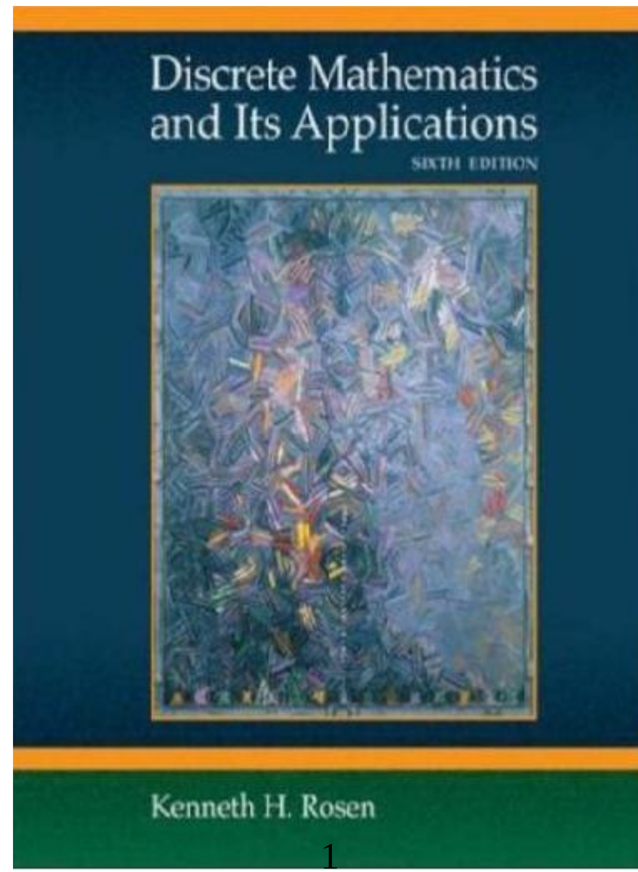
Jiangxi University of Science and Technology

Discrete Mathematics and Its Applications

Lecture016:

Recursive Algorithms

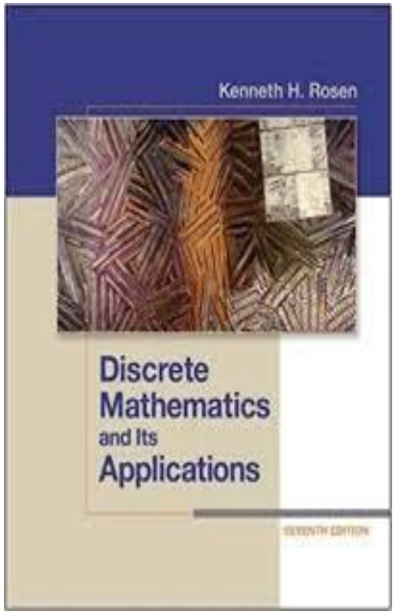


江西理工大學
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Acknowledgement

Most of these slides are adapted from ones created by Professor Bart Selman at Cornell University ,

and Dr Johnnie Baker and **Discrete Mathematics and Its Applications** **(Seventh Edition)** **Kenneth H. Rosen**



Recursive Algorithms

A recursive algorithm is an algorithm that solves the problem by reducing it to an instance of the same problem with smaller input.

Recursive Linear Search

Procedure search(i, j, x: i, j, x integers, $1 \le i \le n$, $1 \le j \le n$)
if $a_i = x$ **then**
    location := i
else if i=j **then**
    location := 0
else
    search(i+1,j,x)

Recursive Binary Search

Procedure *binary search* (i, j, x: i, j, x integers, $1 \le i \le n$, $1 \le j \le n$)

$m := \lfloor (i+j)/2 \rfloor$

if $x = a_m$ **then**

location := m

else if ($x < a_m$ and $i < m$) **then**

binary search(i, m-1,x)

else if ($x > a_m$ and $j > m$) **then**

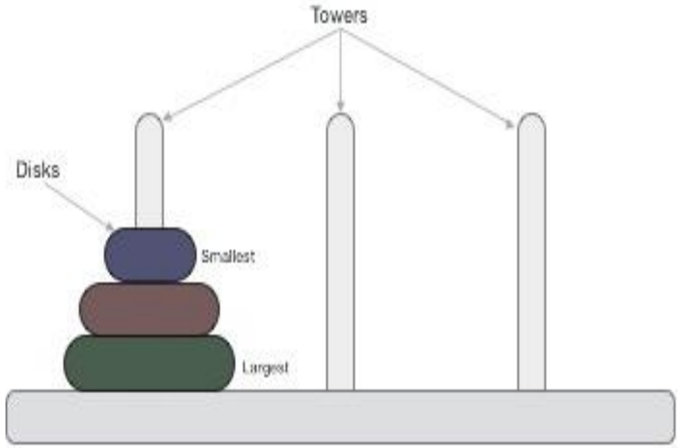
binary search(m+1,j,x)

