Lecture - 8
On
Stacks, Recursion

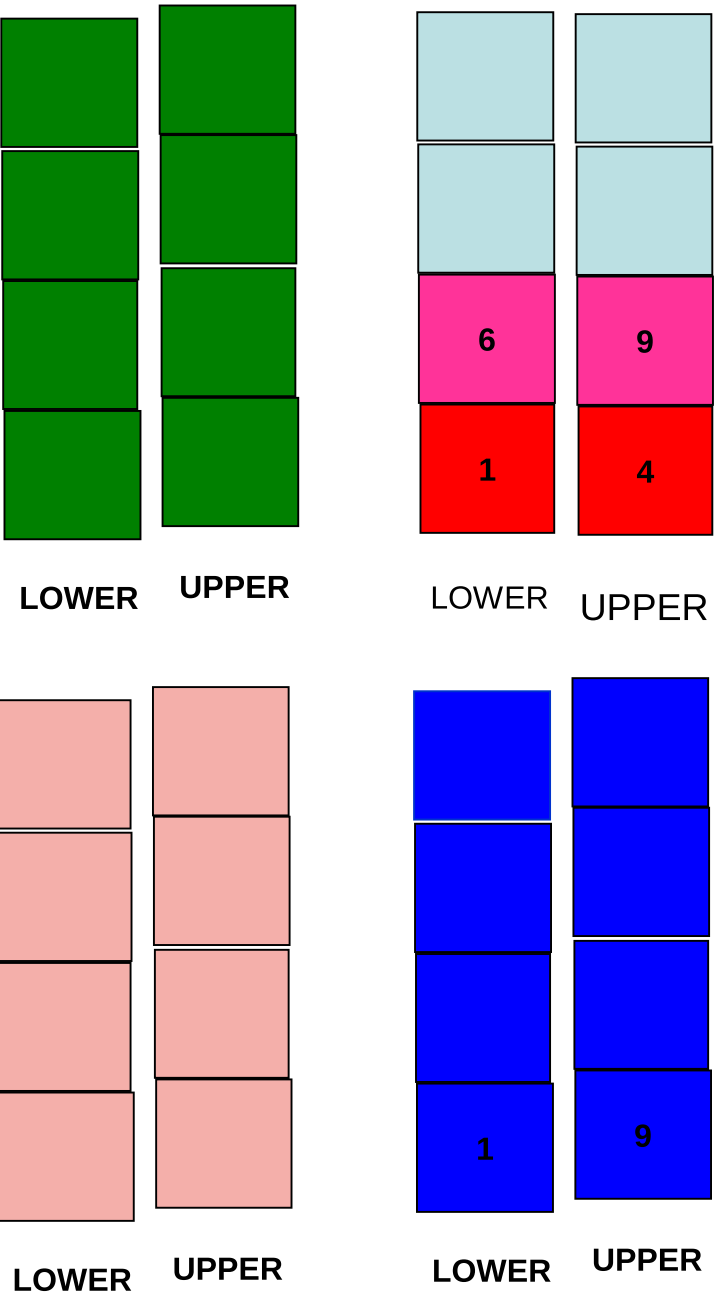
Lecture Outline

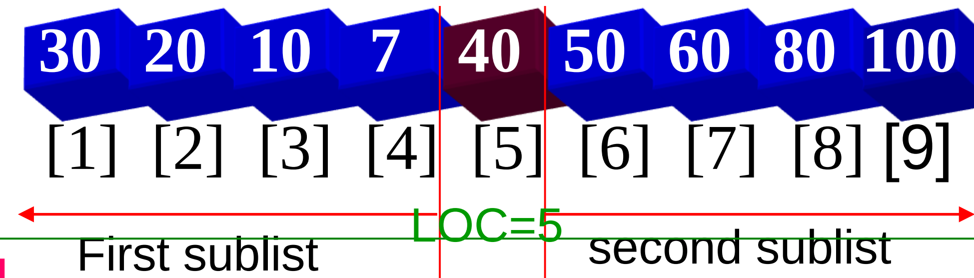
- Quick sort Algorithm
- Recursion
  - Calculate n factorial
  - Fibonacci Sequence
  - TOWERS OF HANOI

Quick sort Algorithm

This algorithm sorts an array A with N elements

1. [Initialize] TOP=NULL.
2. [Push boundary values of A onto stack when A has 2 or more elements]
   If N>1, then TOP:=TOP+1, LOWER[1]:=1 and UPPER[1]:=N.
3. Repeat Step 4 to 7 while TOP!= NULL.
4. [Pop sub list from stack]
   Set BEG:=LOWER[TOP], END:=UPPER[TOP]
   TOP:=TOP-1.
5. Call QUICK(A,N,BEG,END,LOC). [Procedure 6.5]
6. [Push left sub list onto stacks when it has 2 or more elements]
   If BEG<LOC-1 then:
   TOP:=TOP+1, LOWER[TOP]:=BEG,
   UPPER[TOP]=LOC-1
   [End of If structure].
7. [Push right sub list onto stacks when it has 2 or more elements]
   If LOC+1 < END then:
   TOP:=TOP+1, LOWER[TOP]:= LOC+1,
   UPPER[TOP]:= END
   [End of If structure]
   [End of Step 3 loop].
8. Exit





TOP:=NULL

9>1 ? Yes TOP:=TOP+1=1, LOWER[1]:=1, UPPER[1]:=9

Repeat TOP=!NULL.

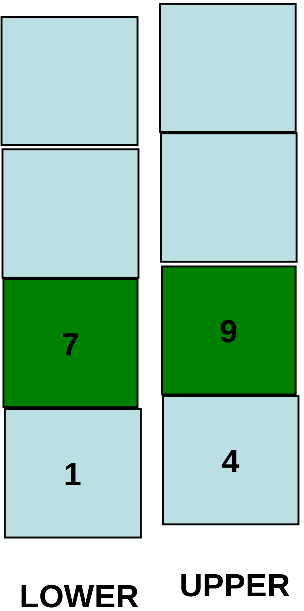
Pop sub list from STACKs
Set BEG:=LOWER[TOP]=1, END:=UPPER[TOP] =9
TOP:=TOP-1.=0

