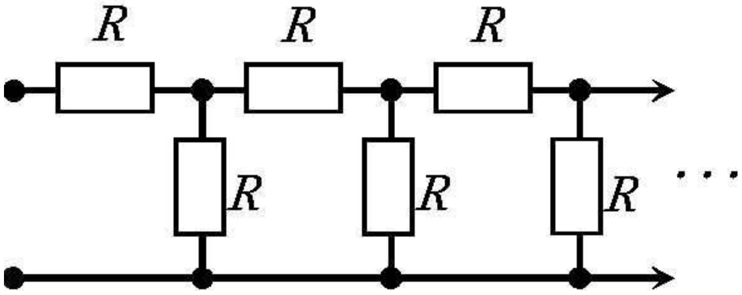
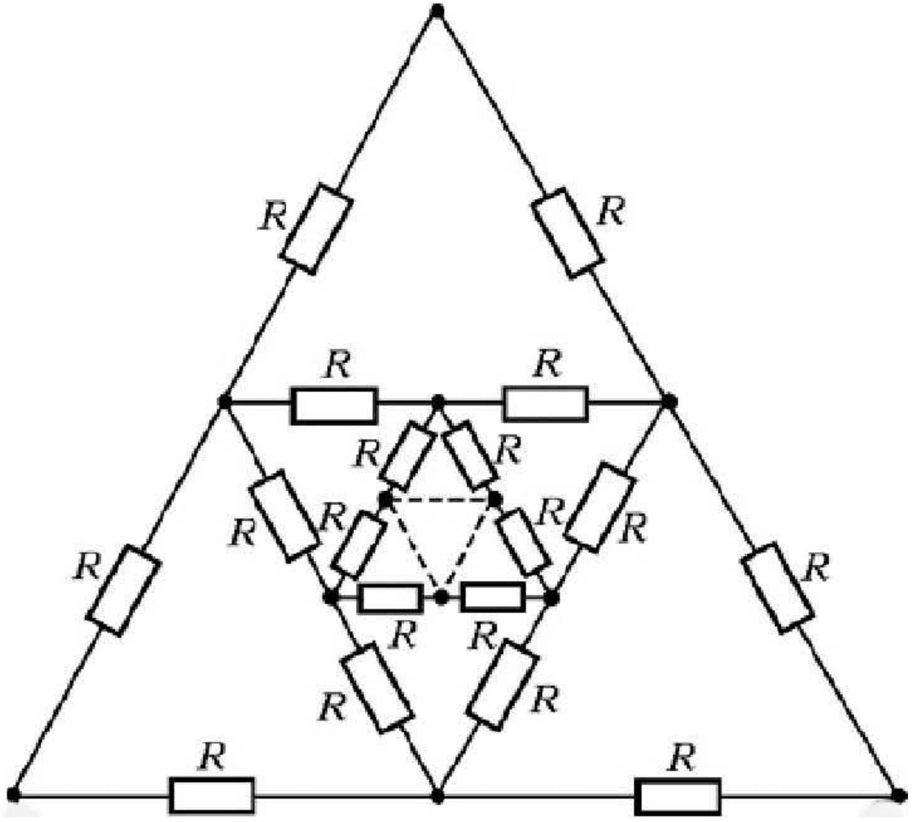
Recursion

February 7, 2018





Mid-Sem Lab Exam: Date and Time

- Lab mid sem on Sunday, February 11
  - Sections A1-A6 11am-2pm
  - Sections A7-A13 230pm-530pm

Mid-Sem Lab Exam: Syllabus

- Up to Lecture 15
  - Datatypes
  - Operators, expressions, statements, I/O
  - Conditional statements
  - Loops
  - Functions

Mid-Sem Lab Exam: Room Assignment

Lab	Sections	Time
CC-02 (NCL)	A1,A2,A3	11am – 2pm
CC-01 (NCL)	A4	11am – 2pm
CC-03 (NCL)	A5	11am – 2pm
Linux Lab 3 (CC)	A6	11am – 2pm

Lab	Sections	Time
CC-02 (NCL)	A9, A10, A11	2:30pm-5:30pm
CC-01 (NCL)	A7	2:30pm-5:30pm
CC-03 (NCL)	A8	2:30pm-5:30pm
Linux Lab 3 (CC)	A12, A13	2:30pm-5:30pm

Tips for Lab Exam

- Read all instructions and the questions carefully.
- Approaching a problem:
  - Don't try to write the entire program in one attempt.
  - Write short segments of code and keep compiling whenever possible to ensure correctness.
  - Use functions to divide your program into smaller components.
  - Write functions to perform ONE task at a time. Combine the tasks in main.

