CSC13: Algorithmics and Program Design

Dr. S. Zeeshan Hussain
Dept. of Computer Science
Jamia Millia Islamia
New Delhi

Unit-Wise Syllabus

- **Algorithmic Problem Solving**: Algorithms; Problem Solving Aspect: Algorithm Devising, Design and Top-down Design; Algorithm Implementation: Essential and Desirable Features of an Algorithm; Efficiency of an Algorithm, Analysis of Algorithms, Pseudo codes; Algorithm Efficiency, Analysis and Order; Importance of Developing Efficient Algorithms; Complexity Analysis of Algorithms: Every-Case Time Complexity, Worst-Case Time Complexity, Average-Case Time Complexity, Best-Case Time Complexity.
- **Basic Algorithms – Exchanging the Values of Two Variables, Counting, Summation of a Set of Numbers, Factorial Computation, Sine Function Computation, Generation of the Fibonacci Sequence, Reversing the Digits of an Integer, Base Conversion, etc. Flowchart. Flowchart – Symbols and Conventions, Recursive Algorithms.**
- **Factoring:** Finding the square root of number, Smallest Divisor of an integer, Greatest common divisor of two integers, generating prime numbers, computing prime factors of an integer, Generation of pseudo random numbers, Raising a number to a large power, Computing the *n*th Fibonacci number.
- **Arrays, Searching and Sorting:** Single and Multidimensional Arrays, Array Order Reversal**,** Array counting, Finding the maximum number in a set, partitioning an array, Monotones Subsequence; Searching: Linear and Binary Array Search; Sorting: Sorting by selection, Exchange and Insertion. Sorting by diminishing increment, Sorting by partitioning.
- **Programming:** Introduction, Game of Life, Programming Style: Names, Documentation and Format, Refinement and Modularity; Coding, Testing and Further Refinement: Stubs and Drivers; Program Tracing, Testing, Evaluation; Program Maintenance: Program Evaluation, Review, Revision and Redevelopment; and Problem Analysis, Requirements Specification, Coding and Programming Principles.
- REFERENCES
- **Dromy: How to Solve by Computer, PE (Unit 1-4)**
- **Kruse: Data Structures and Program Design, PHI (Unit-5)**
- Robertson: Simple Program Design, A Step-by-Step Approach, Thomson

Basic Algorithms

- Exchanging the Values of Two Variables,
- Counting, Summation of a Set of Numbers,
- Factorial Computation,
- Sine Function Computation,
- Generation of the Fibonacci Sequence,
- Reversing the Digits of an Integer,
- Base Conversion, etc

Exchanging the values of two variables

```
Func (var a,b:integer)
var t: integer
begin
t =a
a=b
b=a
end

Func (var a,b:integer) // without third variable
Begin
a= a+b
b=a-b
a=a-b
end
```

Counting

Problem:

Given a set of n students' examination marks (in the range 0 to 100) make a count of number of students that passed the examination. A pass is awarded for all marks of 50 and above.

```
Fun passcount(input,output)
Const passmark =50
var  count,i,n,m
begin
count no. of marks, n
Count =0
i=0
While i<n
i=i+1
If m >=passmark  count = count+1
end
Print number of passes in a set of n marks, count
end
```

Reversing digits of a number

```
Func(n:integer)
begin
reverse:integer
reverse =0
while n > 0
begin
reverse = reverse *10 +n mod 10
n = n div 10
end
print reverse
```

Summation of set of numbers

```
func Sumofnum(n:integer)
begin
var i, sum
i=1;
  sum =0;
 while(i<=n)
begin
 prompt to enter any number
 read number
 if(number < 0)
begin
print " Sorry, the entered
   number is negative
 continue
 endif
sum = sum + number
i= i+1
endwhile
print sum
```