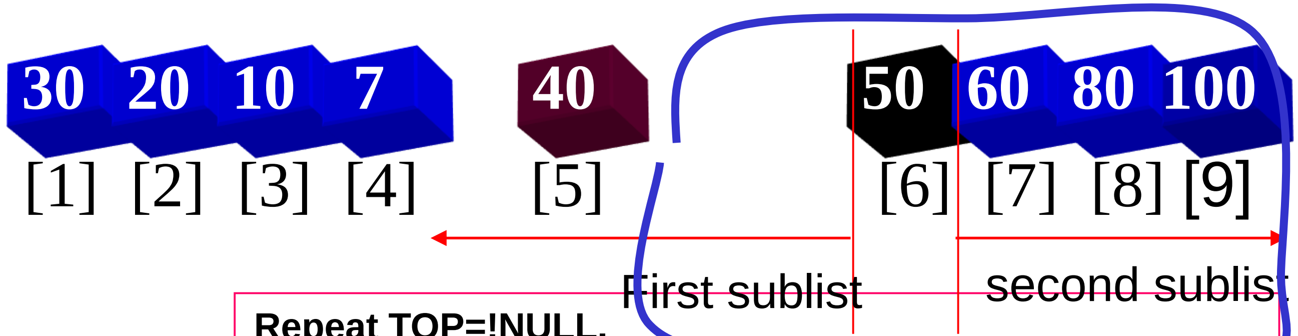
Call QUICK(A,N,1,9,LOC).
LOC=5

[Push left sub list onto stacks when it has 2 or more elements]
If BEG(1)<LOC-1(4) Yes [Left sublist's ending position=loc-1]
TOP:=TOP+1=1, LOWER[TOP]:=1,
UPPER[TOP]=LOC-1=5-1=4

[Push right sub list onto stacks when it has 2 or more elements]
If LOC+1(6) < END(9) ? yes [Right sublist's starting position=loc+1
TOP:=TOP+1=1+1=2, LOWER[TOP]:= LOC+1=6,
UPPER[TOP]:= END=9

Prepared by, Jesmin Akhter,
Lecturer, IIT,JU





Repeat TOP=!NULL.

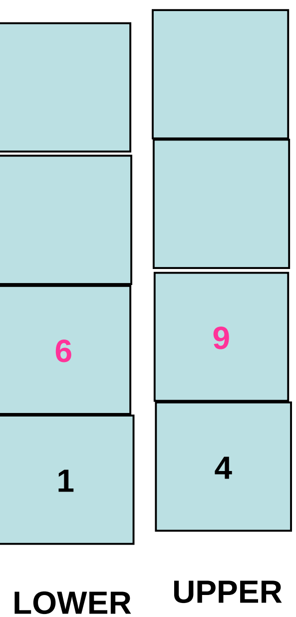
Pop sub list from STACKs LOC=6

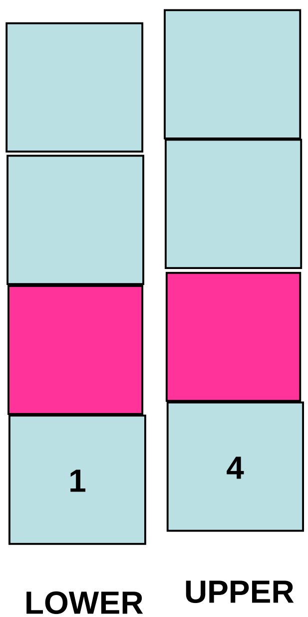
Set BEG:=LOWER[TOP]=6, END:=UPPER[TOP] =9
TOP:=TOP-1.=2-1=1

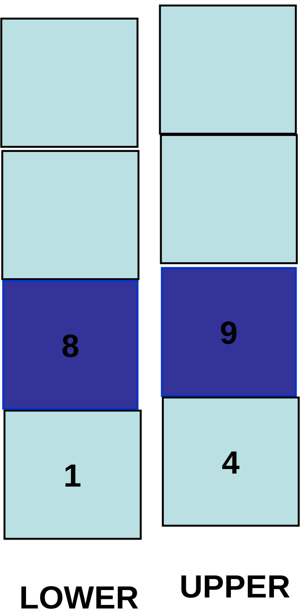
Call QUICK(A,N,6,9,LOC).
LOC=6

[Push left sub list onto stacks when it has 2 or more elements]
If BEG(6)<LOC-1(5) ? No

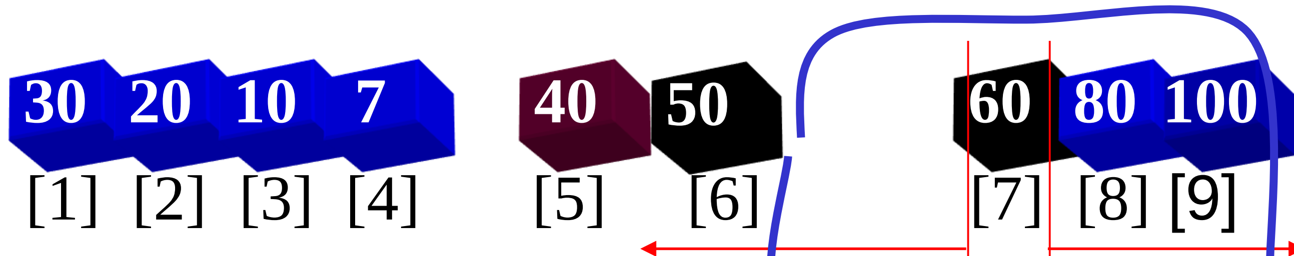
[Push right sub list onto stacks when it has 2 or more elements]
If LOC+1(7) < END(9) ? yes
TOP:=TOP+1=1+1=2, LOWER[TOP]:= LOC+1=7, UPPER[TOP]:= END=9







TOP=2



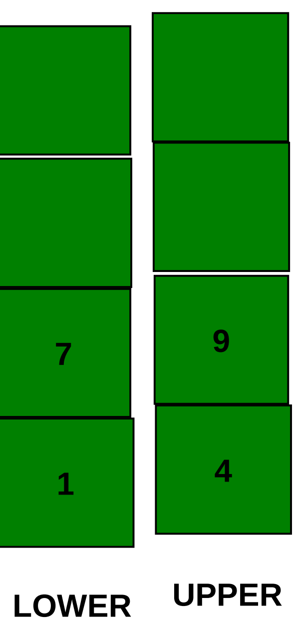
Repeat TOP=!NULL? No First sublist LOC=7 second s