Tips for Lab Exam

- If your program is not working
  - insert dummy printf statements to test where it is going wrong.
    - Specially test input/output of the functions you write.
  - make a variable table (especially when using loops).
  - Debug ONE iteration of a loop at a time.
- Important: Coding Style
  - Write comments to explain your code.
  - Give meaningful variable names.
  - Use proper indentation.
- Do not rely solely on the test cases provided by us.

Conduct for Lab Exam

- Lab exam will be conducted through esc101.cse.iitk.ac.in
- The network will be ON during the exam
- Closed book exam
  - You are **not** allowed to access any other site during the exam
  - Do **not** check emails (@iitk, @gmail, …)
  - Mobiles are **not** allowed on person (even if switched off)
- ITS stores your *keystrokes and program development steps!*

Conduct for Lab Exam

- We will monitor system usage
  - Tracking IP used for submission and incoming and outgoing network traffic
  - Manually and automatically
- Please don’t cheat
  - It’ll make life unpleasant for all concerned

```
1   #include <stdio.h>
2   int max(int a, int b) {
3      if (a > b)
4          return a;
5     else
6          return b;
7   }

8   int main () {
9         int x;     10a
10        x =        max(6, 4);
11        printf("%d",x);
12        return 0;
13  }
```

- Steps when a function is called: max(6,4) in step 10a.
- Allocate space for (i) return value, (ii) store return address and (iii) pass parameters.

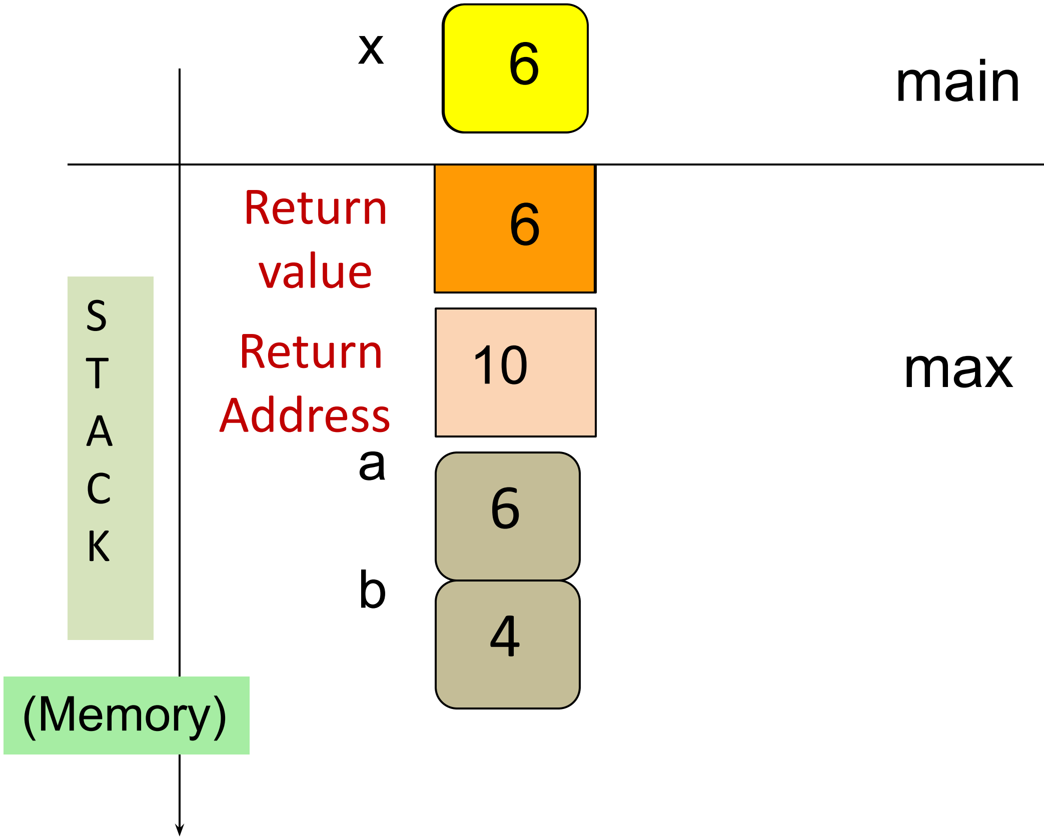
1. Create a box informally called ``Return value'' of same type as the return type of function.
2. Create a box and store the location of the next instruction in the calling function (main)—return address. Here it is 10 (why not 11?). Execution resumes from here once function terminates.
3. Parameter Passing- Create boxes for each formal parameter: a, b here. Initialize them using actual parameters, 6 and 4.

```
1   #include <stdio.h>
2   int max(int a, int b) {
3     if (a > b)
4       return a;
5     else
6       return b;
7   }

8   int main() {
9     int x = -1;
10    x = max(6, 4);
11    printf("%d",x);
12    return 0;
13  }
```

Calling max(6,4):

1. Allocate space for return value.
2. Store return address (10).
3. Pass parameters.



After completing max(), execution in main() will re-start from address 10.