Sine Function Computation

$\sin x = x - x^3/3! + x^5/5! - x^7/7! + x^9/9! \ldots\ldots$

Algorithm:

```
Function SineFn(n:integer)
begin
 i, j, n, fact, sign = - 1: integer
 x, p, sum := 0.0 : real;
read  x
i:=1
for  i <= n do
begin
 p := 1
 fact := 1
j:=1
 for j <=i do
begin
 p := p * x
 fact := fact * j
j:=j+1
endfor  //inner
sign := - 1 * sign
      sum += sign * p / fact
      i+=2;
      endfor // outer
      printf("sin %0.2f = %f", x, sum); return 0;
Endfunction
```

Source Code

```
#include <stdio.h>
int main() { int i, j, n, fact, sign = - 1;
float x, p, sum = 0.0;
 printf("Enter the value of x : ");
scanf("%f", &x);
printf("Enter the value of n : ");
scanf("%d", &n);
for (i = 1; i <= n; i += 2) {
 p = 1; fact = 1;
 for (j = 1; j <= i; j++)
{ p = p * x;
fact = fact * j;
}
sign = - 1 * sign; sum += sign * p / fact;
}
 printf("sin %0.2f = %f", x, sum); return 0;
}
```

Base Conversion

Decimal	Hex	Oct	Bin
0	0	000	0000
1	1	001	0001
2	2	002	0010
3	3	003	0011
4	4	004	0100
5	5	005	0101
6	6	006	0110
7	7	007	0111
8	8	010	1000
9	9	011	1001
10	A	012	1010
11	B	013	1011
12	C	014	1100
13	D	015	1101
14	E	016	1110
15	F	017	1111

Base Conversion

- **Conversion from Decimal number system to other number systems [Binary, Octal, Hexadecimal]**
- To convert a given number from decimal number system to any other number system, follow these steps:

1. Divide the decimal number by r i.e. base of the other system (2, 8, or 16). Remember the quotient and the remainder of this division.
2. After that, divide the quotient (from the first division) by r, again remembering the quotient and the remainder.
3. Keep dividing your new quotient by r until you get a quotient of 0. After each division, keep track of the remainder.
4. When you reach a quotient of 0, the remainders of all the divisions (written in reverse order) will be the equivalent number in base r number system. [Reverse order mean that, the first remainder that you got in step-1 will be the least significant digit (LSD) of the number in base r number system].

Base Conversion

- **Conversion from Other number systems [binary, octal, hexadecimal] à to à decimal number system**
- The conversion process from other number systems [i.e. binary, octal, and hexadecimal] to decimal number system has the same procedure. Here any binary number can be converted into its equivalent decimal number using the weights assigned to each bit position. Incase of binary the weights are 20 (Units), 21 (twos), 22 (fours), 23 (eights), 24 (sixteen) and so on.
- Similarly in case of octal the weights are 8^0,8^1,8^2,8^3 , 8^4 and so on.
- For hexadecimal the weights are 16^0,16^1,16^2,16^3,16^4 and so on. Here few steps are given which are helpful in faster and easy conversion of other systems to decimal number system.

1. *Write the given (i.e. 2, 8, or 16) base number*
2. *Write the corresponding weight x0 , x1 , x2 , x3 ,...., under each digit.*
3. *Cross out any weight under a 0 (means that any 0 involve in given number).*
4. *Add the remaining weights.*

Base Conversion

2	65	
2	32	1
2	16	0
2	8	0
2	4	0
2`	2	0
2	1	0

$(65)_{10} = (1000001)_2$

$(1000001)_2 = 1 \text{x}\ 2^6 + 0\text{x}2^5 + 0\text{x}2^4 + 0\text{x}2^3 + 0\text{x}2^2 + 0\text{x}2^1 + 1\text{x}2^0$

