Chapter 2

Recursion

Objectives

Upon completion you will be able to:

- Explain the difference between iteration and recursion
- Design a recursive algorithm
- Determine when an recursion is an appropriate solution
- Write simple recursive functions

2-1 Factorial - A Case Study

We begin the discussion of recursion with a case study and use it to define the concept.
This section also presents an iterative and a recursive solution to the factorial algorithm.

- **Recursive Defined**
- **Recursive Solution**

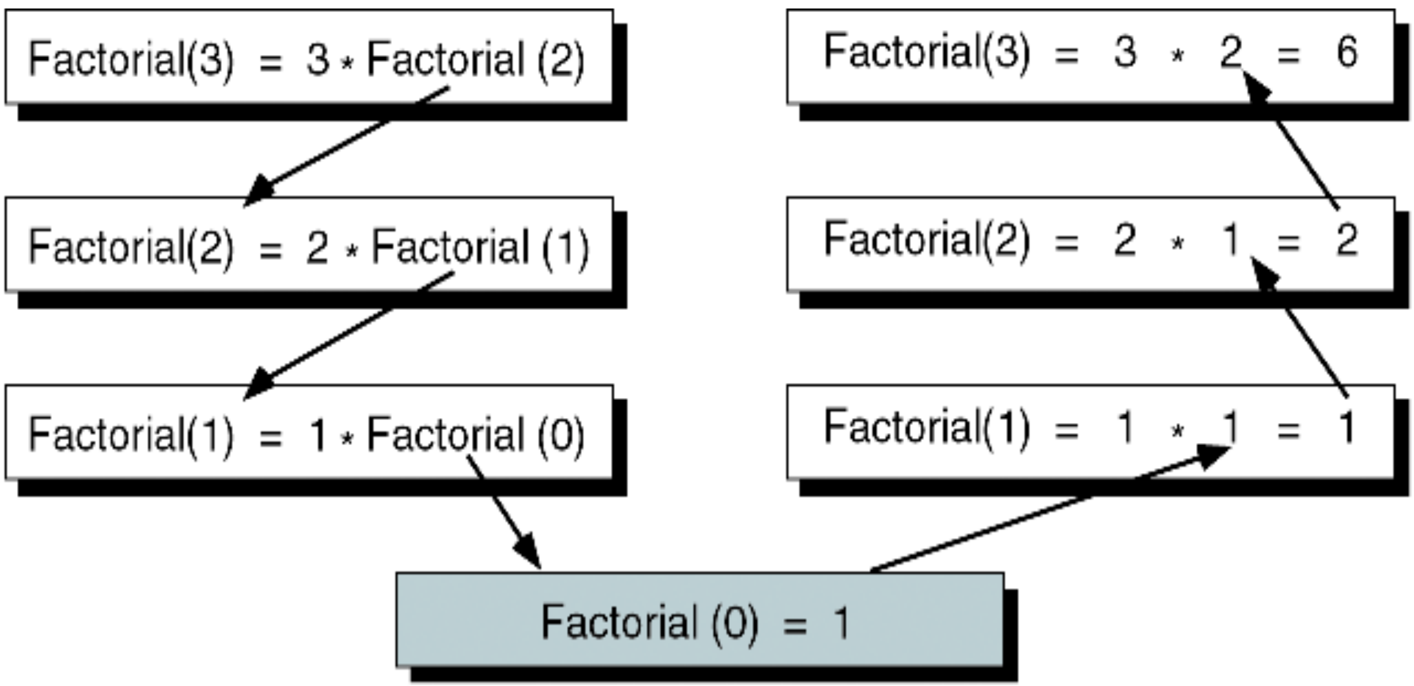
Iterative Factorial Algorithm

$$\text{Factorial}(n) = \begin{bmatrix} 1 & \text{if } n = 0 \\ n \times (n-1) \times (n-2) \times \ldots \times 3 \times 2 \times 1 & \text{if } n > 0 \end{bmatrix}$$

```
Algorithm iterativeFactorial (n)
Calculates the factorial of a number using a loop.
   Pre  n  is the number to be raised factorially
   Post n! is returned
1 set i to 1
2 set factN to 1
3 loop (i <= n)
   1  set factN to factN * i
   2  increment i
4 end loop
5 return factN
end iterativeFactorial
```