Recursion

- A function defined by calling itself, *directly* or *indirectly,* is called a *recursive function.*
  - The phenomenon itself is called recursion
- Examples:
  - Factorial:

```
0! = 1
n! = n * (n-1)!
```

  - Even and Odd:

```
Even(n) = (n == 0) || Odd(n-1)
Odd(n)  = (n != 0) && Even(n-1)
```

Recursive Functions: Properties

- The arguments change between the recursive calls

5! = 5 * 4! = 5 * 4 * 3! = …

- Change is towards a case for which solution is known (base case)
- There must be one or more base cases

0! is 1

Odd(0) is false

Even(0) is true

Example: Factorial

- Write a program to print the factorial of an integer *n*
- Can do it using loops
  - To calculate the factorial of n, you calculate all the factorials in between

```
long int factorial(int n){
     int i;
     long int fact = 1;
     for (i=1; i<=n; i++){
        fact = fact * i;
    }
    return fact;
}
```

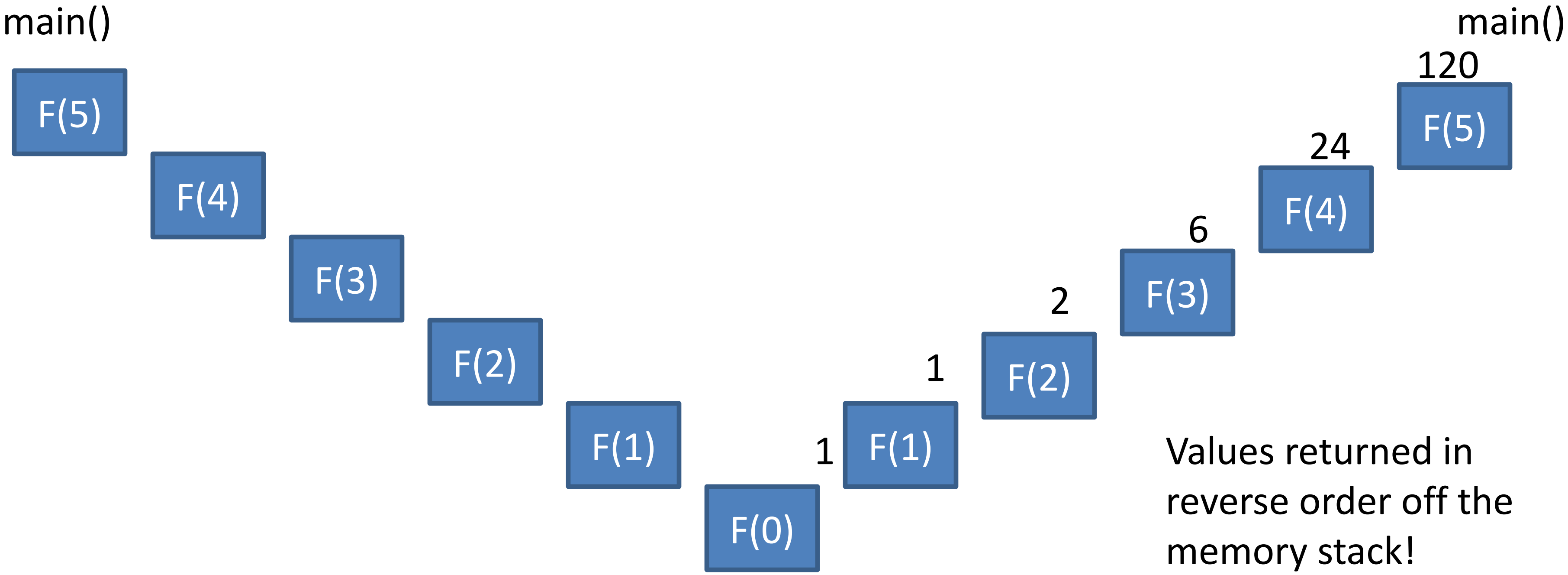
Example: Factorial

- Write a recursive program to print the factorial of an integer *n*
- Process
  - N! = N x (N-1)! *(recursion)*
  - 0! = 1 (*base case*)

```
long int factorial(int n){
    if(n==0)
        return 1;
    else
        return(n*factorial(n-1));
}
```

Tracing the flow of control

```
long int factorial(int n){
    if(n==0)
        return 1;
    else
        return(n*factorial(n-1));
}
```



main()

F(5)

F(4)

F(3)

F(2)

F(1)

F(0)

1

F(1)

1

F(2)

2

F(3)

6

F(4)

24

F(5)

120

main()

Values returned in reverse order off the memory stack!