else *location* :=0

Towers of Hanoi

Three ROD

- Start ROD
- Auxiliary ROD
- End ROD

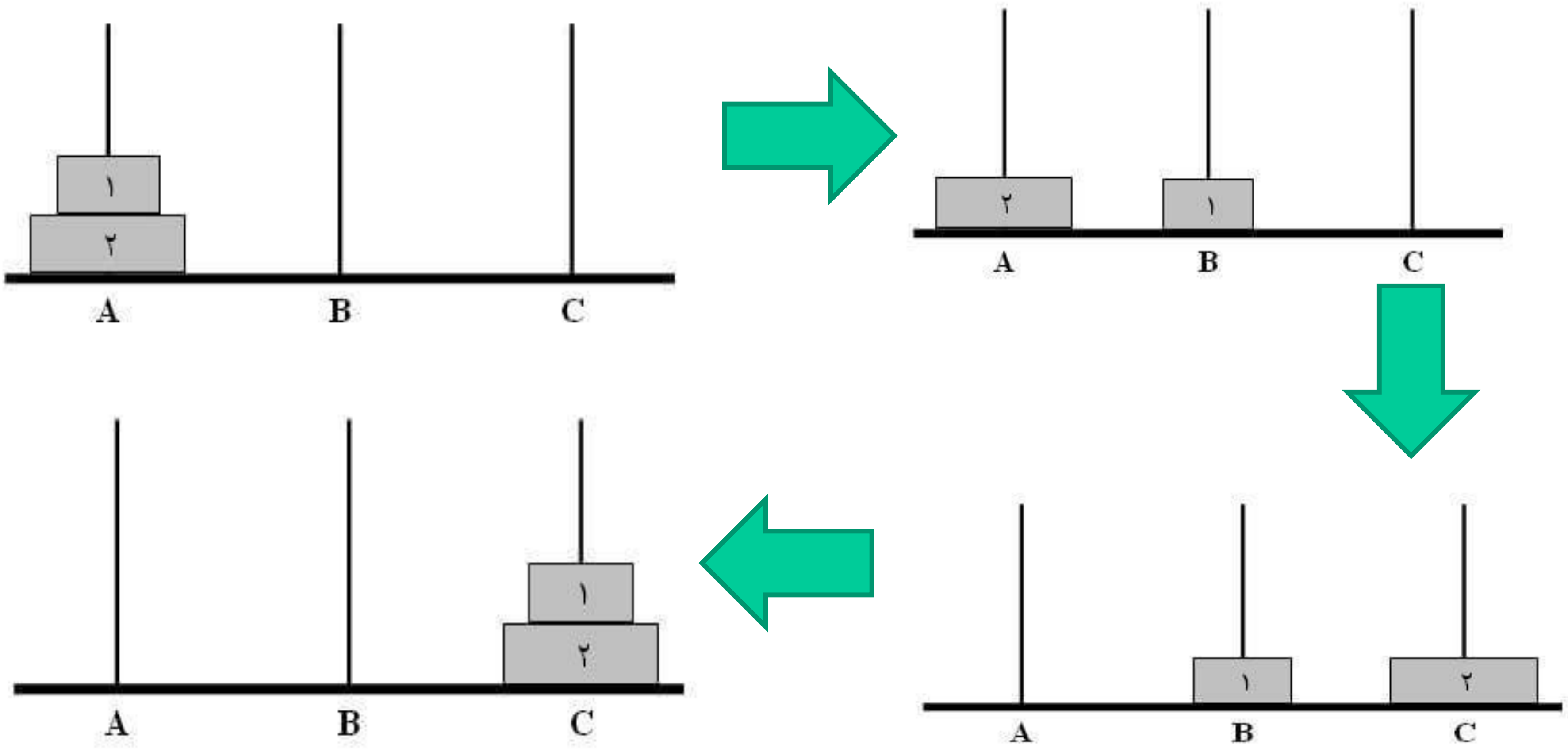


The aim is to transfer all Disk from start rod to the end with this below rule :