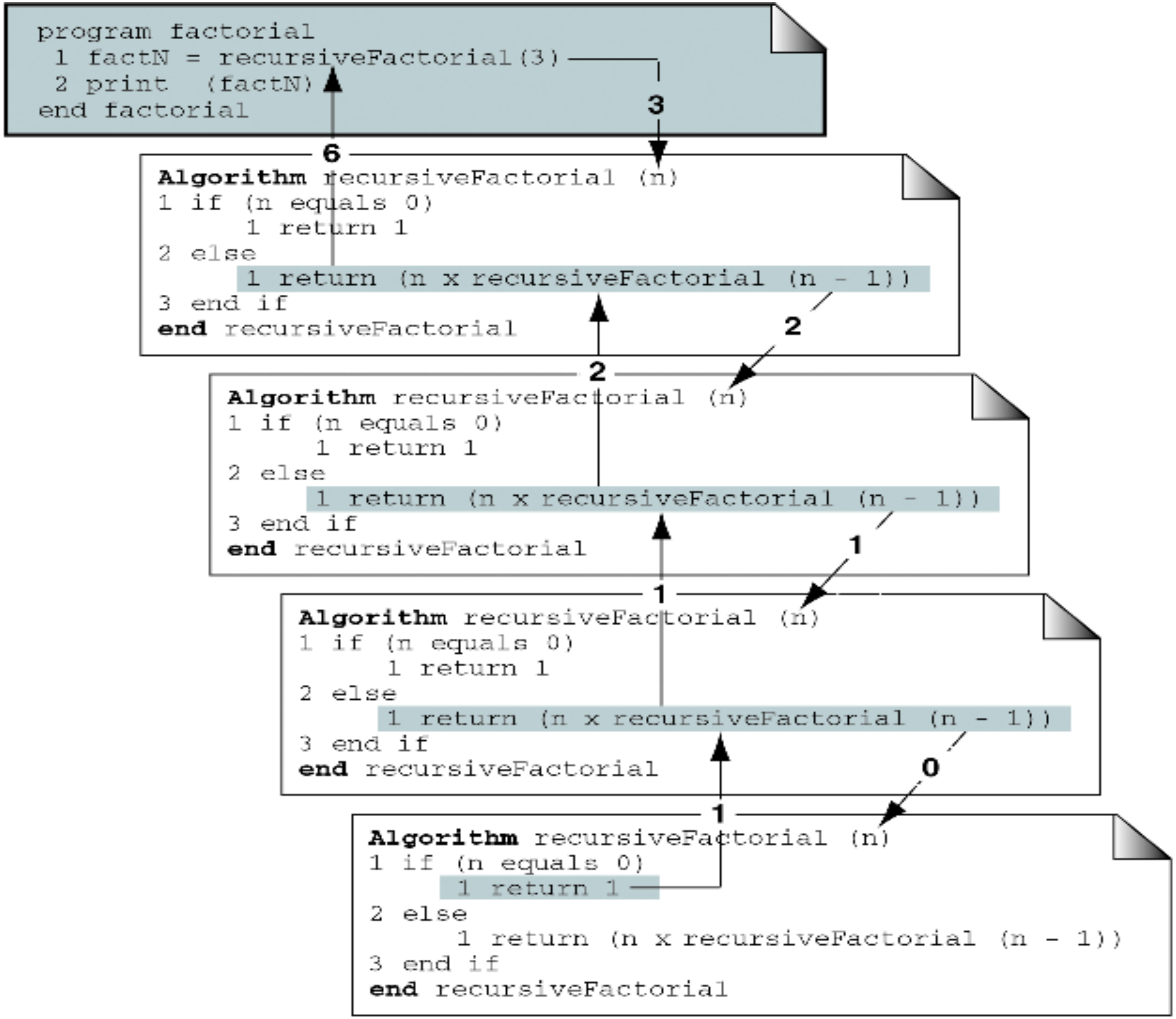
Recursive Factorial Algorithm

$$\text{Factorial}(n) = \begin{bmatrix} 1 & \text{if } n = 0 \\ n \times (\text{Factorial}(n-1)) & \text{if } n > 0 \end{bmatrix}$$



Recursive Factorial Program

```
Algorithm recursiveFactorial (n)
Calculates factorial of a number using recursion.
   Pre    n  is the number being raised factorially
   Post   n! is returned
1 if (n equals 0)
   1  return 1
2 else
   1  return (n * recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
```


program factorial
1 factN = recursiveFactorial(3)
2 print (factN)
end factorial
3
6
Algorithm recursiveFactorial (n)
1 if (n equals 0)
1 return 1
2 else
1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
2
2
Algorithm recursiveFactorial (n)
1 if (n equals 0)
1 return 1
2 else
1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
1
1
Algorithm recursiveFactorial (n)
1 if (n equals 0)
1 return 1
2 else
1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial
0
1
Algorithm recursiveFactorial (n)
1 if (n equals 0)
1 return 1
2 else
1 return (n x recursiveFactorial (n - 1))
3 end if
end recursiveFactorial

2-2 Designing Recursive Algorithms

In this section we present an analytical approach to designing recursive algorithms. We also discuss algorithm designs that are not well suited to recursion.

- **The Design Methodology**
- **Limitation of Recusion**
- **Design Implemenation**

Recursive Design Methdology

- Rules for designing a recursive algorithm
  - Determine the base case (e.g., factorial(0)=1)
  - Determine the general case (e.g., factorial (n) = n x factorial (n-1))
  - Combine the base case and the general cases into an algorithm
- Each call must reduce the size of the problem and move it toward the base case
- The base case, when reached, must terminate without a call to the recursive algorithm (must execute a return)