Recursion and Induction

When programming recursively, think inductively

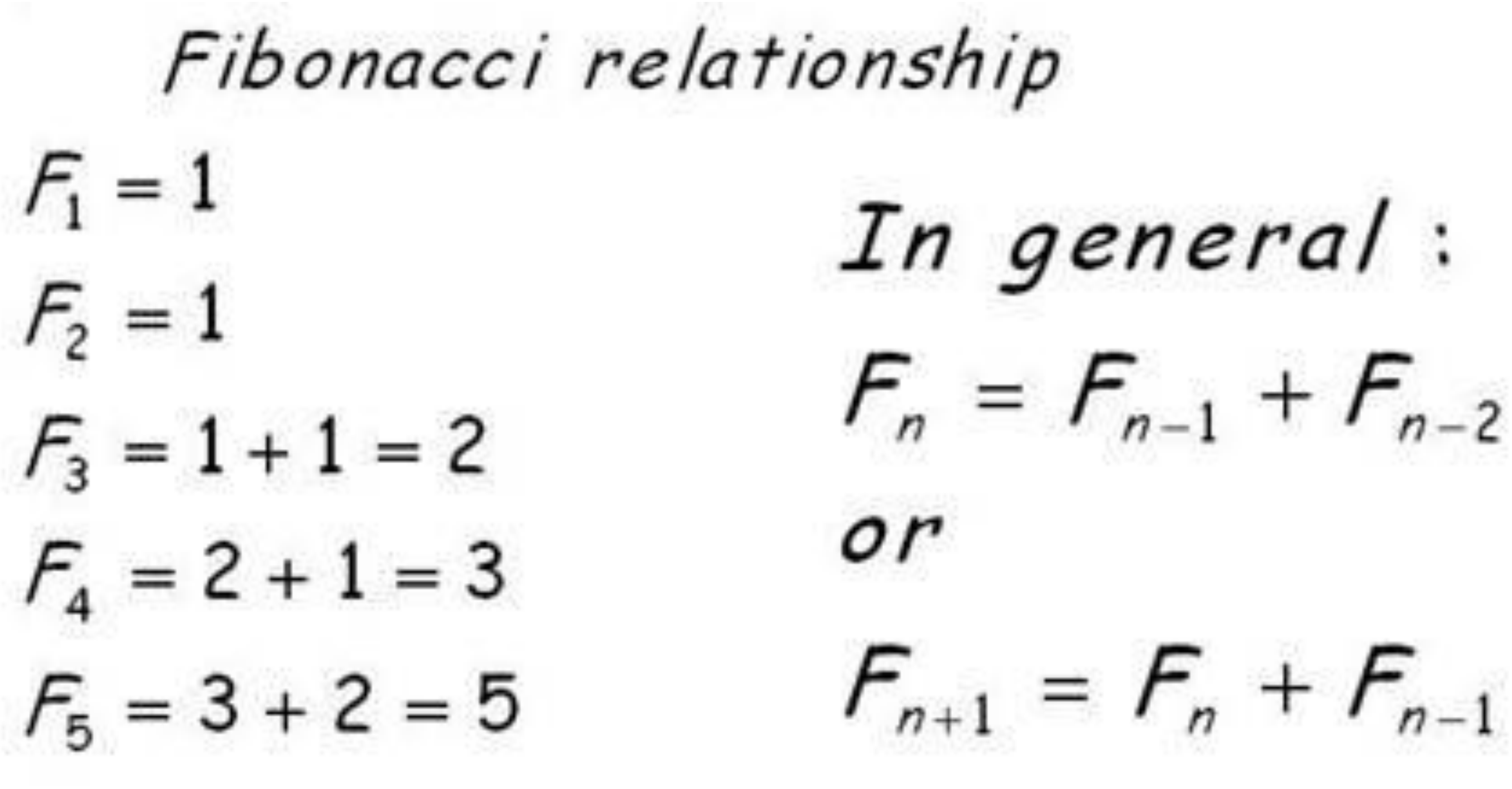
- Mathematical induction for the natural numbers
- Structural induction for other recursively-defined types (to be covered later!)