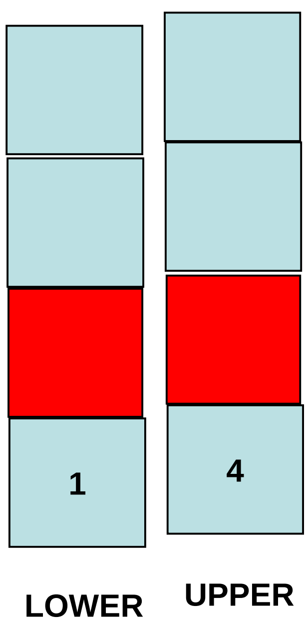
Pop sub list from STACKs
Set BEG:=LOWER[TOP]=7, END:=UPPER[TOP] =9
TOP:=TOP-1.=2-1=1

Call QUICK(A,N,7,9,LOC).
LOC=7

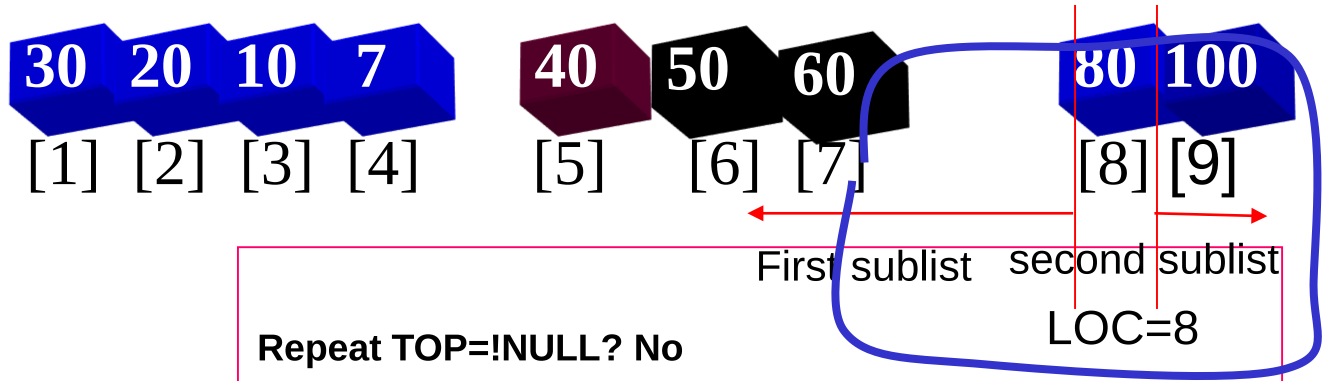
[Push left sub list onto stacks when it has 2 or more elements]
If BEG(7)<LOC-1(6) ? No

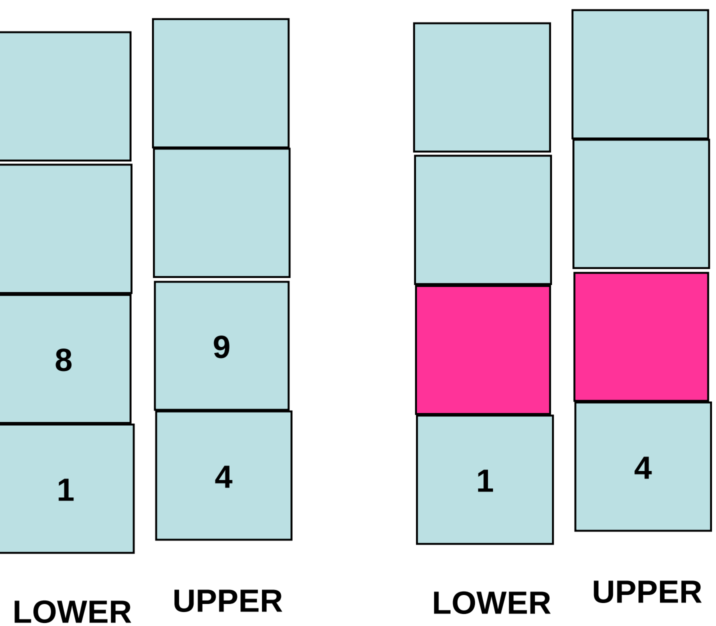
[Push right sub list onto stacks when it has 2 or more elements]
If LOC+1(8) < END(9) ? yes
TOP:=TOP+1=1+1=2, LOWER[TOP]:= LOC+1=8, UPPER[TOP]:= END=9





TOP=1





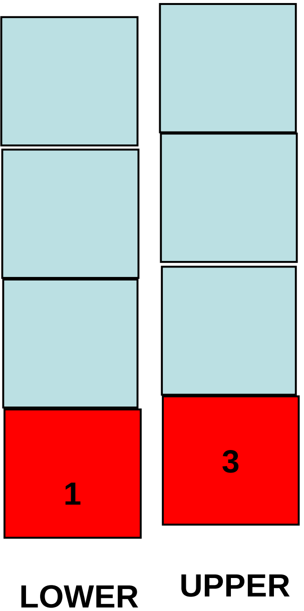
Repeat TOP=!NULL? No

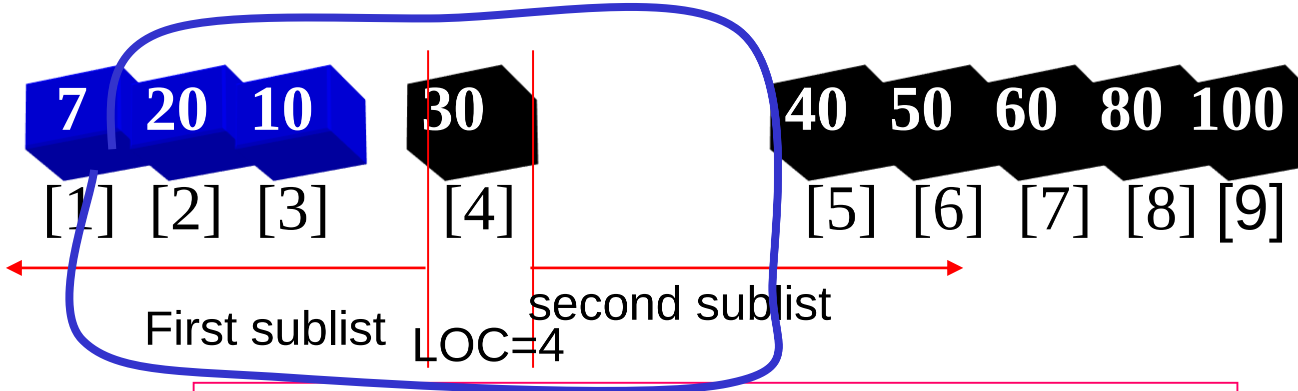
Pop sub list from STACKs
Set BEG:=LOWER[TOP]=8, END:=UPPER[TOP] =9
TOP:=TOP-1.=2-1=1