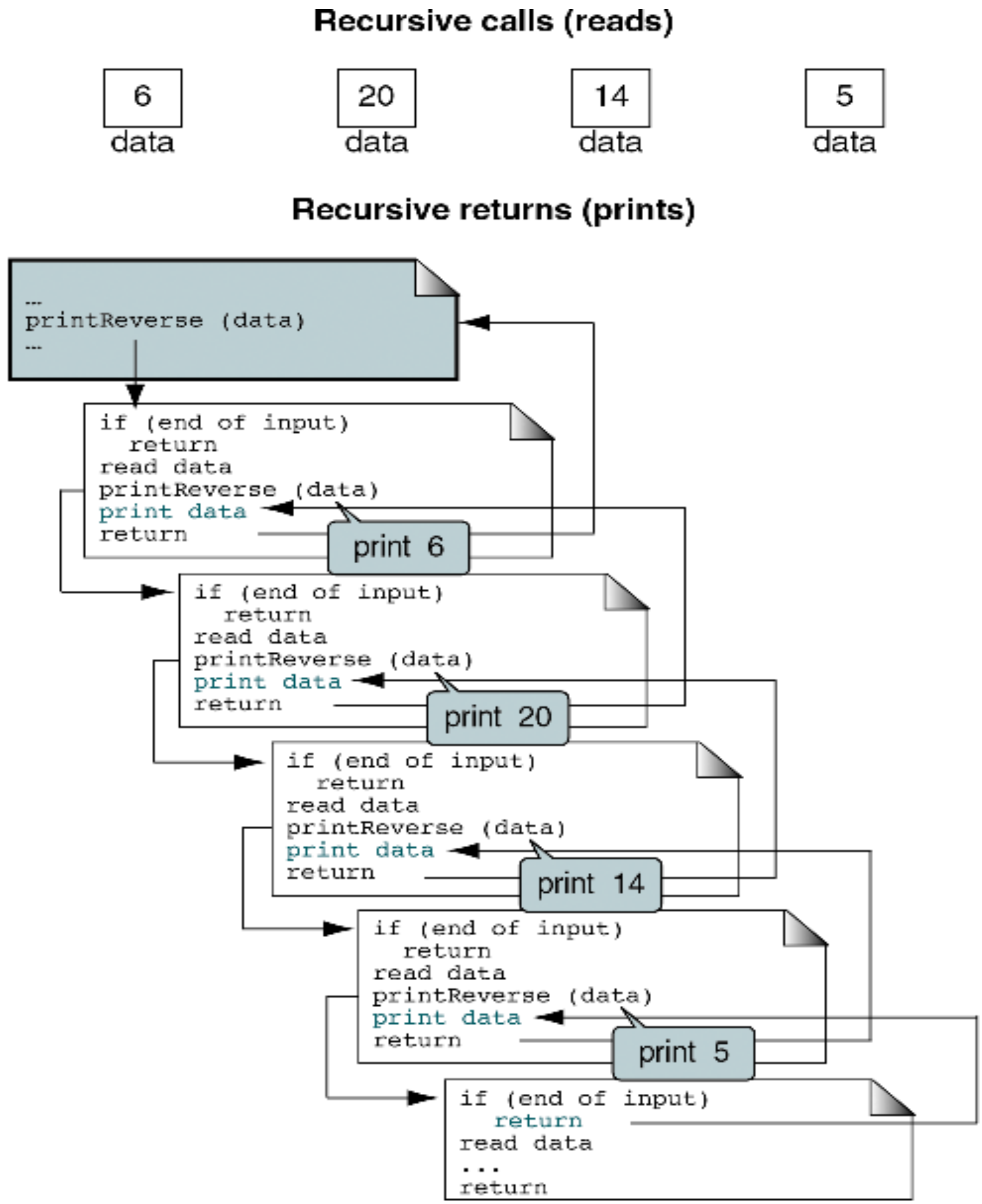
Limitations of Recursive

- Recursive solutions involve extensive overhead (time & memory)
  - Each call takes time to execute !
- Should NOT use recursion when
  - The algorithm or data structure is not naturally suited (e.g., tree structure) to recursion
    通常是「自我相似」的結構
  - Recursive solution is not shorter and more understandable
  - Recursive solution does not run within acceptable time and space limits

Reverse Keyboard Input

```
Algorithm printReverse (data)
Print keyboard data in reverse.
   Pre  nothing
   Post data printed in reverse
1 if (end of input)
   1  return
2 end if
3 read data
4 printReverse (data)
Have reached end of input: print nodes
5 print data
6 return
end printReverse
```

- **Is it naturally suited for recursion ?**
  - **No (list structure)**
- **Is it shorter and more understandable**
  - **Yes**
- **Does it run within acceptable time and space limits**
  - **No (O(n))**



2-3 Recursive Examples

Four recursive programs are developed and analyzed. Only one, the Towers of Hanoi, turns out to be a good application for recursion.

- **Greatest Common Divisor**
- **Fiboncci Numbers**
- **Prefix to Postfix Conversion**
- **The Towers of Honoi**

Greatest Common Divisor Recursive

$$
gcd = \begin{cases} a & \text{if } b = 0 \\ b & \text{if } a = 0 \\ gcd\ (b, a \bmod b) & \text{otherwise} \end{cases}
$$

```
Algorithm gcd (a, b)
Calculates greatest common divisor using the Euclidean algo-
rithm.
   Pre  a and b are positive integers greater than 0
   Post greatest common divisor returned
1 if (b equals 0)
   1  return a
2 end if
3 if (a equals 0)
   2  return b
4 end if
5 return gcd (b, a mod b)
end gcd
```

```c
#include <stdio.h>
#include <ctype.h>

// Prototype Statements
int gcd (int a, int b);

int main (void)
{
// Local Declarations
   int  gcdResult;

// Statements
   printf("Test GCD Algorithm\n");

   gcdResult = gcd (10, 25);
   printf("GCD of 10 & 25 is %d", gcdResult);
   printf("\nEnd of Test\n");
   return 0;
} // main
```

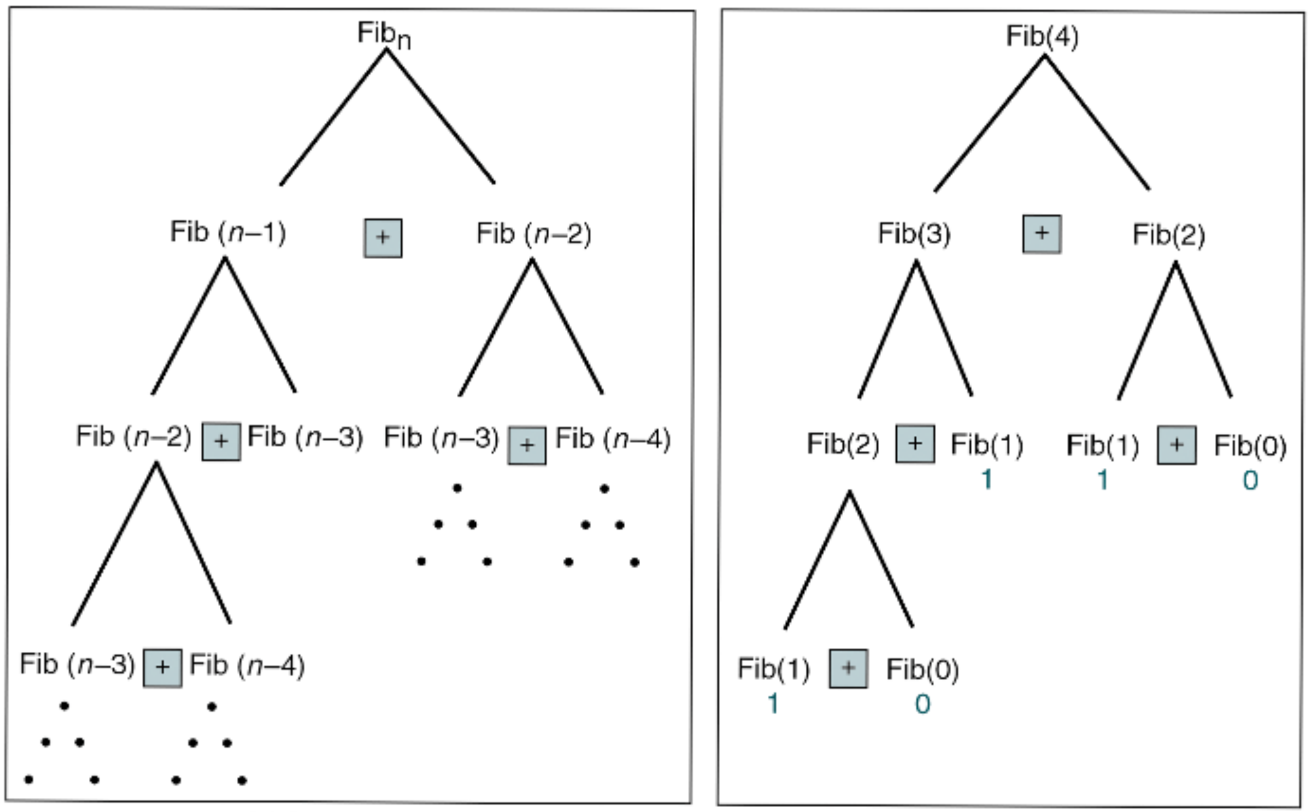
```
/* ================== gcd ==================
   Calculates greatest common divisor using the
   Euclidean algorithm.
      Pre  a and b are positive integers greater than 0
      Post greatest common divisor returned
*/
int gcd (int a, int b)
{
   // Statements
   if (b == 0)
      return a;
   if (a == 0)
      return b;
   return gcd (b, a % b);
}  // gcd
```

```
Results:
Test GCD Algorithm
GCD of 10 & 25 is 5
End of Test
```