$= 64+1 = (65)_{10}$

$(65)_{10} = (001\ 000\ 001)_2$

$= (101)_8$

$(101)_8 = 1 \text{ x } 8^2 + 0 \text{ x } 8^1 + 1 \text{ x } 8^0$

$= 64+1 = (65)_{10}$

$(65)_{10} = 65/16$ quotient-4 remainder- 1 $= (41)_{16}$

$(41)_{16} = 4\text{x } 16^1 + 1 \text{ x } 16^0 = 64+1 = (65)_{10}$

$(101)_8 = (00100\ 0001)_2 = (41)_{16}$

$(8)_{10} = (\quad ? \quad)_2 = (\ ?\ )_8$

Base Conversion

```
// C Program to convert decimal
    to any given base
#include <stdio.h>
#include <string.h>
 // To return char for a value. For
    example '2'
// is returned for 2. 'A' is returned
    // for 10. 'B' for 11
char reVal(int num) {
   if (num >= 0 && num <= 9)
      return (char)(num + '0');
   else
      return (char)(num - 10 + 'A');
}
 // Utility function to reverse a
    string
void strev(char *str) {
   int len = strlen(str);
   int i;
   for (i = 0; i < len/2; i++)    {
      char temp = str[i];
      str[i] = str[len-i-1];
      str[len-i-1] = temp;
   }
}
 // Function to convert a given
    decimal number
// to a base 'base' and
char* fromDeci(char res[], int
    base, int inputNum) {
   int index = 0;  // Initialize index
    of result
    // Convert input number is
    given base by repeatedly
   // dividing it by base and taking
    remainder
   while (inputNum > 0)    {
      res[index++] =
    reVal(inputNum % base);
      inputNum /= base;
   }
   res[index] = '\0';
    // Reverse the result
   strev(res);
      return res;
}

// Main program
int main()
{
   int inputNum, base = 16;
   char res[100];
   printf("\n Enter a decimal
    number: ");
   scanf("%d",&inputNum);
   printf("\n Enter base to be
    converted into: ");
   scanf("%d",&base);

   printf("Equivalent of decimal
    number %d in base %d is "
       " %s\n", inputNum, base,
    fromDeci(res, base,
    inputNum));
   return 0;
}
```

Base Conversion

- ***Program to convert decimal number to its binary equivalent.***

```
#include <math.h>
#include <stdio.h>
long long convert(int n);
int main() {
   int n;
   printf("Enter a decimal number: ");
   scanf("%d", &n);
   printf("%d in decimal = %lld in binary", n,
    convert(n));
   return 0;
}
long convert(int n) {
long bin = 0;
   int rem, i = 1, step = 1;
   while (n != 0) {
      rem = n % 2;
      printf("Step %d: %d/2, Remainder = %d,
   Quotient = %d\n", step++, n, rem, n / 2);
      n /= 2;
      bin += rem * i;
      i *= 10;
   }
   return bin;
}
```

/*Output

```
Enter a decimal number: 19
Step 1: 19/2, Remainder = 1, Quotient = 9
Step 2: 9/2, Remainder = 1, Quotient = 4
Step 3: 4/2, Remainder = 0, Quotient = 2
Step 4: 2/2, Remainder = 0, Quotient = 1
Step 5: 1/2, Remainder = 1, Quotient = 0
19 in decimal = 10011 in binary */
```

Base Conversion

```
// Program to convert binary no
   into  Decimal equvalent
#include <math.h>
#include <stdio.h>
int convert(long n);
int main() {
   long long n;
   printf("Enter a binary number: ");
   scanf("%ld", &n);
   printf("%ld in binary = %d in
    decimal", n, convert(n));
   return 0;
}
```

```
int convert(long n) {
  int dec = 0, i = 0, rem;
  while (n != 0) {
      rem = n % 10;
      n /= 10;
      dec += rem * pow(2, i);
      ++i;
  }
  return dec;
}
/*Output

Enter a binary number: 110110111
110110111 in binary = 439*/
```