Call QUICK(A,N,8,9,LOC).
LOC=8

[Push left sub list onto stacks when it has 2 or more elements]
If BEG(8)<LOC-1(7) ? No

[Push right sub list onto stacks when it has 2 or more elements]
If LOC+1(9) < END(9) ? No



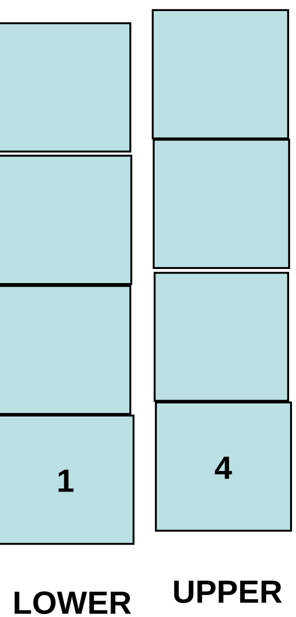


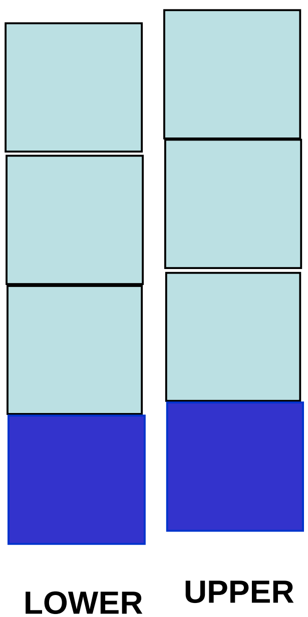
Repeat TOP=!NULL? No

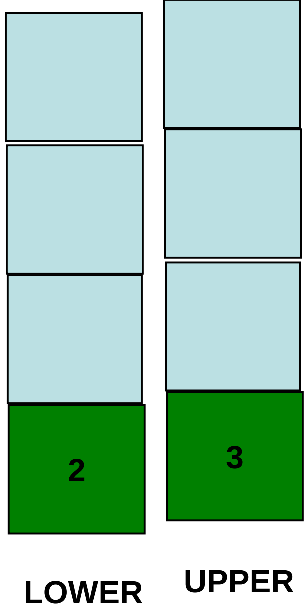
Pop sub list from STACKs
Set BEG:=LOWER[TOP]=1, END:=UPPER[TOP] =4
TOP:=TOP-1.=1-1=0

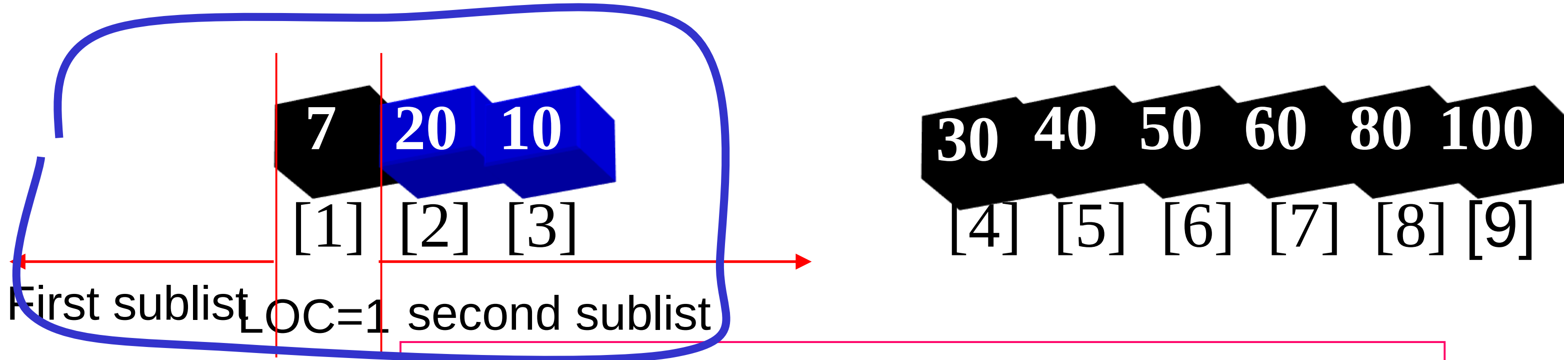
Call QUICK(A,N,1,4,LOC).
LOC=4

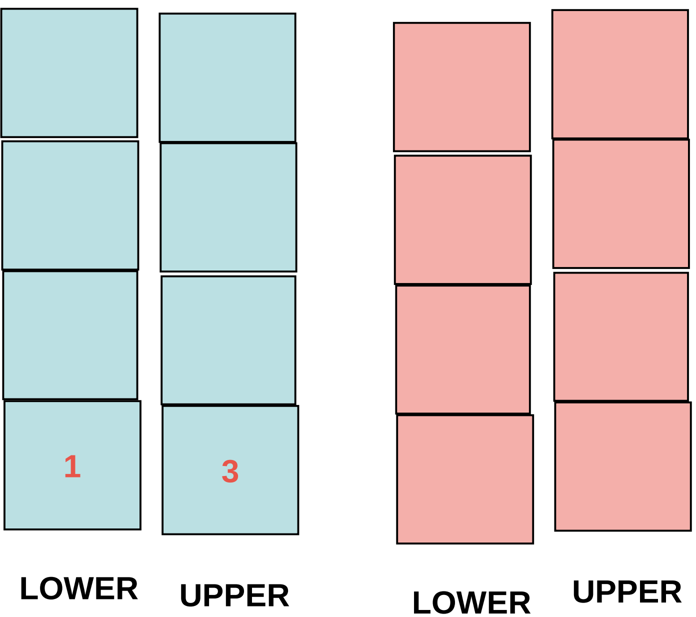
[Push left sub list onto stacks when it has 2 or more elements]
If BEG(1)<LOC-1(3) ? Yes
TOP:=TOP+1=1, LOWER[TOP]:=1,
UPPER[TOP]=LOC-1=4-1=3
[Push right sub list onto stacks when it has 2 or more elements]
If LOC+1(5) < END(4) ? No











Repeat TOP=!NULL? No

Pop sub list from STACKs

Set BEG:=LOWER[TOP]=1, END:=UPPER[TOP] =3

TOP:=TOP-1.=1-1=0

Call QUICK(A,N,1,3,LOC).