Fibonacci Number Recursion

$$\text{Fibonacci}(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{Fibonacci}(n-1) + \text{Fibonacci}(n-2) & \text{otherwise} \end{cases}$$

For example: 0, 1, 1, 2, 3, 5, 8, 13,……



(a) Fib(*n*) (b) Fib(4)

PROGRAM 2-2 Recursive Fibonacci Series

```c
/* This program prints out a Fibonacci series.
      Written by:
      Date:
*/
#include <stdio.h>

// Prototype Statements
   long fib (long num);

int main (void)
{
// Local Declarations
   int seriesSize = 10;

// Statements
   printf("Print a Fibonacci series.\n");

```

PROGRAM 2-2 Recursive Fibonacci Series **(Continued)**

```
    for (int looper = 0; looper < seriesSize; looper++)
        {
         if (looper % 5)
            printf(", %8ld", fib(looper));
         else
            printf("\n%8ld", fib(looper));
        } // for
    printf("\n");
    return 0;
}  // main

/* ================= fib =================
   Calculates the nth Fibonacci number
      Pre  num identifies Fibonacci number
      Post returns nth Fibonacci number
*/
long fib (long num)
{
// Statements
   if (num == 0 || num == 1)
```

continued

PROGRAM 2-2 Recursive Fibonacci Series *(continued)*

```
        // Base Case
        return num;
    return (fib (num - 1) + fib (num - 2));
} // fib
```

```
Results:
Print a Fibonacci series.

     0,        1,        1,        2,        3
     5,        8,       13,       21,       34
```

Fibonacci Calls

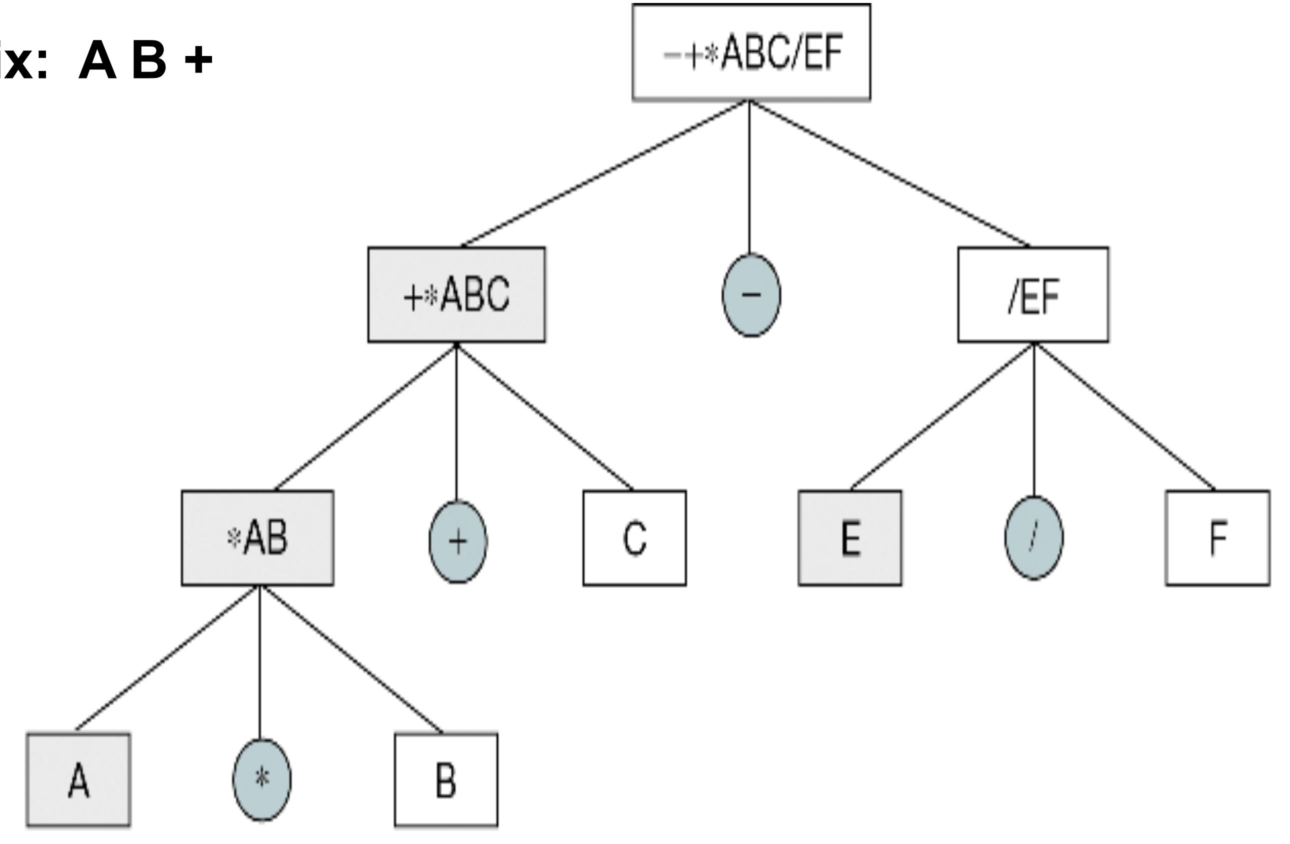
fib(*n*)	Calls	fib(n)	Calls
1	1	11	287
2	3	12	465
3	5	13	753
4	9	14	1219
5	15	15	1973
6	25	20	21,891
7	41	25	242,785
8	67	30	2,692,573
9	109	35	29,860,703
10	177	40	331,160,281

A recursive solution to calculate Fibonacci number is not efficient for large number !