Recursion and Induction

When writing a recursive function

- Write down a clear, concise *specification* of its behavior
- Give an *inductive proof* that your code satisfies the specification

Fibonacci Numbers



Fibonacci relationship

$F_1 = 1$

$F_2 = 1$

$F_3 = 1 + 1 = 2$

$F_4 = 2 + 1 = 3$

$F_5 = 3 + 2 = 5$

In general :

$F_n = F_{n-1} + F_{n-2}$

or

$F_{n+1} = F_n + F_{n-1}$

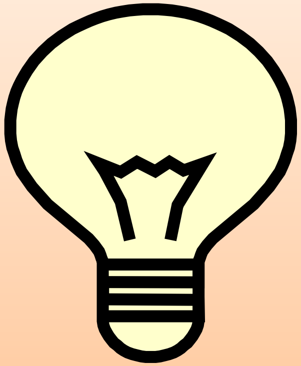
Recursion vs Iteration

```
int fib(int n)
{
   int first = 0, second = 1;
   int next, c;
   if (n <= 1)
       return n;
   for ( c = 1; c < n ; c++ ) {
       next = first + second;
       first = second;
       second = next;
    }
    return next;
}
```

The recursive program is closer to the definition and easier to read.

But very very inefficient

```
int fib(int n)
{
   if ( n <= 1 )
      return n;
   else
      return fib(n-1) + fib(n-2);
}
```



Recursive fib

fib(5)

fib(4) + fib(3)

fib(3) + fib(2)   fib(2) + fib(1)

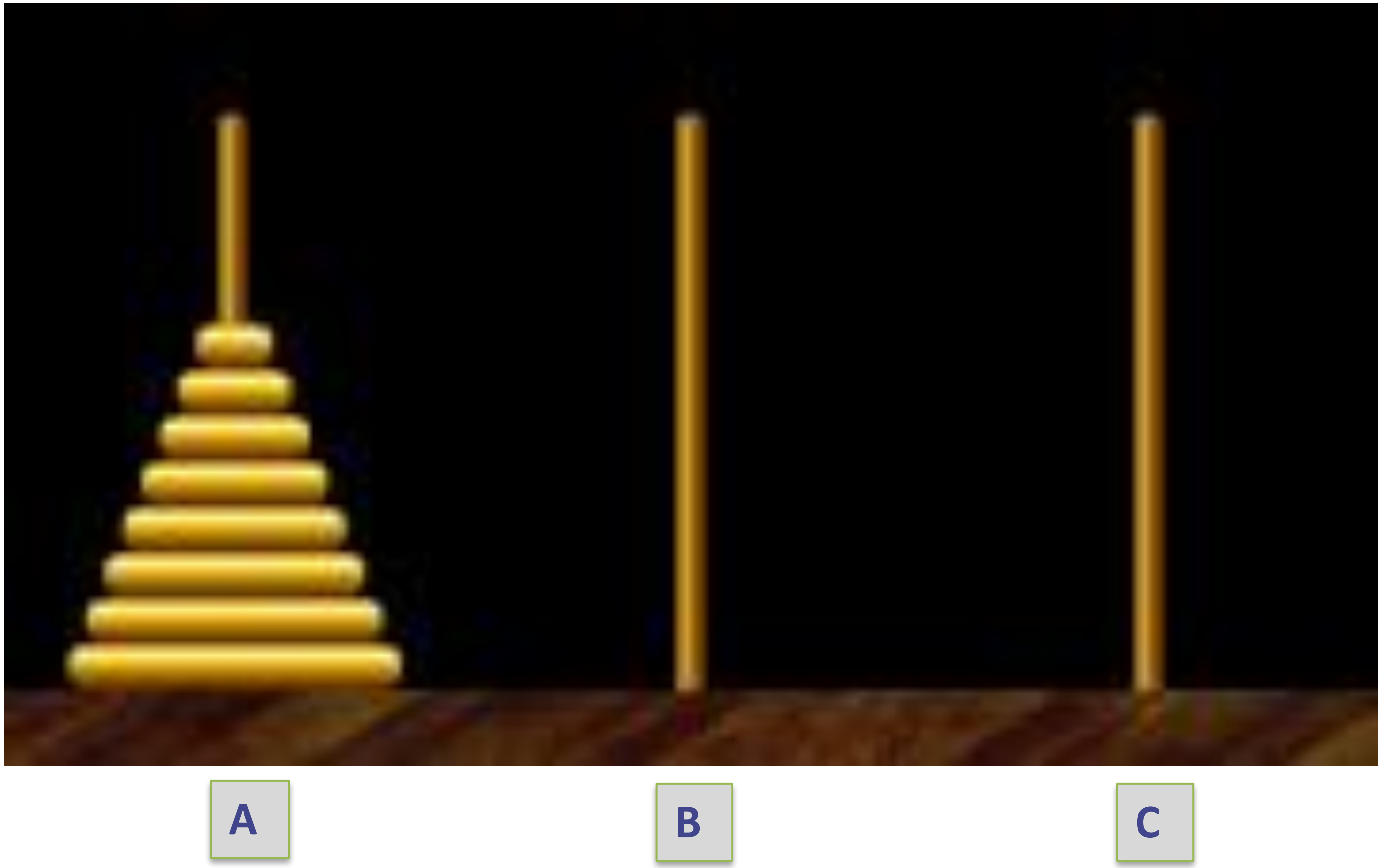
fib(2) + fib(1)   fib(1) + fib(0)   fib(1) + fib(0)

