Recursion

Instructors:
Md Nazrul Islam Mondal
Department of Computer Science & Engineering
Rajshahi University of Engineering & Technology Rajshahi-6204

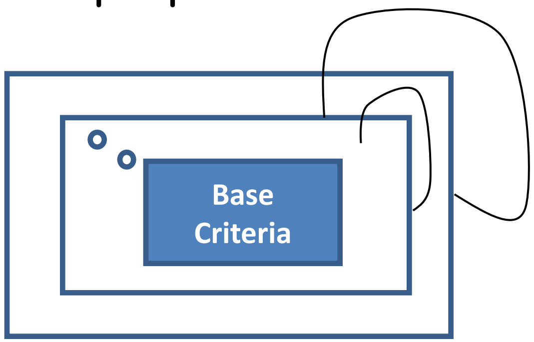
Outline

- Recursion
  - Definition
  - Factorial Function and Procedure
  - Fibonacci Sequence
  - Divide-and-conquer Algorithm (definition and example)
  - Ackermann Function and Procedure
  - Tower of Hanoi

Recursion

Recursion

- **Recursive Procedure –**
  - A procedure contains 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.
  - It must have to properties –
    - There must be certain criteria, call **base criteria**, for which the procedure does not call itself.
    - 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 said to be **well-defined**.



Recursion

- **Recursive function –**
  - The function definition refers to itself and have the following two properties
    - There must be certain **argument**, call **base values**, for which the function does not refer itself.
    - Each time the function does refer to **itself**, the argument of the function must be **closer** to the **base value**.
  - A recursive function with these two properties is also said to be **well-defined**

Recursion Function (Example: Factorial function)

- **Definition**:
  - If n = 0, then n! = 1
  - If n>0, then n! = n.(n-1)!

- **Example**:

(1) $4! = 4 \cdot 3!$
(2) $3! = 3 \cdot 2!$
(3) $2! = 2 \cdot 1!$
(4) $1! = 1 \cdot 0!$
(5) $0! = 1$
(6) $1! = 1 \cdot 1 = 1$
(7) $2! = 2 \cdot 1 = 2$
(8) $3! = 3 \cdot 2 = 6$
(9) $4! = 4 \cdot 6 = 24$

Recursion Using STACK

- **Example:**
- 4! = 4 x 3 x 2 x 1

Postfix: 4 3 2 1 x x x

Left to Right Scan (postfix)

(1) PUSH 4
(2) PUSH 3
(3) PUSH 2
(4) PUSH 1
(5) POP ....

STACK

1
2
3
4

Recursion Procedure (Example: Factorial Procedure)

Procedure 6.9B: FACTORIAL(FACT, N)

This procedure calculates N! and returns the value in the variable FACT.

1. If N = 0, then: Set FACT := 1, and Return.
2. Call FACTORIAL(FACT, N – 1).
3. Set FACT := N*FACT.
4. Return.

- FACTORIAL(FACT, 3)
  - FACTORIAL(FACT, 2)
    - FACTORIAL(FACT, 1)
      - FACTORIAL(FACT, 0)
    - FACT := 1 (POP)
  - FACT := 1*2 = 2 (POP)
- FACT:= 2*3 = 6 (POP)

STACK

FACTORIAL(FACT,1)
FACTORIAL(FACT,2)
FACTORIAL(FACT,3)
MAIN()

Recursion Function (Example: Fibonacci Sequence)

- **Definition**:
  - If $n = 0$ or $n = 1$, then $F_n = n$
  - If $n>0$, then $F_n = F_{n-1} + F_{n-2}$

- **Example**:
  - $F_3 = F_2 + F_1$
    - $F_1 = 1$
    - $F_2 = F_1 + F_0$
      - $F_1 = 1$
      - $F_0 = 0$
    - $F_2 = 1$
  - $F_3 = 1 + 1 = 2$

Recursion Procedure (Example: Fibonacci Procedure)

Procedure 6.10: FIBONACCI(FIB, N)

This procedure calculates F_N and returns the value in the first parameter FIB.

1. If N = 0 or N = 1, then: Set FIB := N, and Return.
2. Call FIBONACCI(FIBA, N – 2).
3. Call FIBONACCI(FIBB, N – 1).
4. Set FIB := FIBA + FIBB.
5. Return.