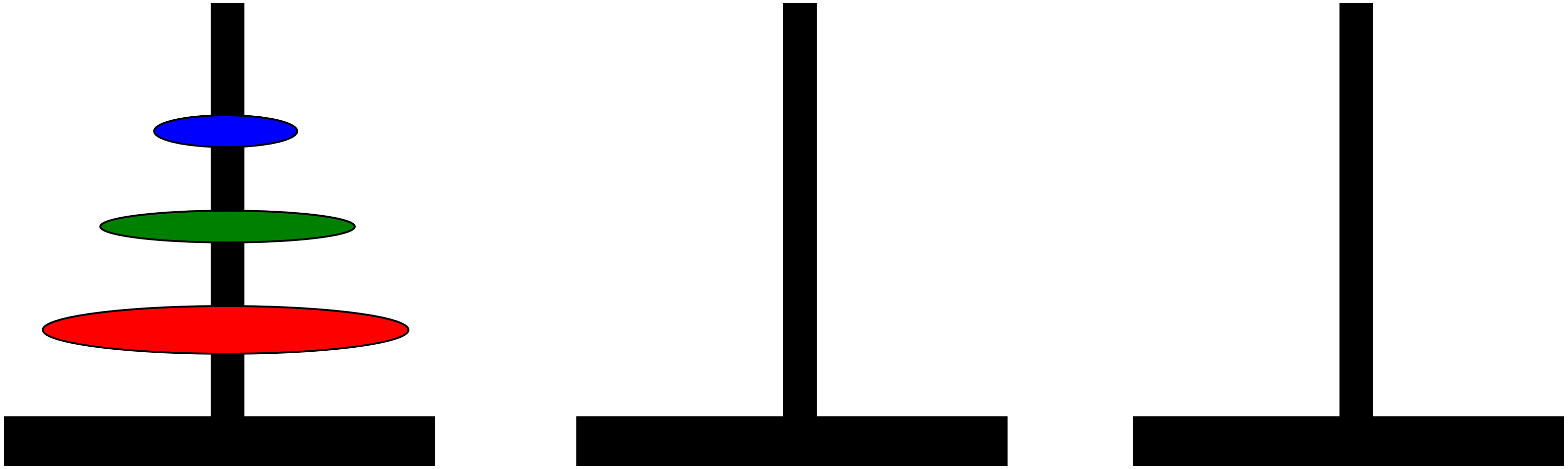
- 1. each time just we can transfer one Disk
- 2.Sort the disk small to big (you should put the Disk based on Size)

- Only one disc could be moved at a time
- A larger disc must never be stacked above a smaller one
- One and only one extra needle could be used for intermediate storage of discs

Towers of Hanoi



Towers of Hanoi (N=3)



Towers of Hanoi

- There are three pegs.
- 64 gold disks, with decreasing sizes, placed on the first peg.
- You need to move all of the disks from the first peg to the second peg.
- Larger disks cannot be placed on top of smaller disks.
- The third peg can be used to temporarily hold disks.

Tower of Hanoi

The disks must be moved within one week. Assume one disk can be moved in 1 second. Is this possible?

To create an algorithm to solve this problem, it is convenient to generalize the problem to the “N-disk” problem, where in our case N = 64.

Tower of Hanoi

How to solve it?

Think recursively!!!!

Suppose you could solve the problem for n-1 disks, i.e., you can move (n-1) disks from one tower to another, without ever having a large disk on top of a smaller disk.

How would you do it for n?

Solution:

- Move top (n-1) disks from tower 1 to tower 3

(you can do this by assumption – just pretend the largest ring is not there at all).

- Move largest ring from tower 1 to tower 2.