Prefix to Postfix Conversion

- **Prefix: + A B**
- **Infix: A + B**
- **Postfix: A B +**

Note in Prefix: the operator is always the first character in the prefix string



ALGORITHM 2-5 Convert Prefix Expression to Postfix

```
Algorithm preToPostFix (preFixIn, postFix)
Convert a preFix string to a postFix string.
   Pre  preFix is a valid preFixIn expression
        postFix is reference for converted expression
   Post postFix contains converted expression
1  if (length of preFixIn is 1)
      Base case: one character string is an operand
   1  set postFix to preFixIn
   2  return
2  end if
   If not an operand, must be an operator
3  set operator to first character of preFixIn
   Find first expression
4  set lengthOfExpr to findExprLen (preFixIn less first char)
5  set temp to substring(preFixIn[2, lengthOfExpr])
6  preToPostFix (temp, postFix1)
   Find second postFix expression
7  set temp to prefixIn[lengthOfExpr + 1, end of string]
8  preToPostFix (temp, postFix2)
   Concatenate postfix expressions and operator
9  set postFix to postFix1 + postFix2 + operator
10 return
end preToPostFix
```

ALGORITHM 2-6 Find Length of Prefix Expression

```
Algorithm findExprLen (exprIn)
Recursively determine the length of a prefix expression.
   Pre  exprIn is a valid prefix expression
   Post length of expression returned
1 if (first character is operator)
  General Case: First character is operator
  Find length of first prefix expression
  1  set len1 to findExprLen (exprIn + 1)
  2  set len2 to findExprLen (exprIn + 1 + len2)
2 else
  Base case--first char is operand
  1  set len1 and len2 to 0
3 end if
4 return len1 + len2 + 1
end findExprLen
```

PROGRAM 2-3 Prefix to Postfix

```
/* Convert prefix to postfix expression.
      Written by:
      Date:
*/
#include <stdio.h>
#include <string.h>

#define OPERATORS "+-*/"

// Prototype Declarations
void preToPostFix  (char* preFixIn, char* exprOut);
int  findExprLen   (char* exprIn);

int main (void)
{
```

continued

PROGRAM 2-3 Prefix to Postfix *(continued)*

```
// Local Definitions
   char  preFixExpr[256]  = "-+*ABC/EF";
   char  postFixExpr[256] = "";

// Statements
   printf("Begin prefix to postfix conversion\n\n");

   preToPostFix (preFixExpr, postFixExpr);
   printf("Prefix expr:  %-s\n", preFixExpr);
   printf("Postfix expr: %-s\n", postFixExpr);

   printf("\nEnd prefix to postfix conversion\n");
   return 0;
}  // main
```

PROGRAM 2-3 Prefix to Postfix *(continued)*

```
/* ==================== preToPostFix ====================
   Convert prefix expression to postfix format.
      Pre  preFixIn is string prefix expression
           expression can contain no errors/spaces
           postFix is string variable for postfix
      Post expression has been converted
*/
void preToPostFix (char* preFixIn, char* postFix)
{
// Local Definitions
   char  operator [2];
   char  postFix1[256];
   char  postFix2[256];
   char  temp    [256];
   int   lenPreFix;

```

```
// Statements
    if (strlen(preFixIn) == 1)
       {
        *postFix          = *preFixIn;
        *(postFix + 1) = '\0';
        return;
       } // if only operand

    *operator          = *preFixIn;
    *(operator + 1) = '\0';

    // Find first expression
    lenPreFix = findExprLen (preFixIn + 1);
    strncpy (temp, preFixIn + 1, lenPreFix);
    *(temp + lenPreFix) = '\0';
    preToPostFix (temp, postFix1);
```

continued

PROGRAM 2-3 Prefix to Postfix *(continued)*

```
    // Find second expression
    strcpy (temp, preFixIn + 1 + lenPreFix);
    preToPostFix (temp, postFix2);

    // Concatenate to postFix
    strcpy (postFix, postFix1);
    strcat (postFix, postFix2);
    strcat (postFix, operator);

    return;
 } // preToPostFix
```

PROGRAM 2-3 Prefix to Postfix *(continued)*

```
/* ==================== findExprLen ====================
   Determine size of first substring in an expression.
      Pre  exprIn contains prefix expression
      Post size of expression is returned
*/
int findExprLen (char* exprIn)
{
// Local Definitions
   int  len1;
   int  len2;

```

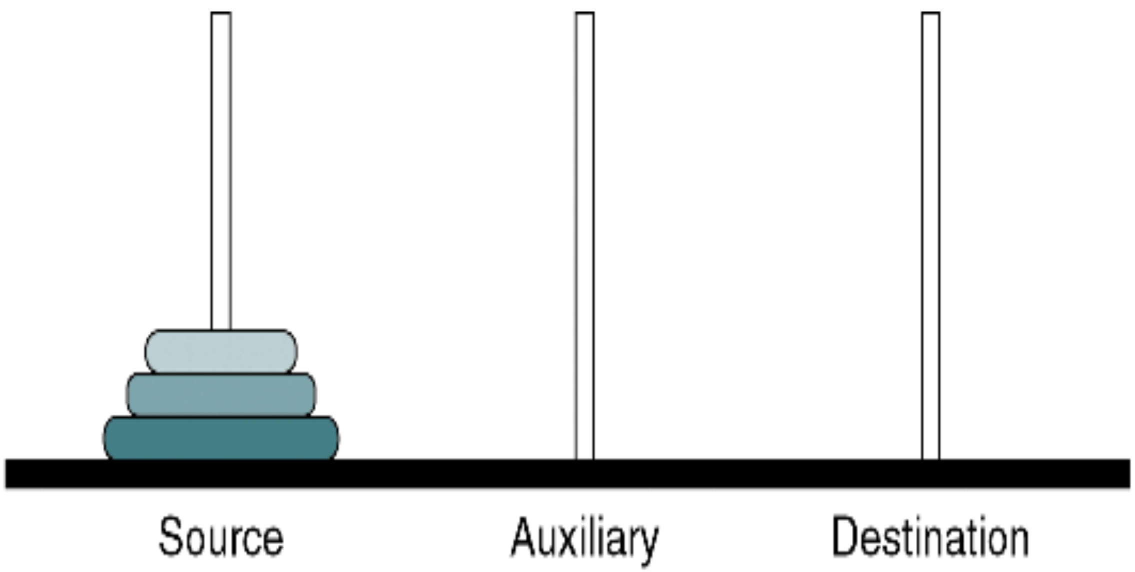
PROGRAM 2-3 Prefix to Postfix *(continued)*

```
// Statements
   if (strcspn (exprIn, OPERATORS) == 0)
         // General Case: First character is operator
         // Find length of first expression
        {
         len1 = findExprLen(exprIn + 1);

         // Find length of second expression
         len2 = findExprLen(exprIn + 1 + len1);
        } // if
   else
         // Base case--first char is operand
         len1 = len2 = 0;
   return len1 + len2 + 1;
}  // findExprLen
```