Recursion Procedure (Example: Fibonacci Procedure)

- FIBONACCI(FIB, 3)
  - FIBONACCI(FIBA, 1)
  - FIBA = 1, (POP)
  - FIBONACCI(FIBB, 2)
    - FIBONACCI(FIBA,0)
    - FIBA = 0 (POP)
    - FIBNACCI(FIBB,1)
    - FIB = 1 (POP)
    - FIB = 1+0 = 1
  - FIBB = 1
- FIB = 1+1 = 2

STACK

FIBONACCI(FIBA,1)
FIBONACCI(FIB,3)
MAIN()

STACK

FIBONACCI(FIBB,1)
FIBONACCI(FIBA,0)
FIBONACCI(FIBB,2)
FIBONACCI(FIB,3)
MAIN()

Recursion (Divide-and-Conquer Algorithm)

- Consider a problem associated with a set S.
- Definition:
  - Partitions S into smaller sets such that the solution of the problem P for S is reduced to the solution of P for one or more of the smaller sets.
- Example: Binary Search, Merge sort, Quick sort

Recursion
(Ackermann Function)

- Definition:
  - If $m = 0$, then $A(m, n) = n+1$
  - If $m \neq 0$ but $n = 0$, then $A(m, n) = A(m-1, 1)$
  - IF $m \neq 0$ and $n \neq 0$, then $A(m, n) = A(m-1, A(m, n-1))$

- The base criteria are the pairs

  $(0, 0), (0, 1), (0, 2), (0, 3) \ldots, (0, n) \ldots$

- $A(1,3) = ?$

Recursion
(Ackermann Function)

$$
\begin{array}{ll}
\textbf{(1)} & A(1,3) = A(0, A(1,2)) \\
\textbf{(2)} & \quad A(1,2) = A(0. A(1,1)) \\
\textbf{(3)} & \qquad A(1,1) = A(0. A(1,0)) \\
\textbf{(4)} & \qquad\quad A(1,0) = A(0,1) \\
\textbf{(5)} & \qquad\qquad A(0,1) = 1 + 1 = 2 \\
\textbf{(6)} & \qquad\quad A(1,0) = 2 \\
\textbf{(7)} & \qquad A(1,1) = A(0,2) \\
\textbf{(8)} & \qquad\quad A(0,2) = 2 + 1 = 3 \\
\textbf{(9)} & \qquad A(1,1) = 3 \\
\textbf{(10)} & \quad A(1,2) = A(0,3) \\
\textbf{(11)} & \qquad A(0,3) = 3 + 1 = 4 \\
\textbf{(12)} & \quad A(1.2) = 4 \\
\textbf{(13)} & A(1.3) = A(0.4) \\
\textbf{(14)} & \quad A(0.4) = 4 + 1 = 5 \\
\textbf{(15)} & A(1,3) = 5
\end{array}
$$

Recursion (Tower of Hanoi)

- The rules of the game are as follows:
  - Only one disk may be moved at a time (specially, only the top disk on any peg may be moved to any other peg)
  - At no time can a larger disk be paced on a smaller disk.

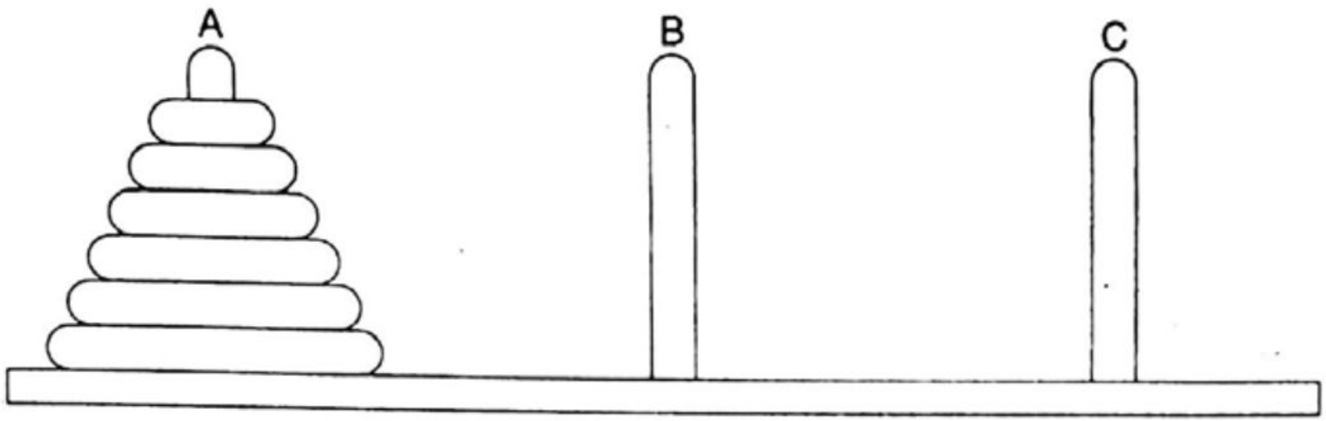


Fig. 6.14 *Initial Setup of Towers of Hanoi with $n = 6$*

Recursion
(Tower of Hanoi)