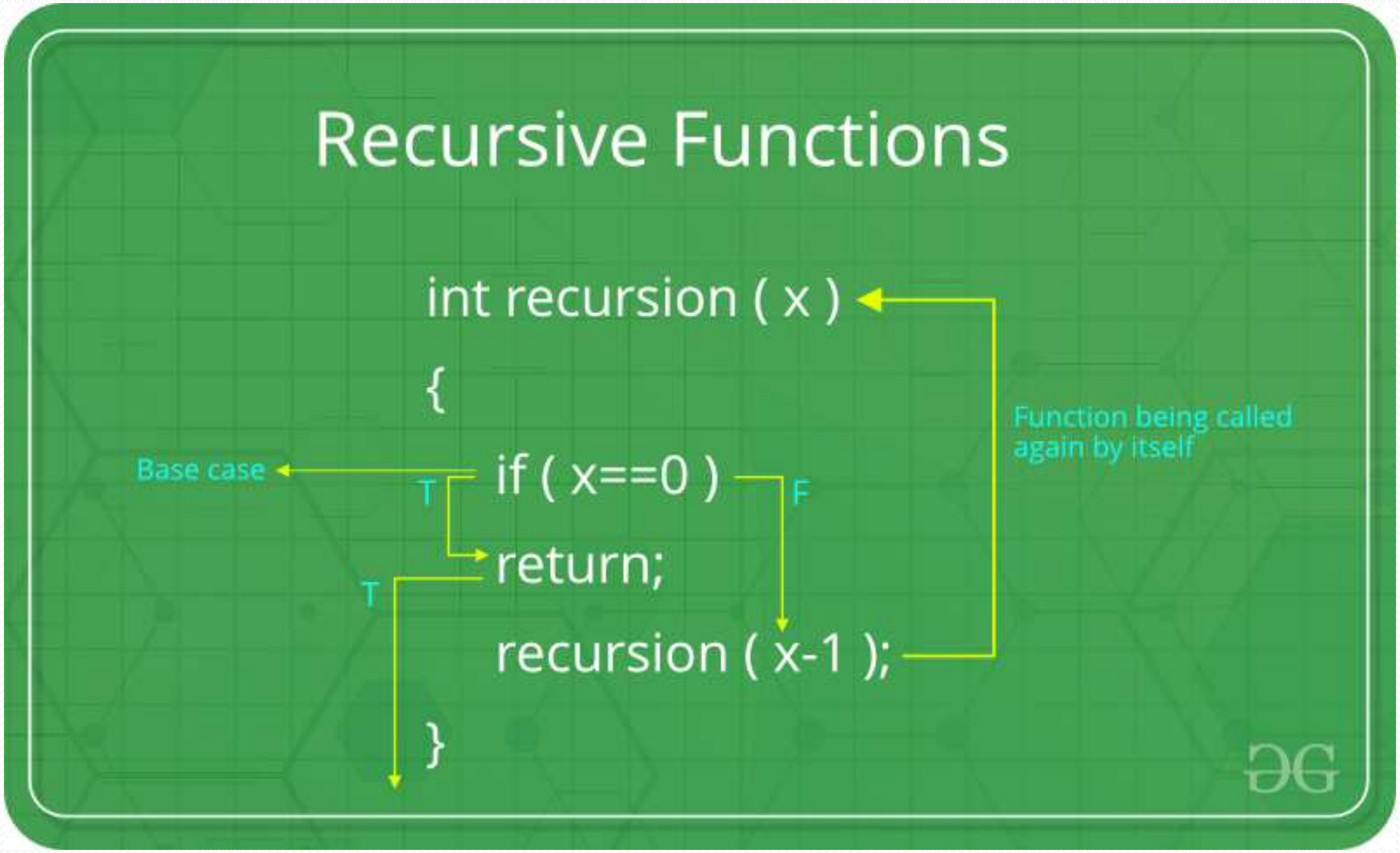
Recursive Function

- In programming terms a recursive function can be defined as a routine that calls itself directly or indirectly.
- Using recursive algorithm, certain problems can be solved quite easily. ***Towers of Hanoi (TOH)*** is one such programming exercise. Try to write an *iterative* algorithm for TOH. Moreover, every recursive program can be written using iterative methods.
- Mathematically recursion helps to solve few puzzles easily. **For example, a routine interview question, In a party of N people, each person will shake her/his hand with each other person only once. On total how many hand-shakes would happen?**
  Solution:
  It can be solved in different ways, graphs, recursion, etc. Let us see, how recursively it can be solved.
  There are N persons. Each person shake-hand with each other only once. **Considering N-th person, (s)he has to shake-hand with (N-1) persons**.
- Now the problem reduced to small instance of (N-1) persons. Assuming T_N as total shake-hands, it can be formulated recursively.
  $\boldsymbol{T_N = (N\text{-}1) + T_{N\text{-}1}}$ **[$\boldsymbol{T_1 = 0}$**, i.e. the last person have already shook-hand with every one]
  Solving it recursively yields an arithmetic series, which can be evaluated to **N(N-1)/2.**