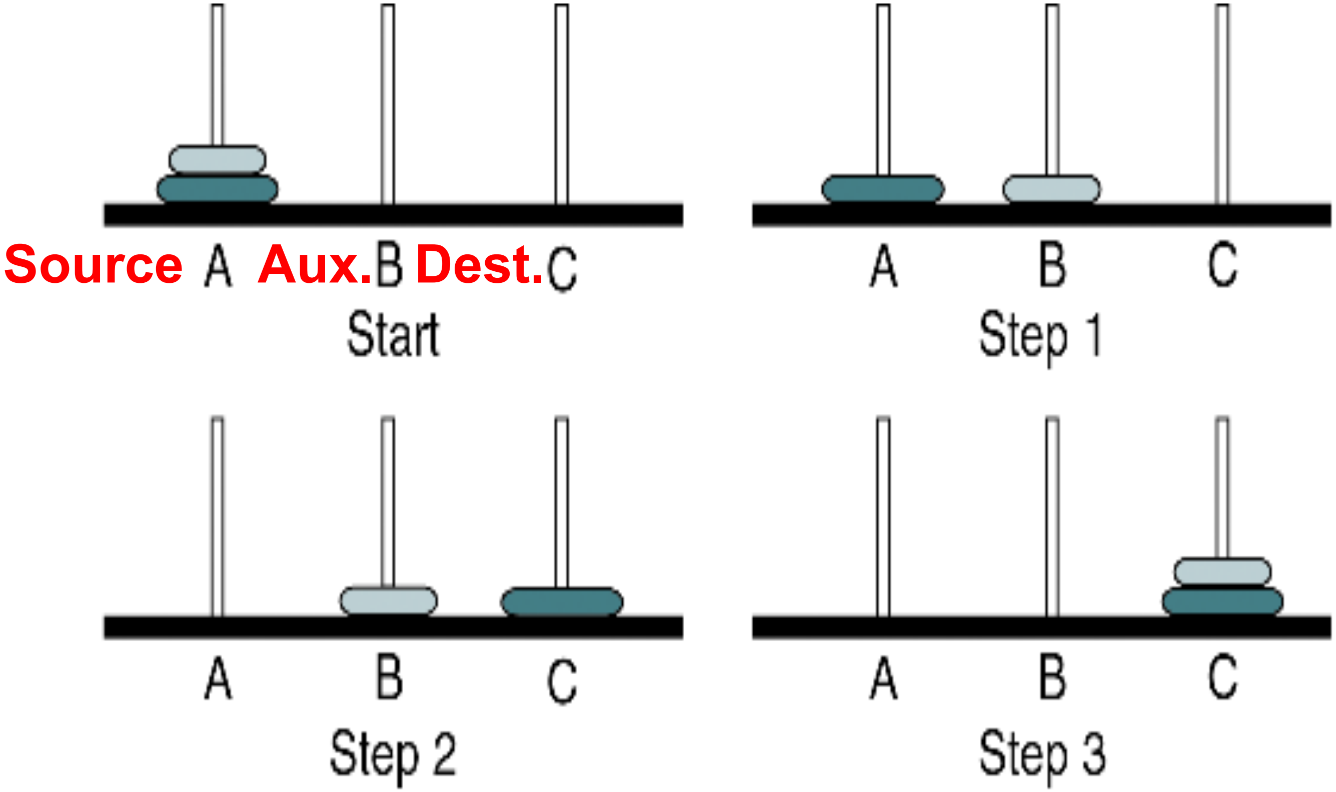
```
Results:
Begin prefix to postfix conversion

Prefix expr:  -+*ABC/EF
Postfix expr: AB*C+EF/-

End prefix to postfix conversion
```