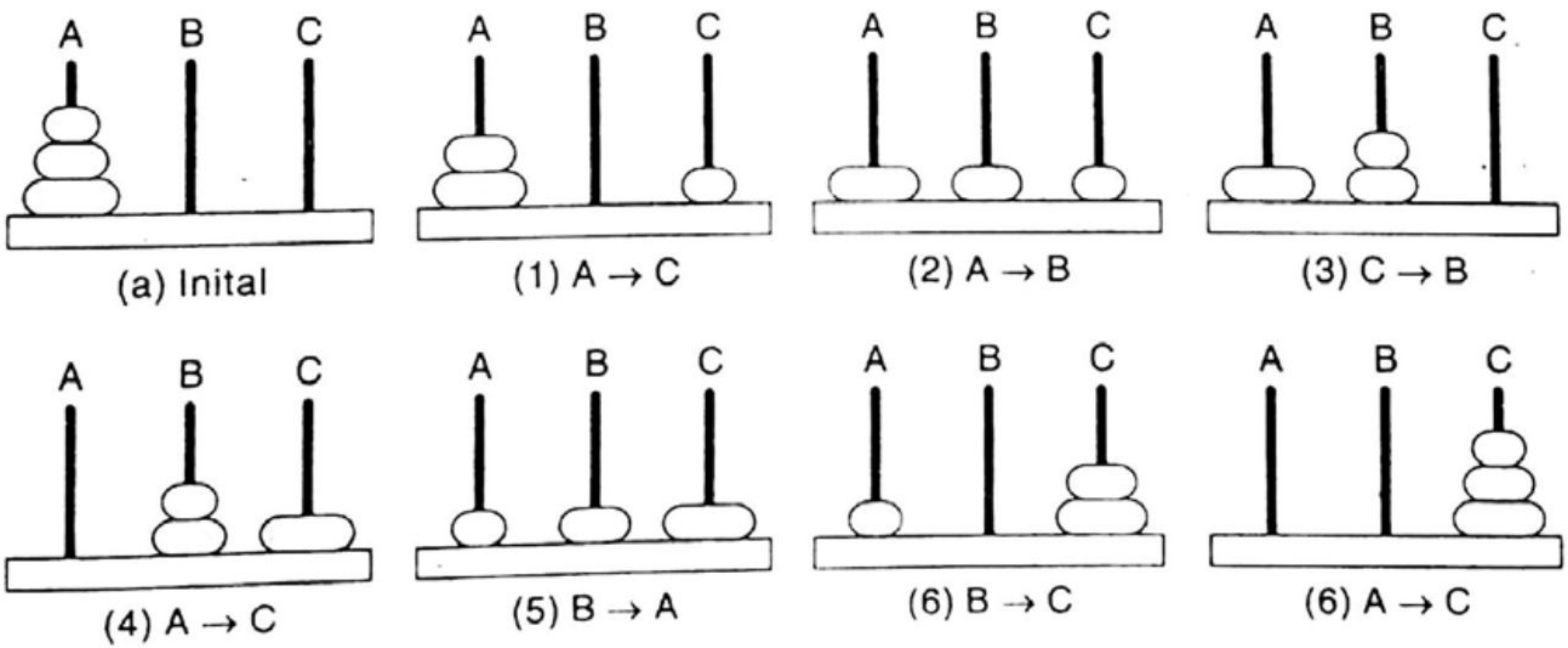


Fig. 6.15

Recursion
(Tower of Hanoi)