Recursive Function

- ***<u>Exercise:</u>* In a party of N couples, only one gender (either male or female) can shake hand with every one. How many shake-hands would happen?**
  Usually recursive programs results in poor time complexities. An example is Fibonacci series. The time complexity of calculating **n-th Fibonacci number using recursion is approximately 1.6^n. It means the same computer takes almost 60% more time for next Fibonacci number**. Recursive Fibonacci algorithm has overlapped subproblems. There are other techniques like *dynamic programming* to improve such overlapped algorithms.
  However, few algorithms, (e.g. merge sort, quick sort, etc…) results in optimal time complexity using recursion.
  Base Case:
  One critical requirement of recursive functions is termination point or base case. Every recursive program must have base case to make sure that the function will terminate. Missing base case results in unexpected behaviour.

Recursive Function



Recursive Functions

```
int recursion ( x )
{
    if ( x==0 )
    return;
    recursion ( x-1 );
}
```

Recursive Function Example

- ***To find the sum of N Natural numbers***

10+9+8+7+6+5+4+3+2+1 = ? Here n =10

s= n+sum(n-1), n>0

=10+sum(9)

=10+9+sum(8)

=10+9+8+sum(7)

= 10+9+8+7+6+5+4+3+2+sum(1)

10+9+8+7+6+5+4+3+2+1+sum(0)

=55

History of Tower of Hanoi

- There is a story about an ancient temple in India (Some say it's in Vietnam – hence the name Hanoi) has a large room with three towers surrounded by 64 golden disks.
- These disks are continuously moved by priests in the temple. According to a prophecy, when the last move of the puzzle is completed the world will end.
- These priests acting on the prophecy, follow the immutable rule by Lord Brahma of moving these disk one at a time.
- Hence this puzzle is often called Tower of Brahma puzzle.
- Tower of Hanoi is one of the classic problems to look at if you want to learn recursion.
- It is good to understand how recursive solutions are arrived at and how parameters for this recursion are implemented.

Tower of Hanoi

- **Tower of Hanoi is a mathematical puzzle where we have three rods and n disks. The objective of the puzzle is to move the entire stack to another rod, obeying the following simple rules:**

1. **Only one disk can be moved at a time.**
2. **Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.**
3. **No disk may be placed on top of a smaller disk.**

Approach :

Take an example for 2 disks :

- Let rod 1 = 'A', rod 2 = 'B', rod 3 = 'C'.
- Step 1 : Shift first disk from 'A' to 'B'.
- Step 2 : Shift second disk from 'A' to 'C'.
- Step 3 : Shift first disk from 'B' to 'C'. The pattern here is : Shift 'n-1' disks from 'A' to 'B'. Shift last disk from 'A' to 'C'. Shift 'n-1' disks from 'B' to 'C'.