fib(1) + fib(0)

Recursion: Pros and Cons

- Pros
  - Very intuitive
  - Very compact in terms of code length
  - Very cool to look at
- Cons
  - Slower usually
  - Stack overflow
  - Harder to debug

Recursion : Tower of Hanoi



No disk may be placed on top of a smaller disk.

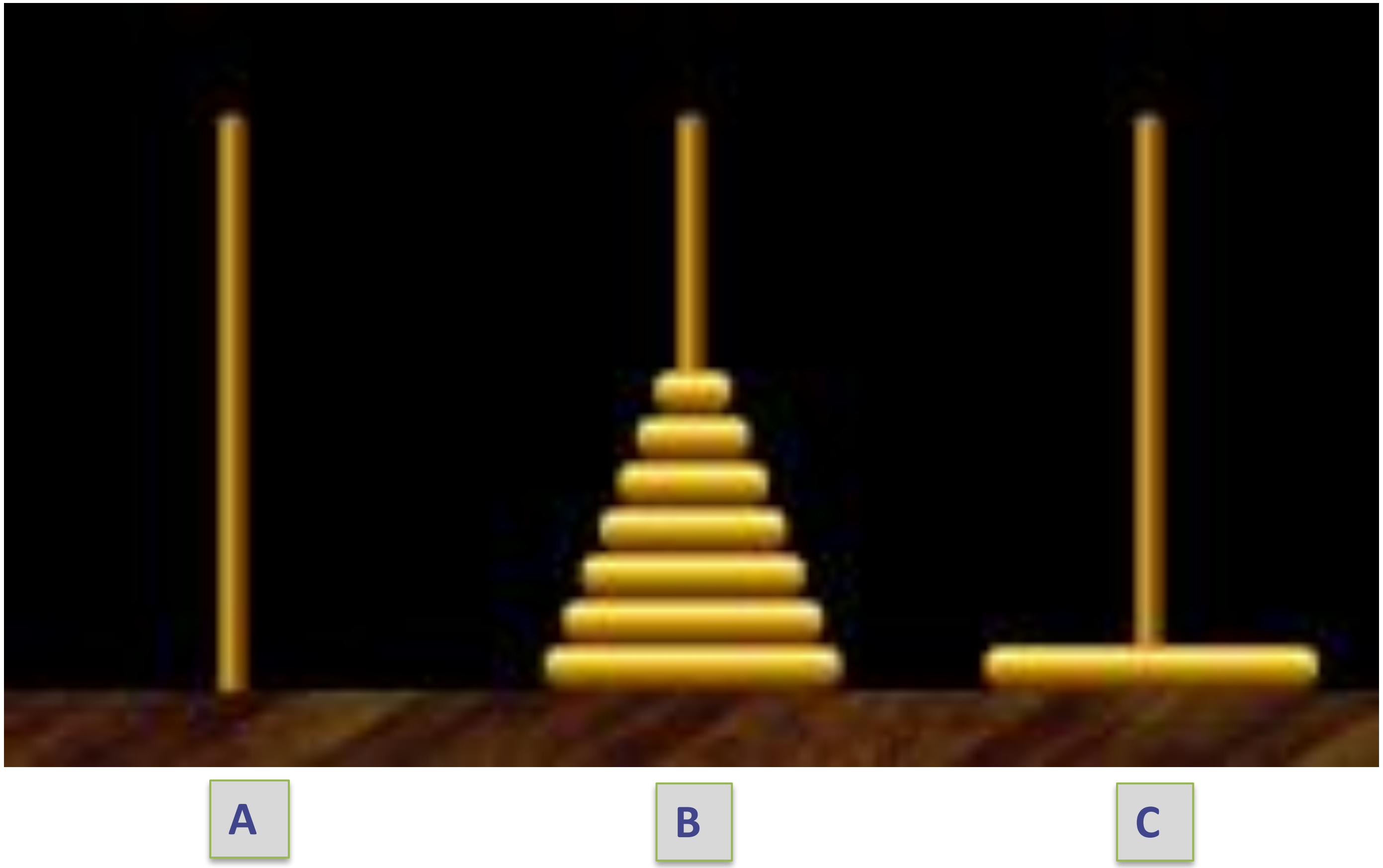
Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