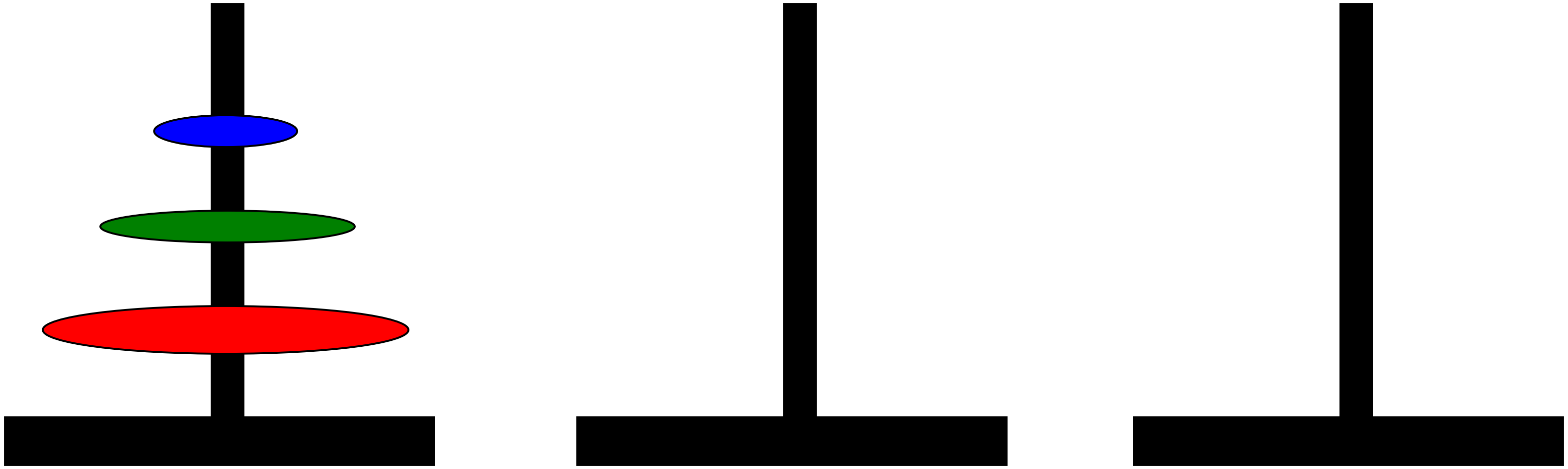
- Move top (n-1) rings from tower 3 to tower 2

(again, you can do this by assumption).

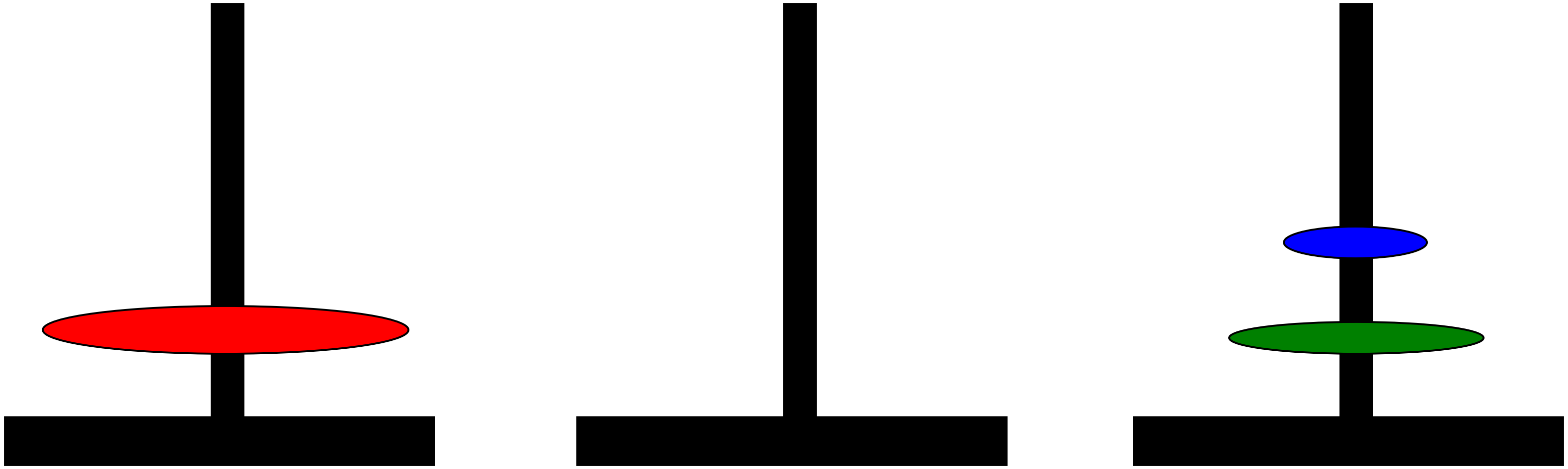
C++ and Tower of Hanoi

```
void hanoi(int nDisk, char start, char temp, char finish){
    if(nDisk == 1)
      cout << start << " -> " << finish << endl;
    else{
      hanoi(nDisk - 1, start, finish, temp);
      cout << start << " -> " << finish << endl;
      hanoi(nDisk - 1, temp, start, finish);
    }
}
```

