Rather than finding a separate solution for each n, we use the technique of recursion to develop a general solution. First we observe that the solution to the Towers of Hanoi problem for $n > 1$ disks may be reduced to the following subproblems:

(1) Move the top $n - 1$ disks from peg A to peg B.

(2) Move the top disk from peg A to peg C: $A \to C$.

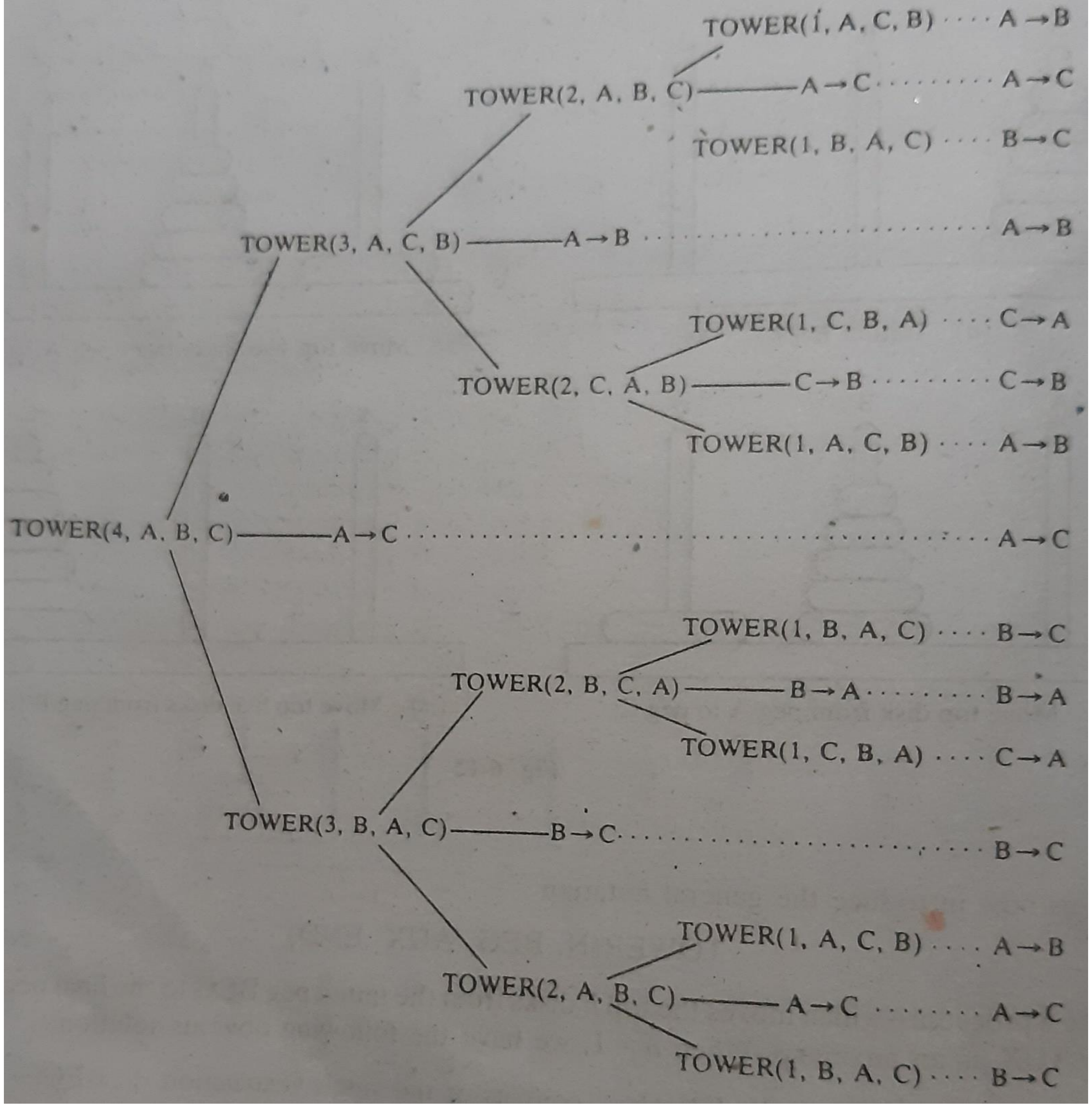
(3) Move the top $n - 1$ disks from peg B to peg C.

TOWER(4, A, B, C)

Observe that the recursive solution for $n = 4$ disks consists of the following 15 moves:

$A \to B$	$A \to C$	$B \to C$	$A \to B$	$C \to A$	$C \to B$	$A \to B$	$A \to C$
$B \to C$	$B \to A$	$C \to A$	$B \to C$	$A \to B$	$A \to C$	$B \to C$	

In general, this recursive solution requires $f(n) = 2^n - 1$ moves for n disks.



TOWER(1, A, C, B) · · · · A → B
TOWER(2, A, B, C) —— A → C · · · · · · · · · A → C
TOWER(1, B, A, C) · · · · B → C
TOWER(3, A, C, B) —— A → B · · · · · · · · · A → B
TOWER(1, C, B, A) · · · · C → A
TOWER(2, C, A, B) —— C → B · · · · · · · · · C → B
TOWER(1, A, C, B) · · · · A → B
TOWER(4, A, B, C) —— A → C · · · · · · · · · A → C
TOWER(1, B, A, C) · · · · B → C
TOWER(2, B, C, A) —— B → A · · · · · · · · · B → A
TOWER(1, C, B, A) · · · · C → A
TOWER(3, B, A, C) —— B → C · · · · · · · · · B → C
TOWER(1, A, C, B) · · · · A → B
TOWER(2, A, B, C) —— A → C · · · · · · · · · A → C
TOWER(1, B, A, C) · · · · B → C

Recursive algorithm for Tower of Hanoi Problem

TOWER(N, BEG, AUX, END)
This procedure gives a recursive solution to the Towers of Hanoi problem for N disks.

1. If N = 1, then:
   (*a*) Write: BEG → END.
   (*b*) Return.
   [End of If structure.]
2. [Move N − 1 disks from peg BEG to peg AUX.]
   Call TOWER(N − 1, BEG, END, AUX).
3. Write: BEG → END.
4. [Move N − 1 disks from peg AUX to peg END.]
   Call TOWER(N − 1, AUX, BEG, END).
5. Return.