LOC=1

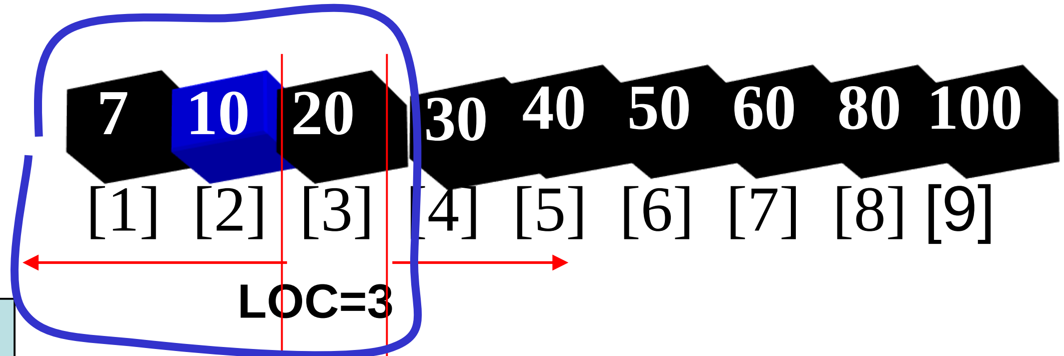
[Push left sub list onto stacks when it has 2 or more elements]

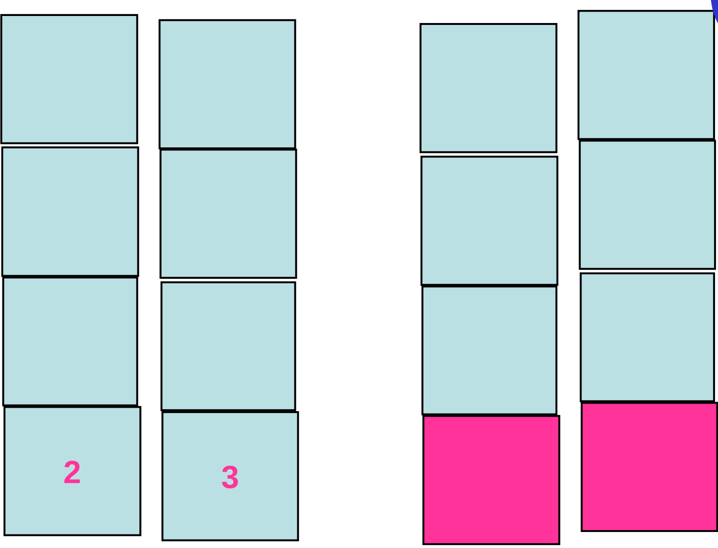
If BEG(1)<LOC-1(0) ? No

[Push right sub list onto stacks when it has 2 or more elements]

If LOC+1(2) < END(3) ? Yes

TOP:=TOP+1=0+1=1, LOWER[TOP]:= LOC+1=2, UPPER[TOP]:= END=3





LOWER UPPER LOWER UPPER

Repeat TOP=!NULL? No

Repeat TOP=!NULL? Yes

Pop sub list from STACKs

Set BEG:=LOWER[TOP]=2, END:=UPPER[TOP] =3

TOP:=TOP-1.=1-1=0

Call QUICK(A,N,2,3,LOC).

LOC=3

[Push left sub list onto stacks when it has 2 or more elements]

If BEG(2)<LOC-1(2) ? No

[Push right sub list onto stacks when it has 2 or more elements]

If LOC+1(4) < END(3) ? No

Recursion

- A function is said to be recursively defined, if a function containing either a Call statement to itself or a Call statement to a second function that may eventually result in a Call statement back to the original function.
- A recursive function must have the following properties:

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

  2. Each time the function does call itself (directly or indirectly), the argument of the function must be closer to a base value

Example 1:

- Factorial function: In general, we can express the factorial function as follows: $n! = n * (n-1)!$
- The factorial function is only defined for positive integers.
- if $n<=1$, then $n! = 1$
- if $n>1$, then $n! = n * (n-1)!$

Calculate n factorial

Procedure 6.7A: FACTORIAL(FACT, N)

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

1. If N=0, then: Set FACT :=1, and Return
2. Set FACT = 1.
3. Repeat for K=1 to N
   Set FACT : = K*FACT
4. Return.

Procedure 6.7B: FACTORIAL(N)

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

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

FACTORIAL(FACT, N)

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

factorial function

- Assume the number typed is 3, that is, numb=3.

```
fac(3) :
 3 <= 1 ?                  No.
 fac(3) = 3 * fac(2)
   fac(2) :
      2 <= 1 ?             No.
      fac(2) = 2 * fac(1)
          fac(1) :
             1 <= 1 ?      Yes.
             return 1
     fac(2) = 2 * 1 = 2
     return fac(2)
fac(3) = 3 * 2 = 6
return fac(3)
fac(3)  has the value 6
```

```
int fac(int numb){
  if (numb<=1)
     return 1;
  else
     return numb * fac(numb-1);
}
```

Example 2

Fibonacci Sequence : 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,…………………

Definition 6.2

(a) If n=0 or n=1, then Fn =n

(b) If n>1, then Fn =Fn-2 +Fn-1.

Procedure 6.8 : FIBONACCI(FIB, N)

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

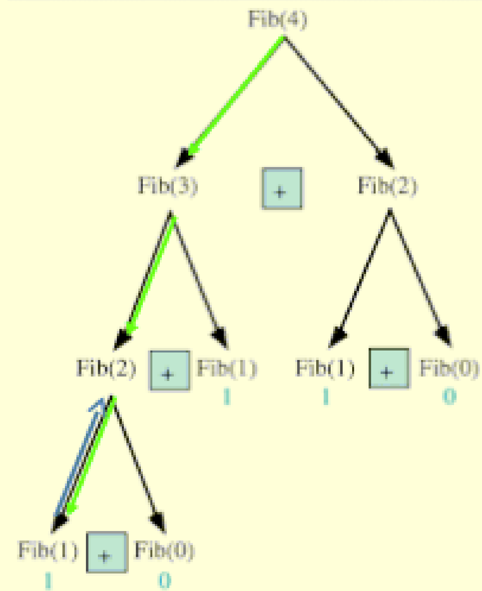
1. If N=0 or N=1, then : set FIB :=N, and return.
2. Call FIBONACCI(FIBA, N-2).
3. Call FIBONACCI(FIBB, N-1).
4. Set FIB : =FIBA +FIBB.
5. Return.