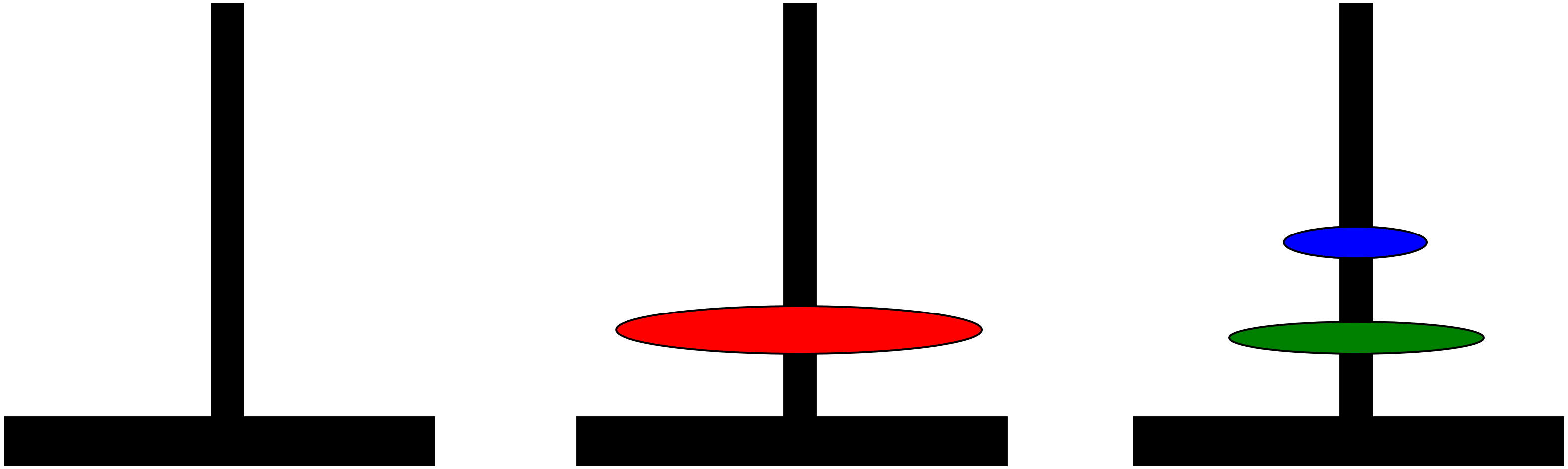
Recursive Solution



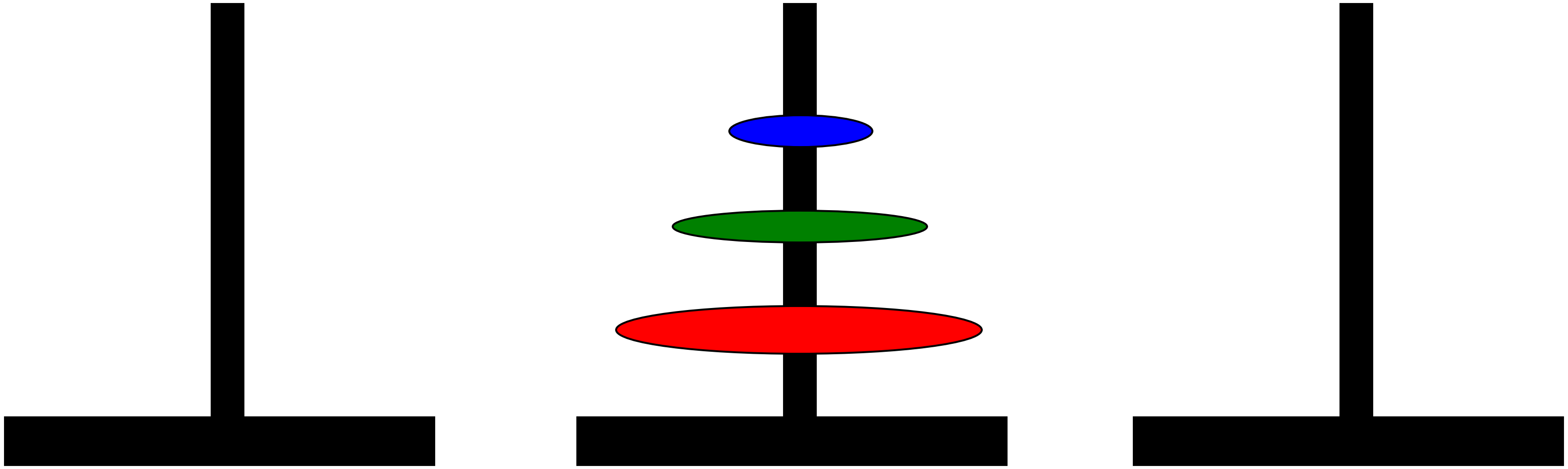
Recursive Solution



Recursive Solution



Recursive Solution



Towers of Hanoi

Procedure *TowerHanoi* (n, a, b, c: n, x, y, z integers, $1 \le a \le 3$, $1 \le b \le 3$, $1 \le c \le 3$)

if n= 1 **then**

move(a,b)

else

begin

*TowerHanoi(*n-1, a, c, b)

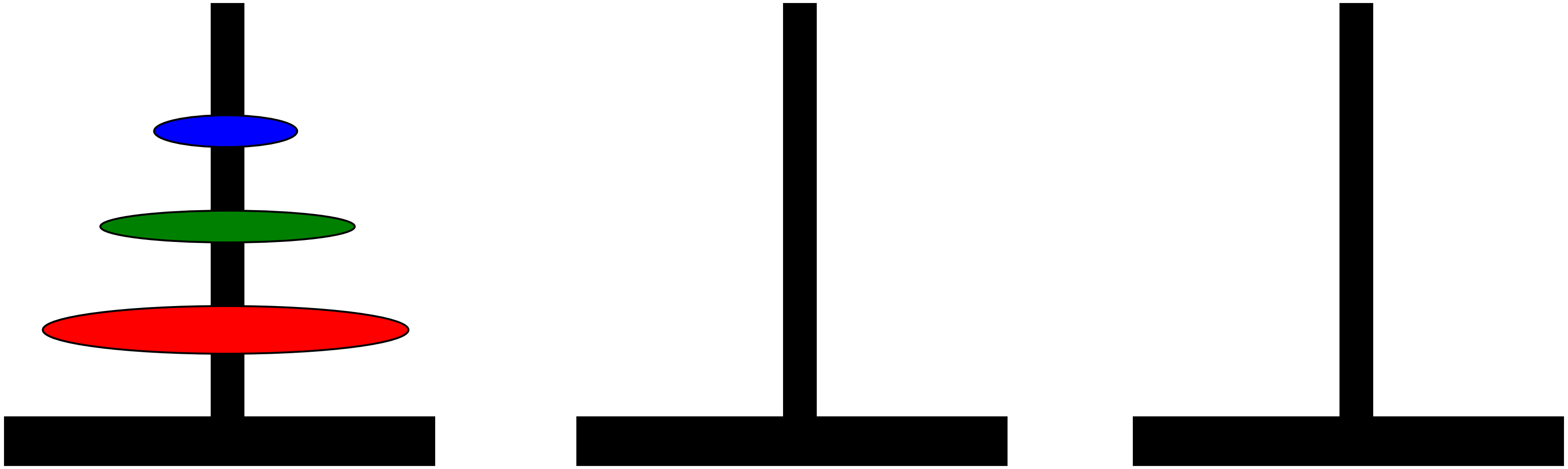
move(a,b);

TowerHanoi (n-1,c,b,a);