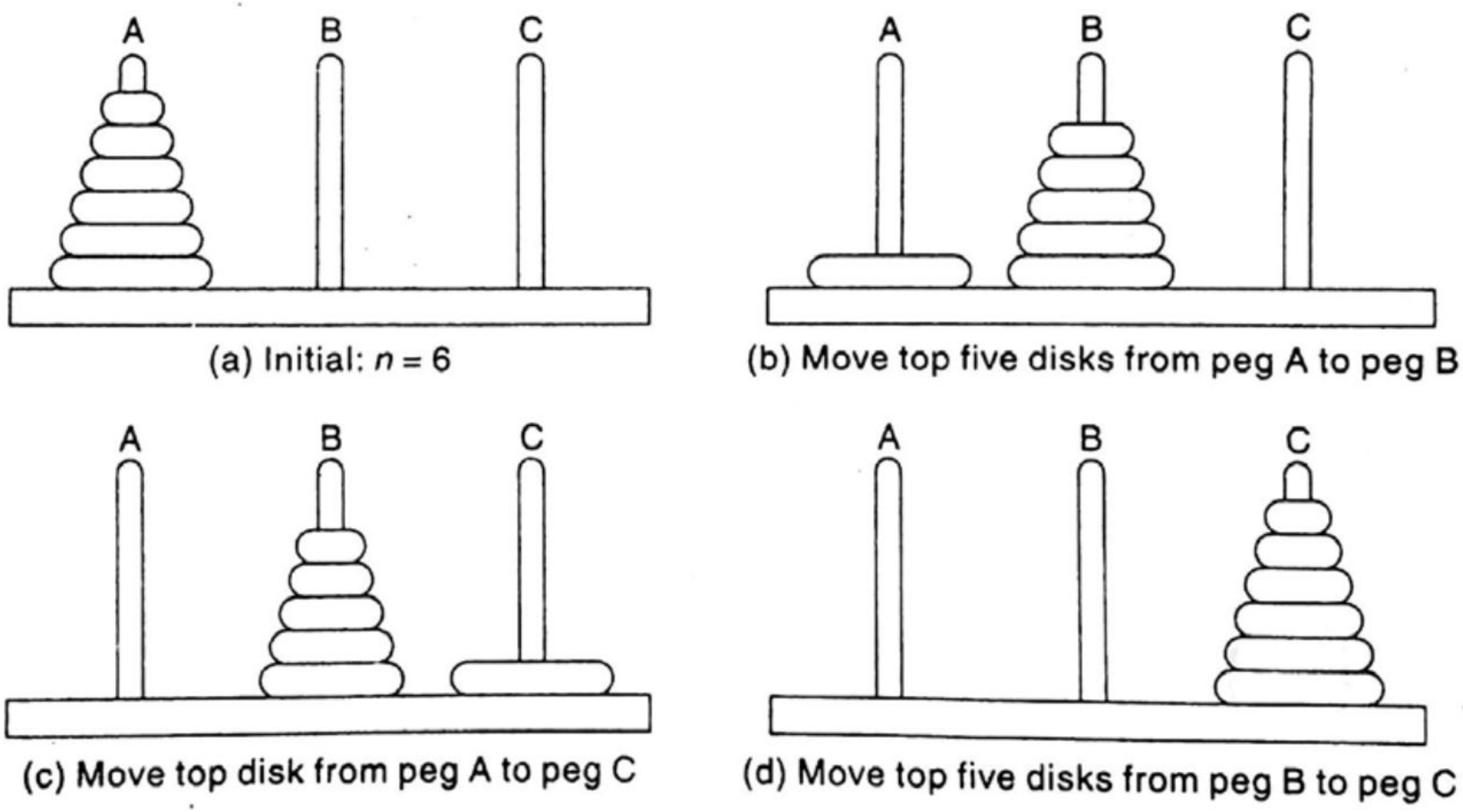
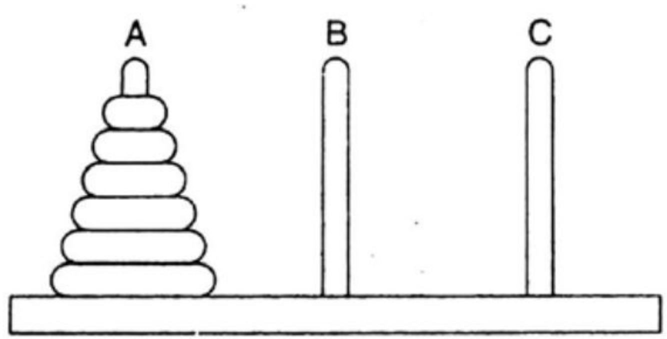


Fig. 6.16

Recursion
(Tower of Hanoi)

- General Notation:

  TOWER(N, BEG, AUX, END)