Towers of Hanoi

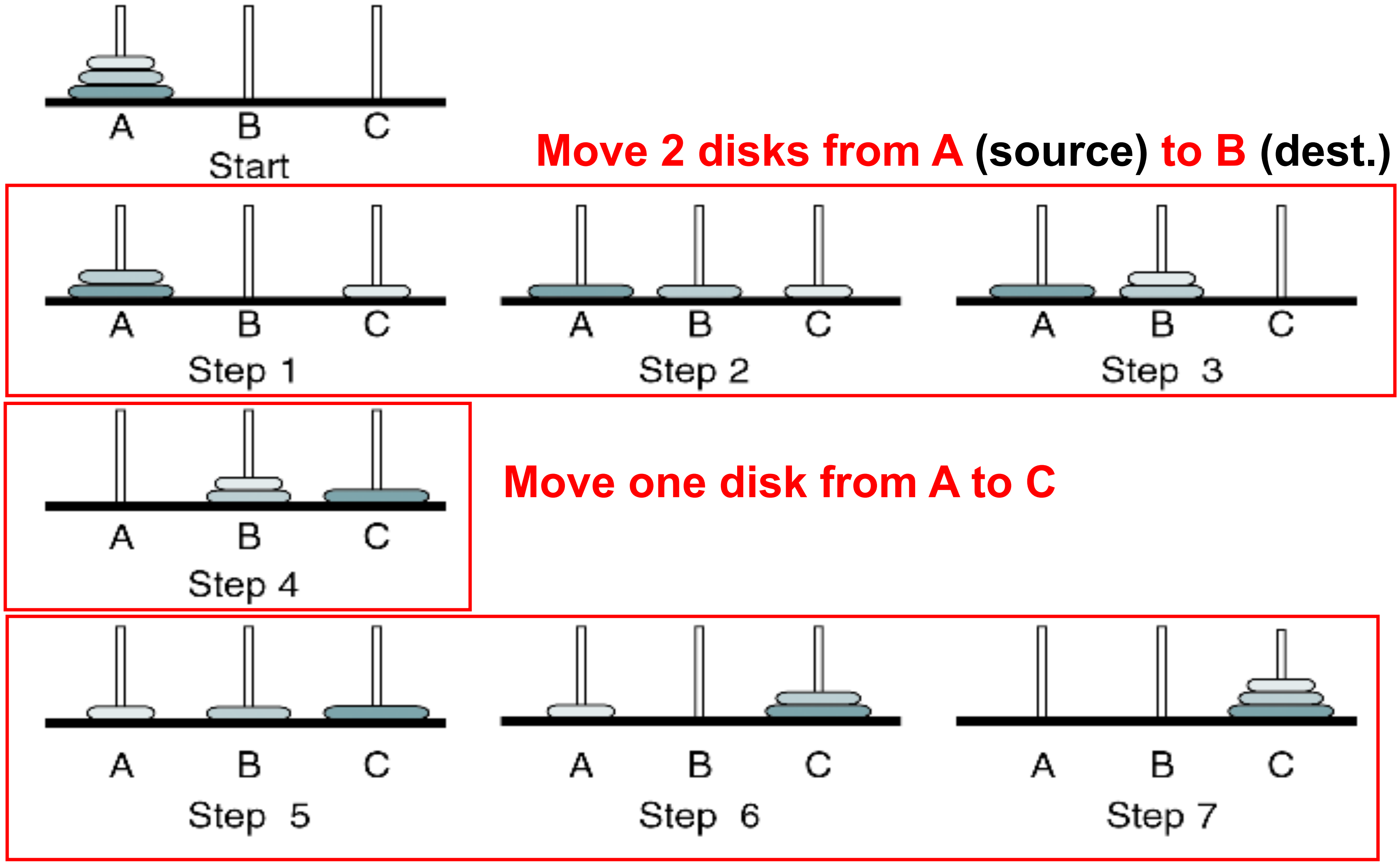
- Only one disk could be moved at a time and a larger disk must never be stacked above a smaller one
- One and only one auxiliary needle could be used for the intermediate storage of disks



Towers Solution for Two Disks



Towers Solution for Three Disks (cont.)



Move 2 disks from B (source) to C (dest.)

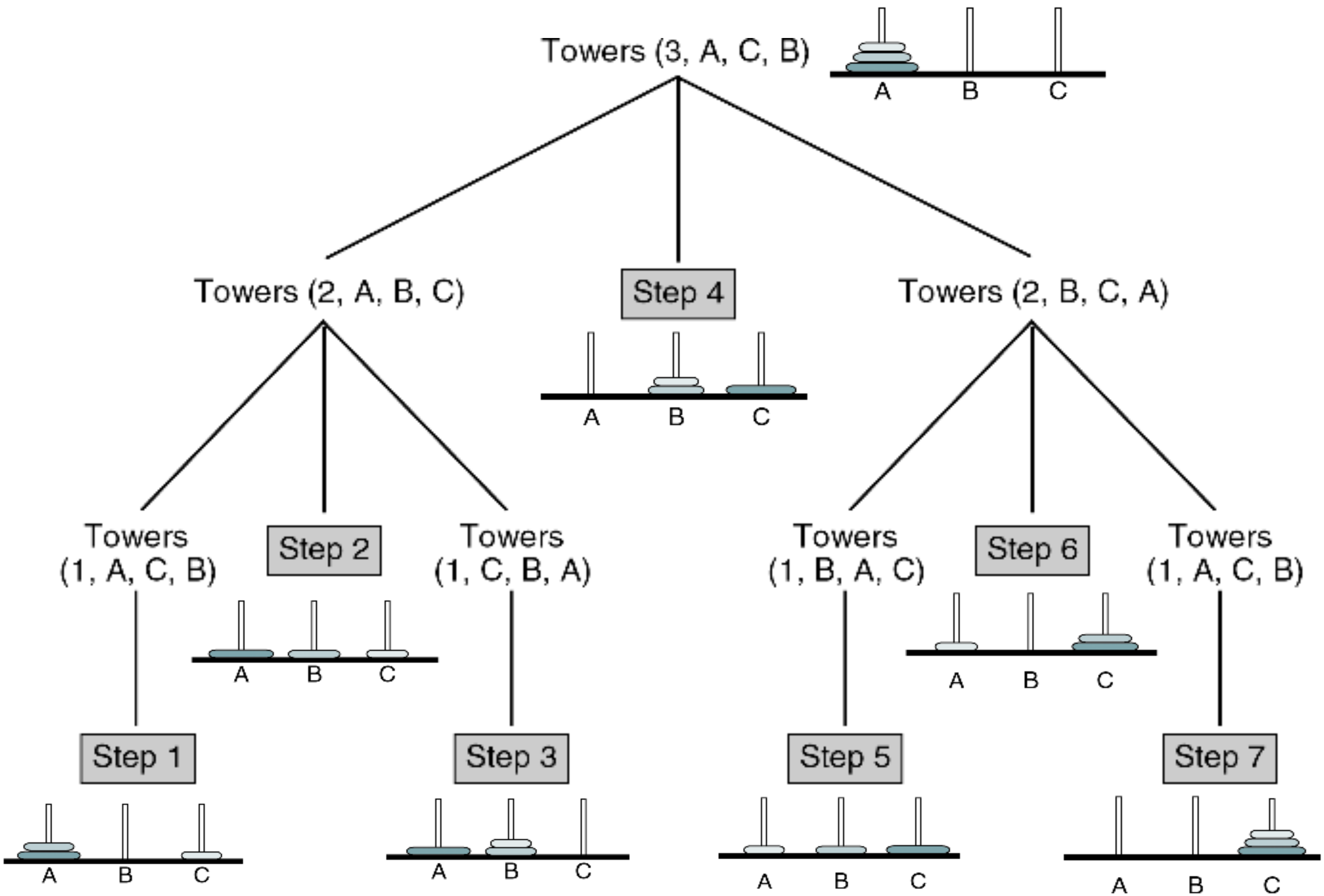
Towers Solution for N Disks

- Generalization of the problem
  - General case: Move n-1 disks from source to auxiliary
  - Base case: Move one disk from source to destination
  - General case: Move n-1 disks from aux. to destination
- Four parameters for the algorithm “Towers”
  - The number of disks to be moved
  - The source needle
  - The destination needle
  - The auxiliary needle