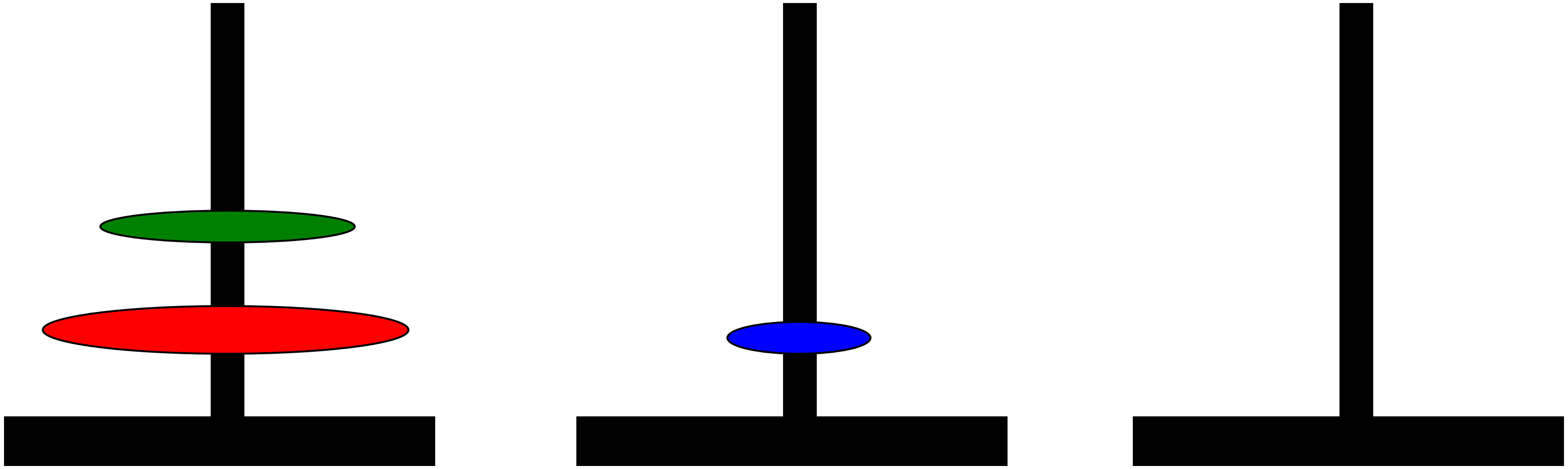
end

{*TowerHanoi* is the procedure to move *n* disks from tower *a* to tower *b* using tower *c* as an intermediate tower; *move* is the procedure to move a disk from tower a to tower b)