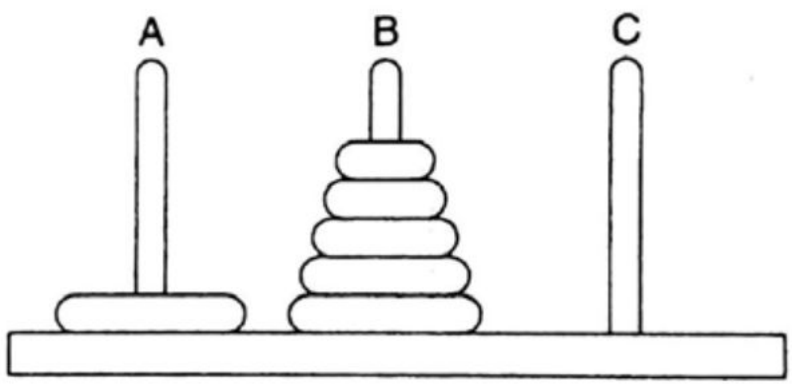


(a) Initial: $n = 6$

TOWER(N,A,B,C)

Recursion
(Tower of Hanoi)

- TOWER (1, A, B, C) means A→ C



(b) Move top five disks from peg A to peg B

Recursion
(Tower of Hanoi)

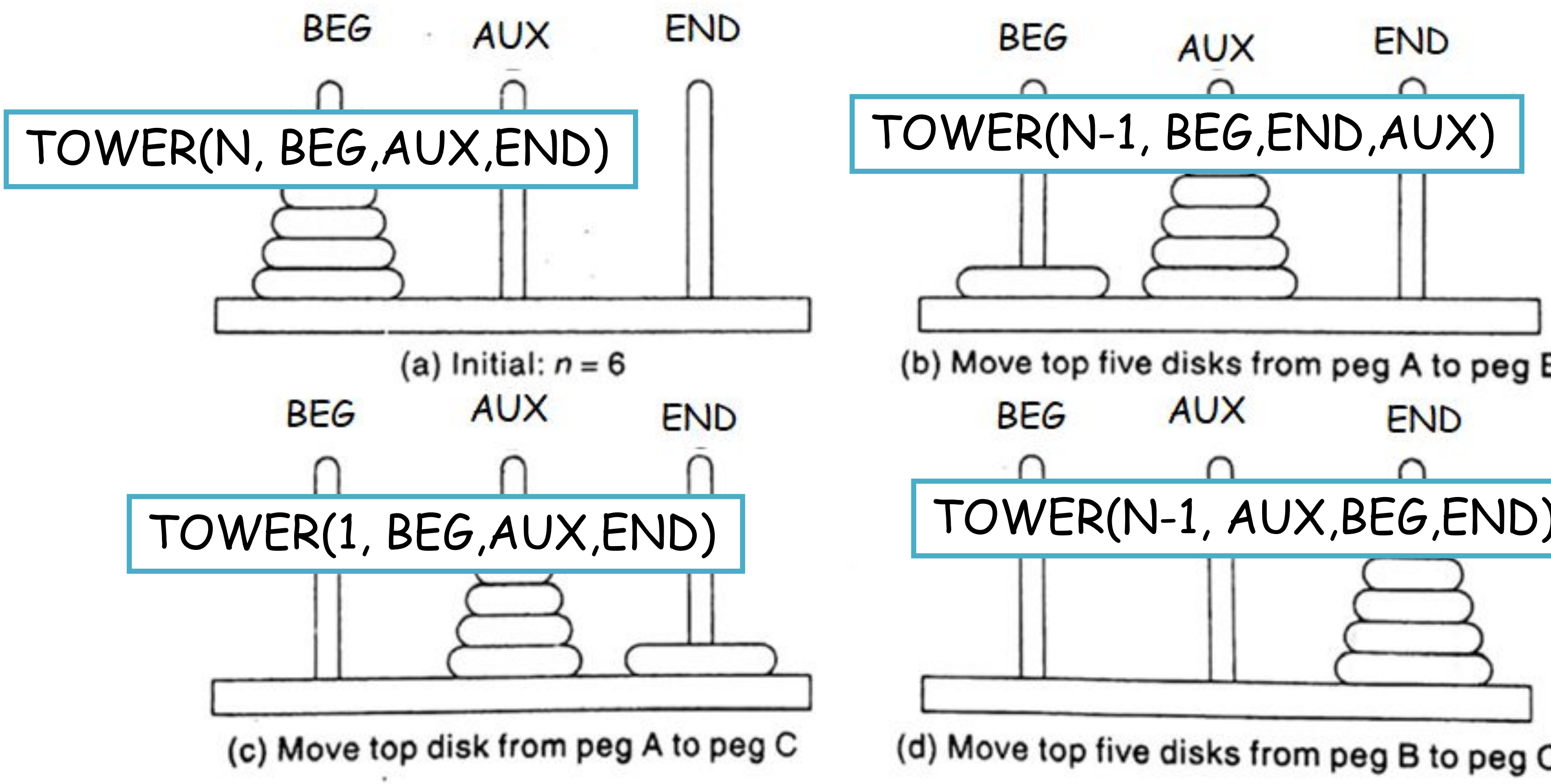


Fig. 6.16

Recursion
(Tower of Hanoi)

- TOWER (4, A, B, C) - Recursive Call

TOWER (4, A, B, C)

TOWER(3, A, C, B) **TOWER(1**, A, B, C) or A□C **TOWER(3**, B,A,C)

TOWER(2, A, B, C) **TOWER(1**, A,C,B) or A □B **TOWER(** 2, C, A, B)

TOWER(1, A, C,B) or A □B **TOWER(1**, A,B,C) or A□C **TOWER(1**, B, A, C) or B□C