Tower of Hanoi

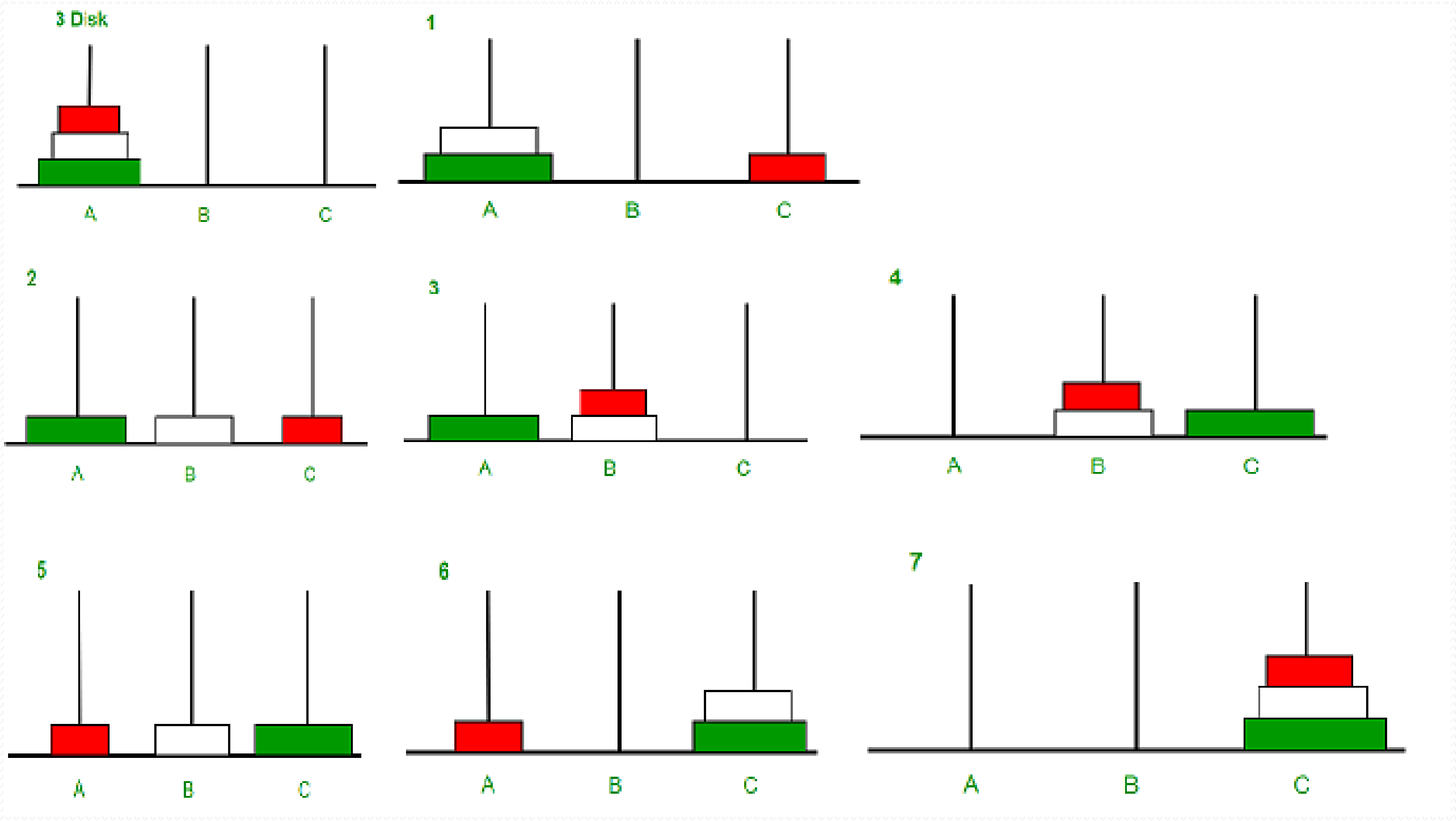


Image illustration for 3 disks

ToH

- The question is what is the minimum number of moves(aN) required to move all the N–disks to another peg (rod).
- Let's look at a recursive solution
- ***One can already see that a1=1,a2=3,a3)=7 and so on.***
- For a given N number of disks, the way to accomplish the task in a minimum number of steps is:
- Move the top N–1 disks to an intermediate peg.
- Move the bottom disk to the destination peg.
- Finally, move the N–1 disks from the intermediate peg to the destination peg.
- Therefore, the recurrence relation for this puzzle would become:
- ***a1=1, a2=3; aN=2aN–1+1; N ≥ 2***

$$
\begin{aligned}
a_N &= 2a_{N-1} + 1 \\
&= 2(2a_{N-2} + 1) + 1 \\
&= 2^2 a_{N-2} + 2 + 1 \\
&= 2^3 a_{N-3} + 2^2 + 2 + 1 \\
&= 2^{N-1} a_{N-(N-1)} + 2^{n-2} + \ldots + 2^2 + 2 + 1 \\
&= 2^N - 1
\end{aligned}
$$