Recursion : Tower of Hanoi ..2

No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

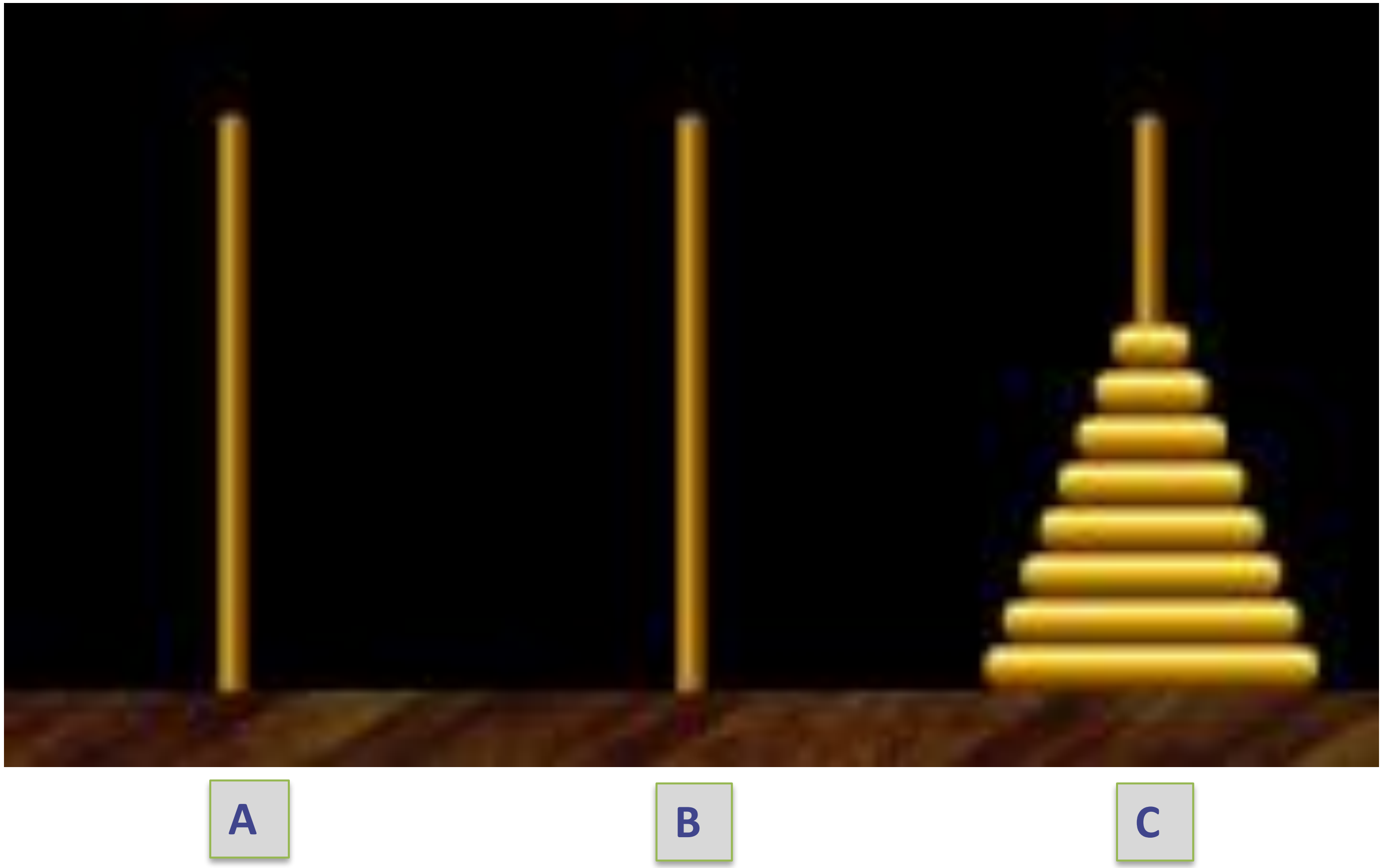
Recursion : Tower of Hanoi ..3



No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

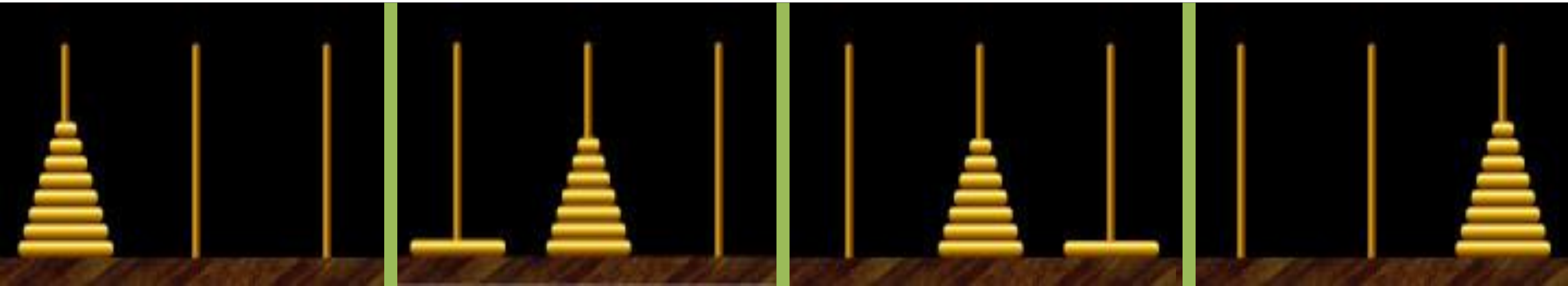
Recursion : Tower of Hanoi ..4



No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

```
// move n disks From A to C using B as aux
void hanoi(int n, char A, char C, char B) {
  if (n==0) { return; } // nothing to move!!
    // recursively move n-1 disks
    // from A to B using C as aux
    hanoi(n-1, A, B, C);
    // atomic move of a single disk
    printf("Move 1 disk from %c to %c\n", A, C);
    // recursively move n-1 disks
    // from B to C using A as aux
    hanoi(n-1, B, C, A);
}
```



OUTPUT for hanoi(4, ‘A’, ‘C’, ‘B’)

```
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from C to A
Move 1 disk from C to B
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from B to A