Pseudocode note:Tower (num,source, destination, auxiliary)

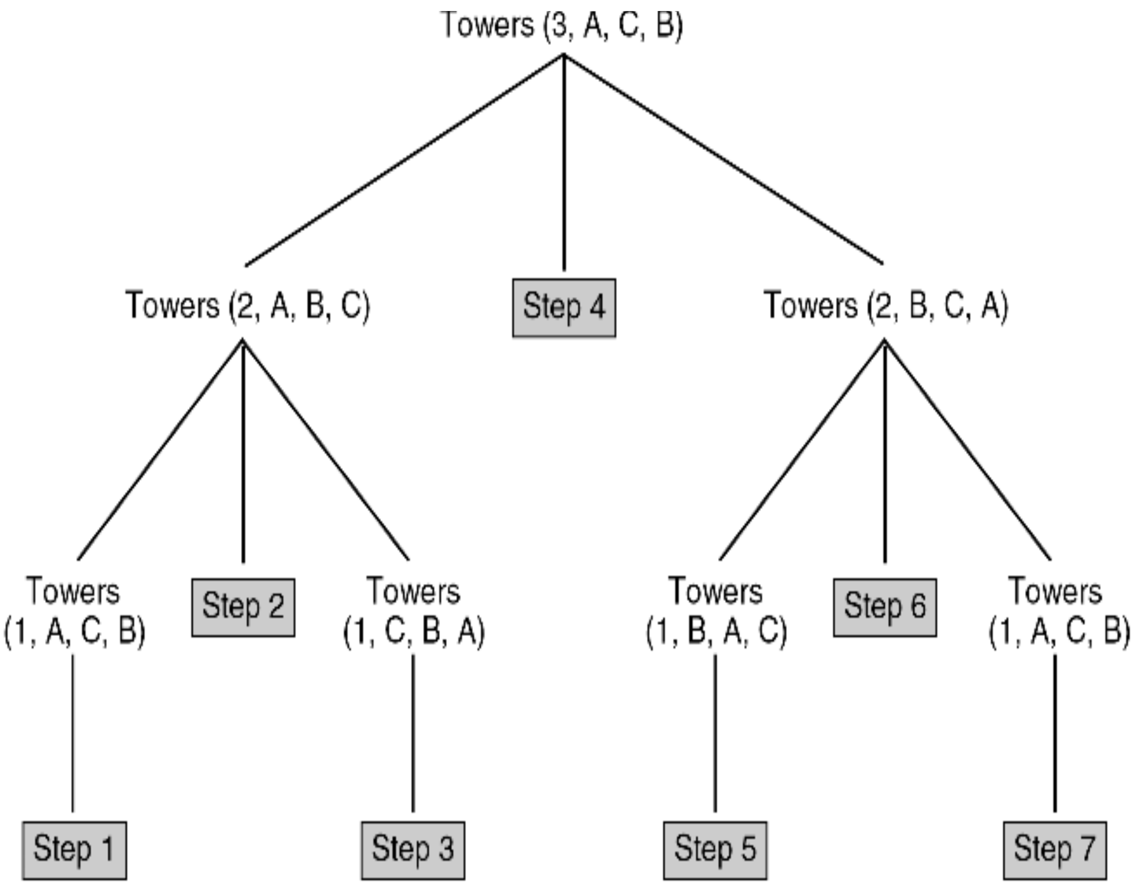
1. Call Towers (n-1, source, auxiliary, destination)
2. Move one disk from source to destination
3. Call Towers (n-1, auxiliary, destination, source

Towers Solution for Three Disks



ALGORITHM 2-7 Towers of Hanoi

```
Algorithm towers (numDisks, source, dest, auxiliary)
Recursively move disks from source to destination.
   Pre  numDisks is number of disks to be moved
        source, destination, and auxiliary towers given
   Post steps for moves printed
1 print("Towers: ", numDisks, source, dest, auxiliary)
2 if (numDisks is 1)
   1  print ("Move from ", source, " to ", dest)
3 else
   1  towers (numDisks - 1, source, auxiliary, dest, step)
   2  print ("Move from " source " to " dest)
   3  towers (numDisks - 1, auxiliary, dest, source, step)
4 end if
end towers
```



Calls:	**Output:**
`Towers (3, A, C, B)`	
`Towers (2, A, B, C)`	
`Towers (1, A, C, B)`	
	`Move from A to C`
	`Move from A to B`
`Towers (1, C, B, A)`	
	`Move from C to B`
	`Move from A to C`
`Towers (2, B, C, A)`	
`Towers (1, B, A, C)`	
	`Move from B to A`
	`Move from B to C`
`Towers (1, A, C, B)`	
	`Move from A to C`

PROGRAM 2-4 Towers of Hanoi

```
/* Test Towers of Hanoi
      Written by:
      Date:
*/
#include <stdio.h>

// Prototype Statements
   void towers (int  n,     char source,
                char dest,  char auxiliary);

int main (void)
{
// Local Declarations
   int numDisks;

```

continued

PROGRAM 2-4 Towers of Hanoi (Continued)

```
// Statements
   printf("Please enter number of disks: ");
   scanf ("%d", &numDisks);

   printf("Start Towers of Hanoi.\n\n");

   towers (numDisks, 'A', 'C', 'B');

   printf("\nI Hope you didn't select 64 "
           "and end the world!\n");
   return 0;
} // main
```

PROGRAM 2-4 Towers of Hanoi (Continued)

```
/* ==================== towers ====================
   Move one disk from source to destination through
   the use of recursion.
      Pre  The tower consists of n disks
           Source, destination, & auxiliary towers
      Post Steps for moves printed
*/
void towers (int   n,    char  source,
                   char  dest, char  auxiliary)
{
// Local Declarations
   static int step = 0;

```

PROGRAM 2-4 Towers of Hanoi *(Continued)*

```
// Statements
   printf("Towers (%d, %c, %c, %c)\n",
                  n, source, dest, auxiliary);
   if (n == 1)
      printf("\t\t\tStep %3d: Move from %c to %c\n",
             ++step, source, dest);
   else
      {
       towers (n - 1, source, auxiliary, dest);
       printf("\t\t\tStep %3d: Move from %c to %c\n",
              ++step, source, dest);
       towers (n - 1, auxiliary, dest, source);
      } // if … else
   return;
}  // towers
```