Tower of Hanoi- Example

Examples:

<u>Input : 2</u>

Output : Disk 1 moved from A to B

Disk 2 moved from A to C

Disk 1 moved from B to C

<u>Input : 3</u>

Output : Disk 1 moved from A to C

Disk 2 moved from A to B

Disk 1 moved from C to B

Disk 3 moved from A to C

Disk 1 moved from B to A

Disk 2 moved from B to C

Disk 1 moved from A to C

Recursive Function for ToH

- ***// C recursive function to solve tower of hanoi puzzle***

```
#include <stdio.h>
 void towerOfHanoi(int n, char from_rod,
                char to_rod, char aux_rod)
{
    if (n == 1)
    {
       printf("\nMove disk 1 from rod %c to  rod %c" , from_rod, to_rod);
       return;
    }
    towerOfHanoi(n - 1, from_rod, aux_rod, to_rod);
    printf("\nMove disk %d from rod %c to  rod %c" , n, from_rod, to_rod);
towerOfHanoi(n - 1, aux_rod, to_rod, from_rod);
}

int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B'); // A, B and C are names of rods
    return 0;
}
```

ToH

Now one can solve this relation in a recusion as follows-

Disks	Move
1	1
2	3
3	7
4	15
5	31
6	63
7	127
8	255
9	511
10	1023

ToH

Output: ***Tower of Hanoi Solution for 4 disks:***

A: [4, 3, 2, 1] B: [] C: []

1. Move disk from rod A to rod B A: [4, 3, 2] B: [1] C: []
2. Move disk from rod A to rod C A: [4, 3] B: [1] C: [2]
3. Move disk from rod B to rod C A: [4, 3] B: [] C: [2, 1]
4. Move disk from rod A to rod B A: [4] B: [3] C: [2, 1]
5. Move disk from rod C to rod A A: [4, 1] B: [3] C: [2]
6. Move disk from rod C to rod B A: [4, 1] B: [3, 2] C: []
7. Move disk from rod A to rod B A: [4] B: [3, 2, 1] C: []
8. Move disk from rod A to rod C A: [] B: [3, 2, 1] C: [4]
9. Move disk from rod B to rod C A: [] B: [3, 2] C: [4, 1]
10. Move disk from rod B to rod A A: [2] B: [3] C: [4, 1]
11. Move disk from rod C to rod A A: [2, 1] B: [3] C: [4]
12. Move disk from rod B to rod C A: [2, 1] B: [] C: [4, 3]
13. Move disk from rod A to rod B A: [2] B: [1] C: [4, 3]
14. Move disk from rod A to rod C A: [] B: [1] C: [4, 3, 2]
15. Move disk from rod B to rod C A: [] B: [] C: [4, 3, 2, 1]

ToH (Recursive)

```
#include <stdio.h>
// Assuming n-th disk is bottom disk
    (count down)
void tower(int n, char sourcePole,
      char destinationPole, char
    auxiliaryPole)
{
   // Base case (termination condition)
if( n ==0)
   return;

// Move first n-1 disks from source pole
// to auxiliary pole using destination as
// temporary pole
tower(n - 1, sourcePole, auxiliaryPole,
   destinationPole);
// Move the remaining disk from source
// pole to destination pole
printf("nMove the disk  %d" from  %c to
    %c",n,  sourcePole ,destinationPole ");

// Move the n-1 disks from auxiliary (now
    source)  pole to destination pole using
    source pole as  temporary (auxiliary)
    pole
tower(n - 1, auxiliaryPole,
    destinationPole,   sourcePole);
}
 int main()
{    tower(3, 'S', 'D', 'A');
    return 0; }
```

TOH

- **Output : For 3 disks**

```
Move the disk 1 from S to D
Move the disk 2 from S to A
Move the disk 1 from D to A
Move the disk 3 from S to D
Move the disk 1 from A to S
Move the disk 2 from A to D
Move the disk 1 from S to D
```

- ***While the 3-disk puzzle required 7 steps. 4-disk requires 7+1+7 = 15 steps to be solved.***

Q & A