Trace a Fibonacci Number

- Assume the input number is 4, that is, num=4:

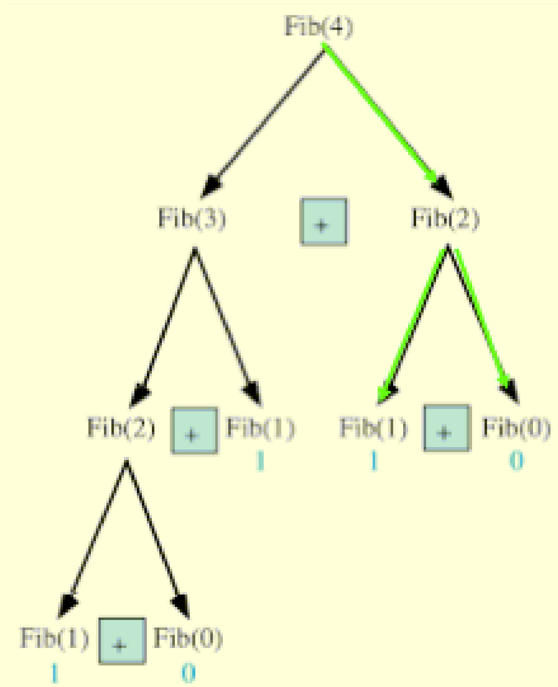
```
fib(4):
    4 == 0 ? No;    4 == 1? No.
    fib(4) = fib(3) + fib(2)
    fib(3):
        3 == 0 ? No; 3 == 1? No.
        fib(3) = fib(2) + fib(1)
        fib(2):
            2 == 0? No; 2==1? No.
            fib(2) = fib(1)+fib(0)
            fib(1):
                1== 0 ? No; 1 == 1? Yes.
                 fib(1) = 1;
              return fib(1);
```

```
int fib(int num)
{
   if (num == 0) return 0;
   if (num == 1) return 1;
   return
     (fib(num-1)+fib(num-2));
}
```



Trace a Fibonacci Number

```
fib(2):
    2 == 0 ? No; 2 == 1?   No.
    fib(2) = fib(1) + fib(0)
    fib(1):
      1== 0 ? No; 1 == 1?  Yes.
       fib(1) = 1;
       return fib(1);
     fib(0):
       0 == 0 ?   Yes.
       fib(0) = 0;
       return fib(0);
     fib(2) = 1 + 0 = 1;
     return fib(2);
  fib(4) = fib(3) + fib(2)
         = 2 + 1 = 3;
  return fib(4);
```

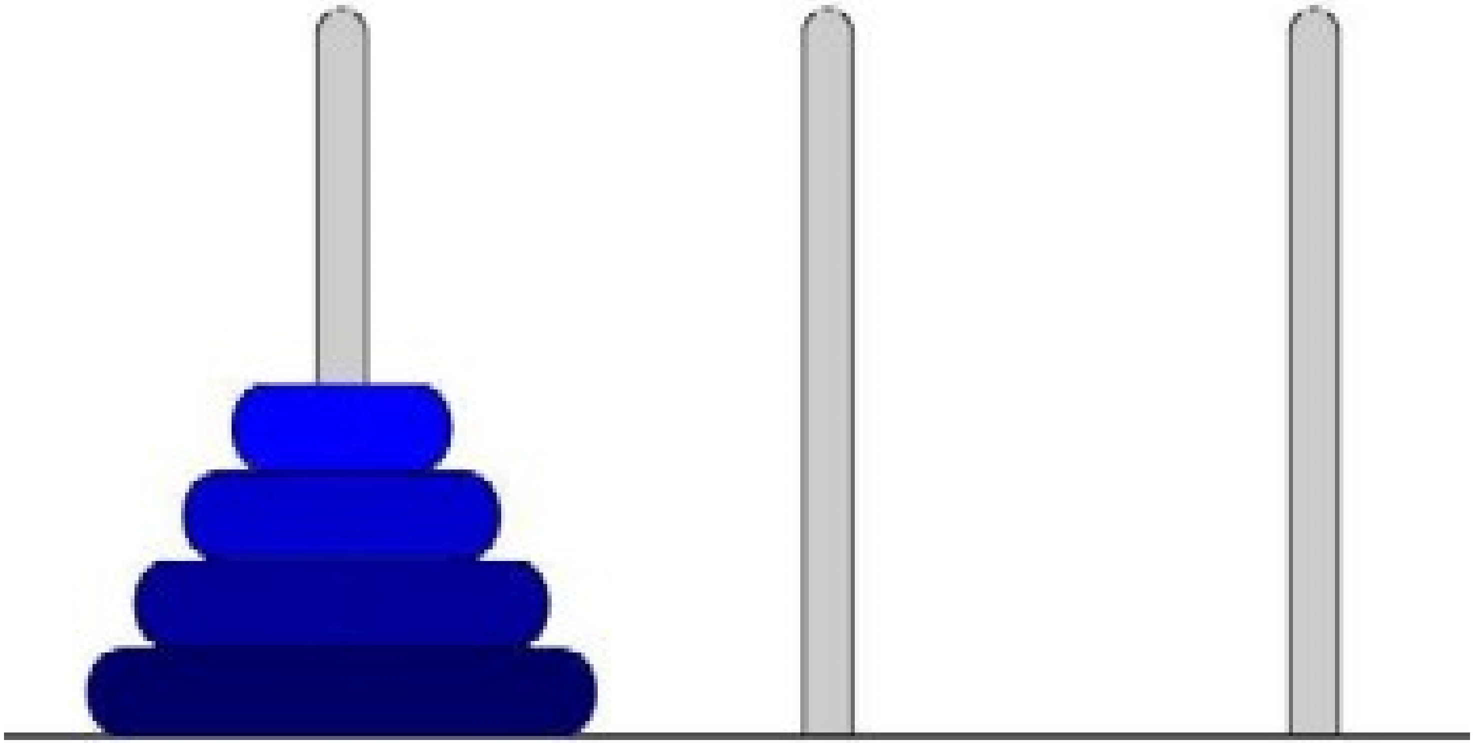


Fibonacci number w/o recursion

```
//Calculate Fibonacci numbers iteratively
//much more efficient than recursive solution

int fib(int n)
{
  int f[n+1];
  f[0] = 0; f[1] = 1;
  for (int i=2; i<= n; i++)
      f[i] = f[i-1] + f[i-2];
  return f[n];
}
```