Recursion (Tower of Hanoi)

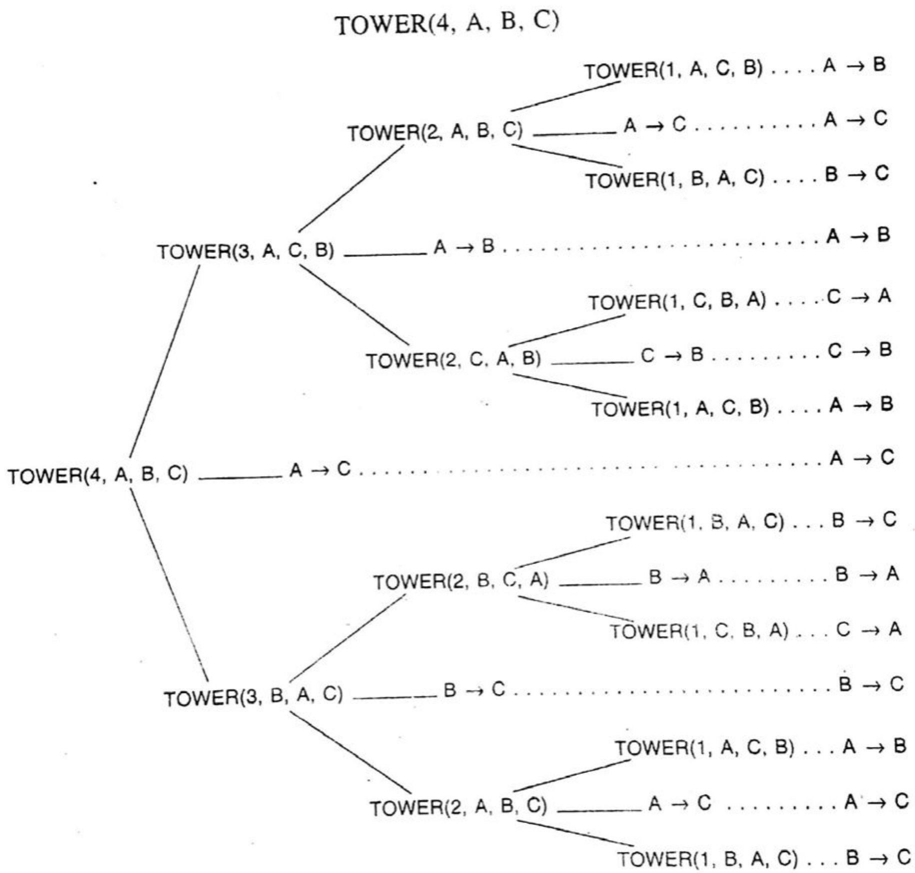


Fig. 6.17 *Recursive Solution to Towers of Hanoi Problem for $n = 4$*

Recursion (Tower of Hanoi)

Procedure 6.11: 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.

Recursion
(Tower of Hanoi)

- Is it divide-and-conquer algorithm?
  - Yes, the solution for n disk is reduced to a solution for n-1 disks and a solution for n = 1 disk.

- **Nonrecursive Procedure** (See Book and try yourself, our objective is to learn recursion)

Any Query?