Move 1 disk from C to A
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
```

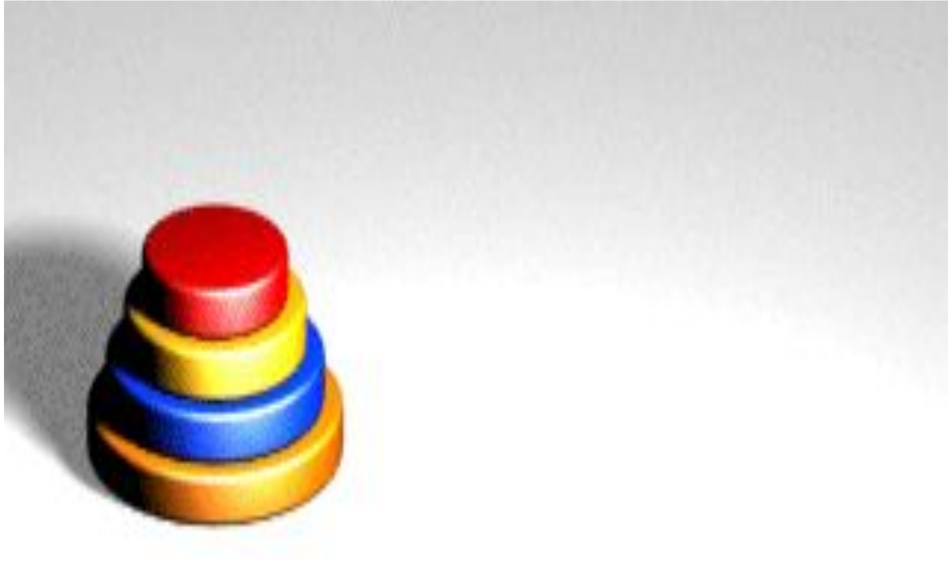


Image Source: http://upload.wikimedia.org/wikipedia/commons/6/60/Tower_of_Hanoi_4.gif

The puzzle was invented by the French mathematician **Édouard Lucas** in 1883.

There used to be a story about a temple in Kashi Vishwanath, which contains a large room with three posts surrounded by 64 golden disks. Brahmin priests have been moving these disks, in accordance with the immutable rules of the Brahma. The puzzle is therefore also known as the **Tower of Brahma** puzzle.

According to the legend, when the last move of the puzzle will be completed, the world will end.

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them $2^{64}-1$ seconds or roughly 585 billion years or about 127 times the current age of the sun.

Source: https://en.wikipedia.org/wiki/Tower_of_Hanoi

Next Class

- Introduction to arrays
  - Syntax
  - Basic I/O
  - Simple usage