TOWERS OF HANOI



Prepared by, Jesmin Akhter,
Lecturer, IIT,JU

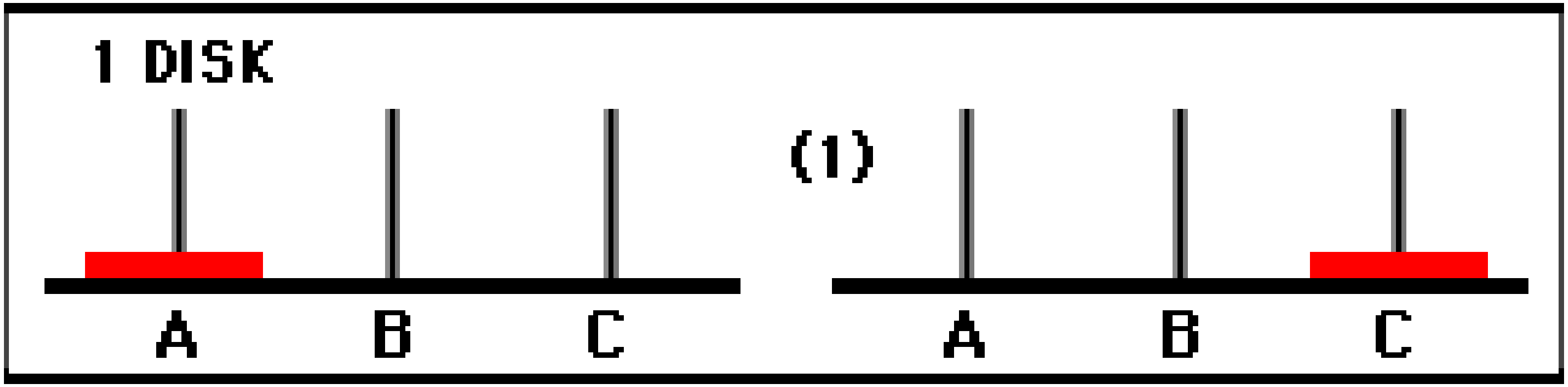
TOWERS OF HANOI

- Disks of different sizes (call the number of disks "n") are placed on the left hand post,
- arranged by size with the smallest on top.
- You are to transfer all the disks to the right hand post in the fewest possible moves, without ever placing a larger disk on a smaller one.

The object is to move all the disks over to another pole. But you cannot place a larger disk onto a smaller disk.

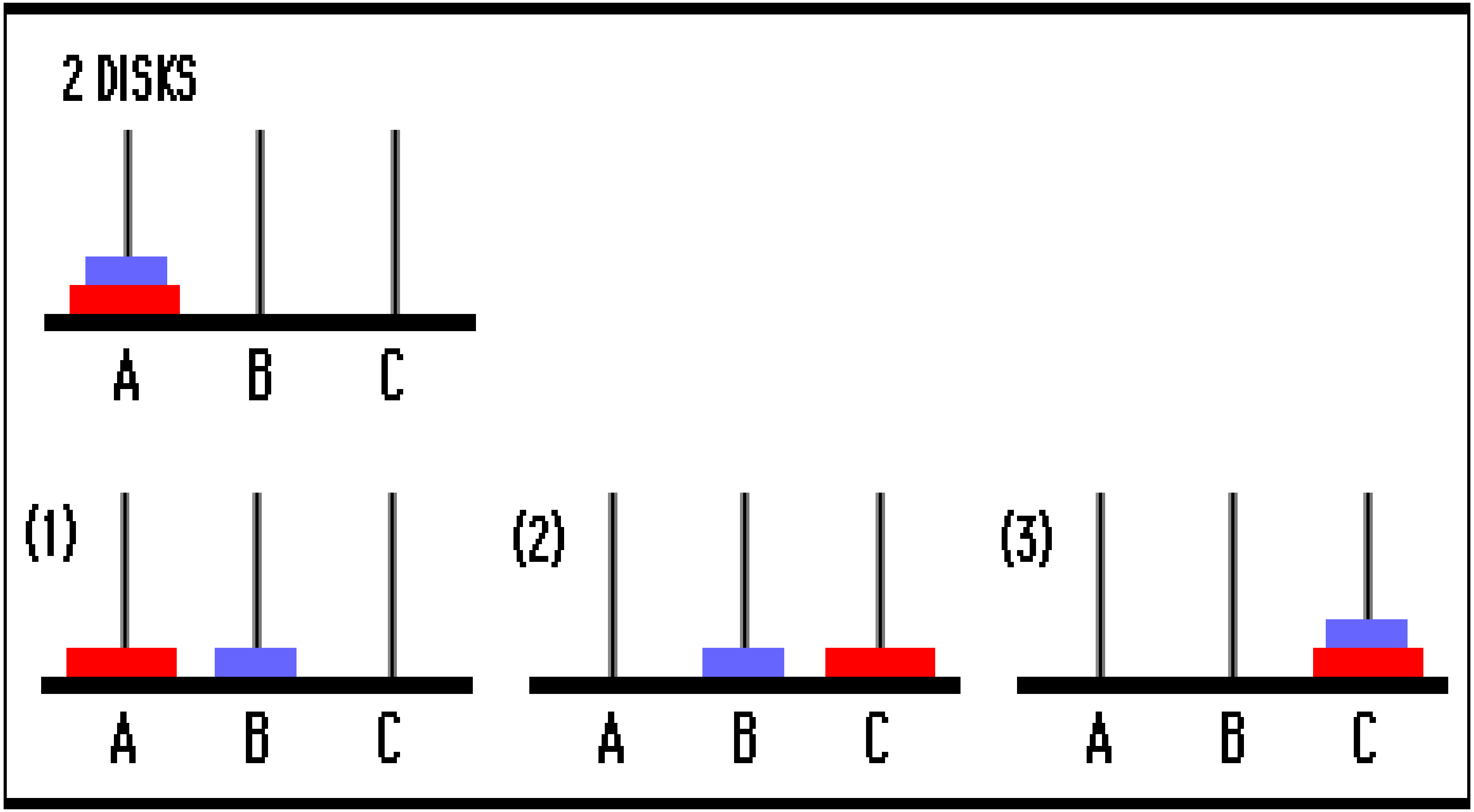
TOWERS OF HANOI

- **How many moves will it take to transfer n disks from the left post to the right post?**
- Let's look for a pattern in the number of steps it takes to move just one, two, or three disks. We'll number the disks starting with disk 1 on the bottom.**1 disk: 1 move**
- Move 1: move disk 1 to post C



2 disks: 3 moves

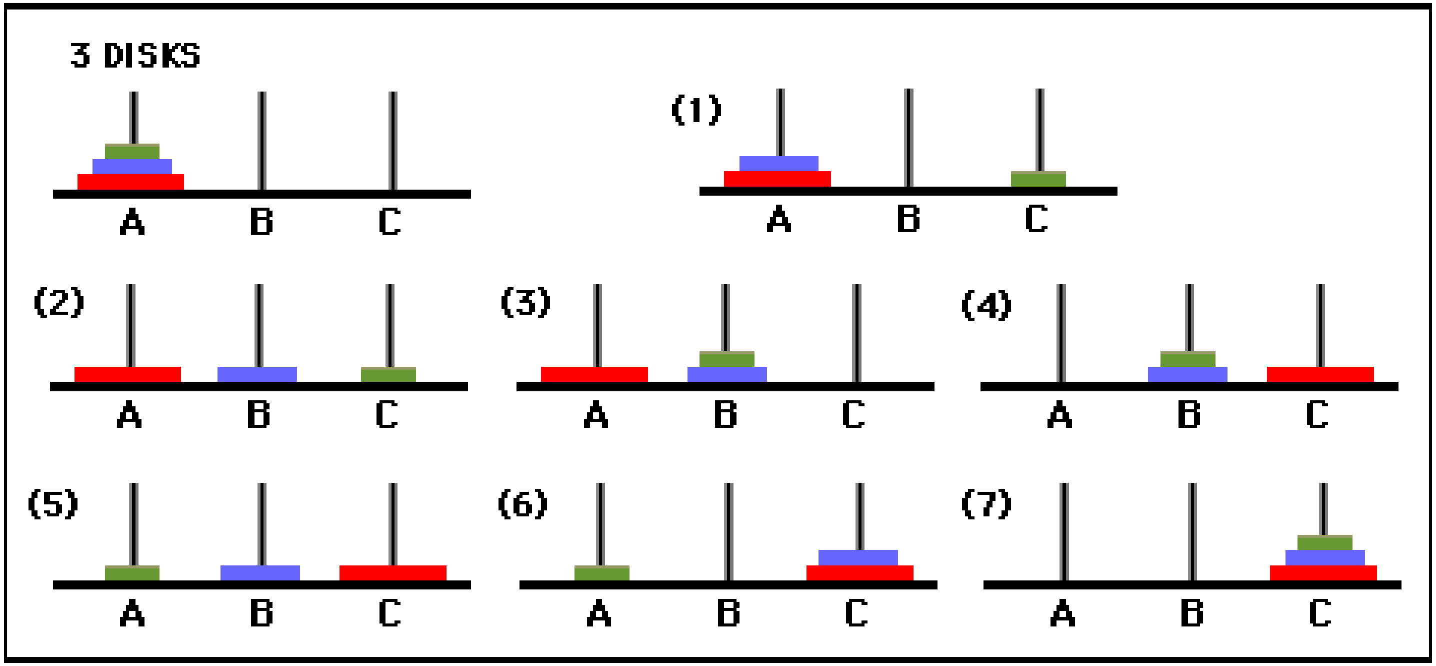
Move 1: move disk 2 to post B
Move 2: move disk 1 to post C
Move 3: move disk 2 to post C



Prepared by, Jesmin Akhter,
Lecturer, IIT,JU

3 disks: 7 moves

Move 1: move disk 3 to post C
Move 2: move disk 2 to post B
Move 3: move disk 3 to post B
Move 4: move disk 1 to post C
Move 5: move disk 3 to post A
Move 6: move disk 2 to post C
Move 7: move disk 3 to post C



TOWERS OF HANOI

- **Its solution touches on two important topics :**

- A. recursive functions and stacks
- B. recurrence relations

- **B. Recurrence relations**
- Let TN be the minimum number of moves needed to solve the puzzle with N disks. From the previous section T1 = 1, T2 = 3 and T3 = 7
- A trained mathematician would also note that T0 = 0. Now let us try to derive a general formula.

- Thus we can define the quantity TN as
- $T_0 = 0$
  $T_N = 2T_{N-1} + 1$ for $N > 0$ We may compute
- $T_1 = 2T_0 + 1 = 1$,
- $T_2 = 2T_1 + 1= 3$,
- $T_3 = 2T_2 + 1 = 7$
- $T_4 = 2T_3 + 1 = 15$ and so on sequentially.

TOWERS OF HANOI

- **A. Recursive pattern**
- From the moves necessary to transfer one, two, and three disks, we can find a *recursive pattern* - a pattern that uses information from one step to find the next step - for moving n disks from post A to post C:
- First, transfer n-1 disks from post A to post B. The number of moves will be the same as those needed to transfer n-1 disks from post A to post C. Call this number M moves. [As you can see above, with three disks it takes 3 moves to transfer two disks (n-1) from post A to post C.]
- Next, transfer disk 1 to post C [1 move].
- Finally, transfer the remaining n-1 disks from post B to post C. [Again, the number of moves will be the same as those needed to transfer n-1 disks from post A to post C, or M moves.]

for **1 disk** it takes 1 move to transfer 1 disk from post A to post C;

for **2 disks**, it will take 3 moves: 2M + 1 = 2(**1**) + 1 = **3**

for **3 disks**, it will take 7 moves: 2M + 1 = 2(**3**) + 1 = **7**

for **4 disks**, it will take 15 moves: 2M + 1 = 2(**7**) + 1 = **15**

for **5 disks**, it will take 31 moves: 2M + 1 = 2(**15**) + 1 = **31**

for **6 disks**... ?

Procedure6.9

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 $\longrightarrow$ 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 $\longrightarrow$ END
4. [Move N-1 disks from peg AUX to peg END]
   call TOWER(N-1, AUX, BEG, END)
5. Return.

Applications

- The Tower of Hanoi is frequently used in psychological research on problem solving. There also exists a variant of this task called Tower of London for neuropsychological diagnosis and treatment of executive functions.
- The Tower of Hanoi is also used as Backup rotation scheme when performing computer data Backups where multiple tapes/media are involved.
- As mentioned above, the Tower of Hanoi is popular for teaching recursive algorithms to beginning programming students. A pictorial version of this puzzle is programmed into the emacs editor, accessed by typing M-x hanoi. There is also a sample algorithm written in Prolog.
- The Tower of Hanoi is also used as a test by neuropsychologists trying to evaluate frontal lobe (The *frontal lobes* are considered our emotional control center and home to our personality.)deficits.