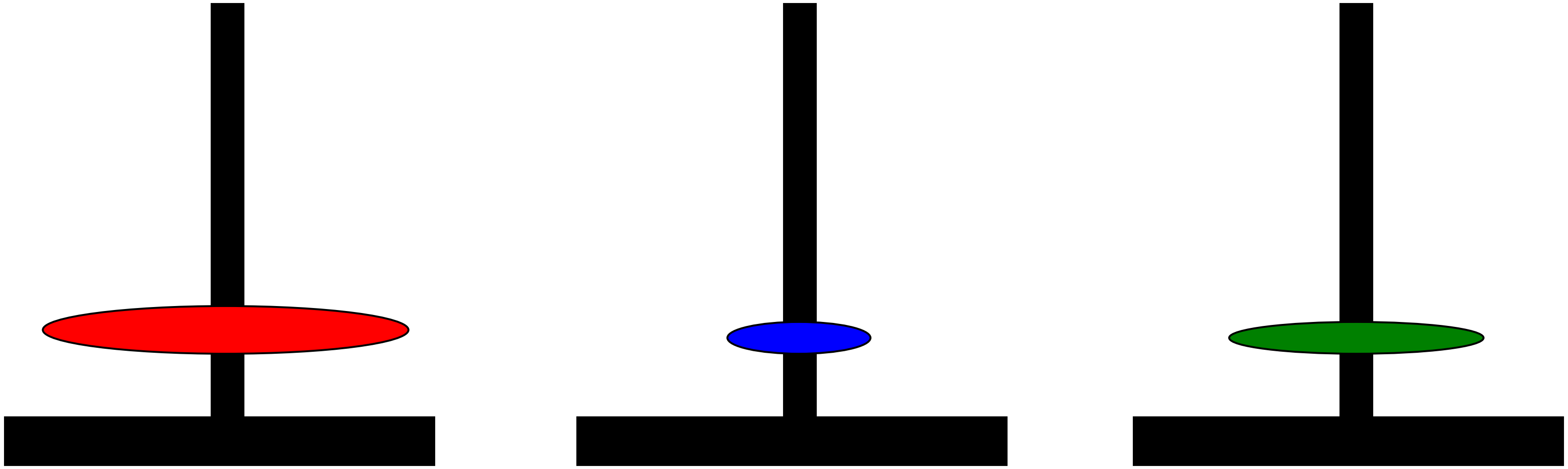
Tower of Hanoi



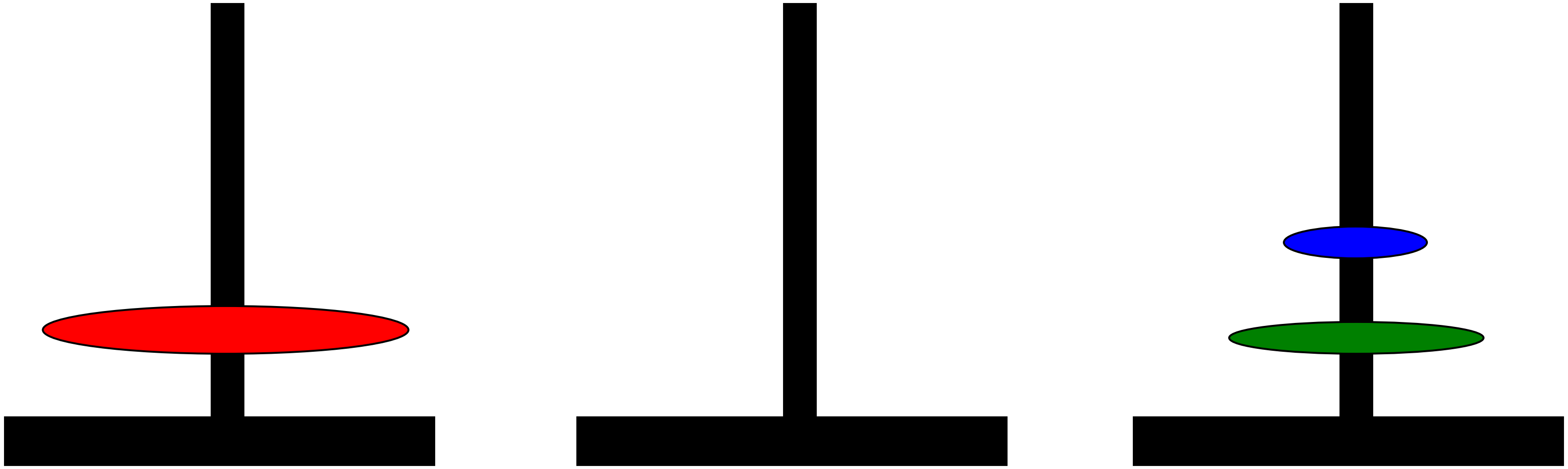
Tower of Hanoi



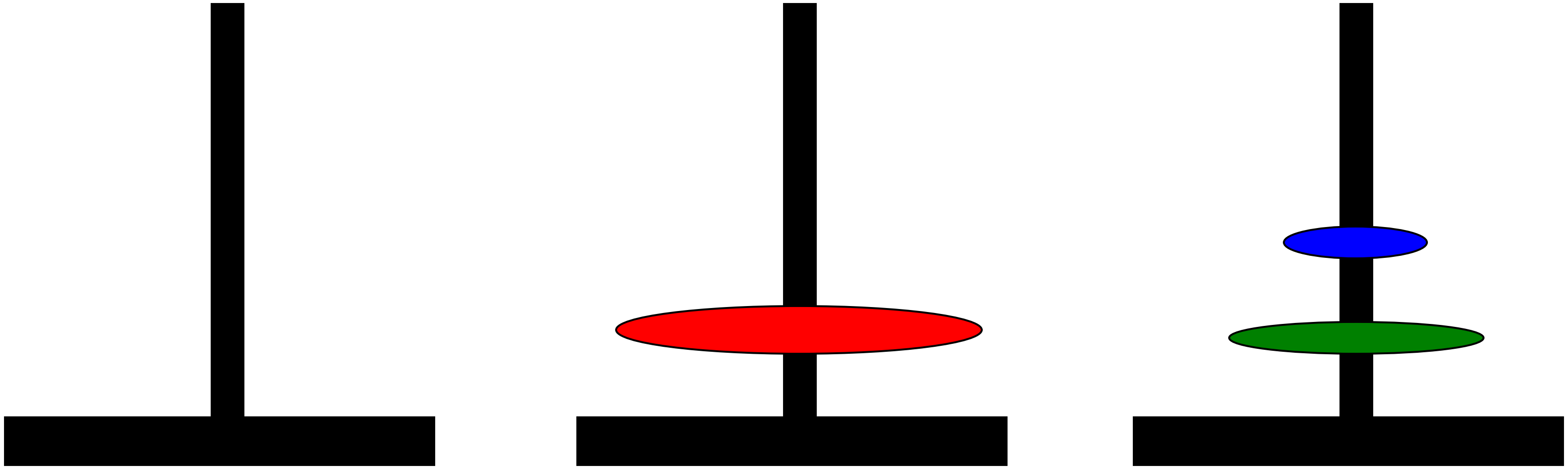
Tower of Hanoi



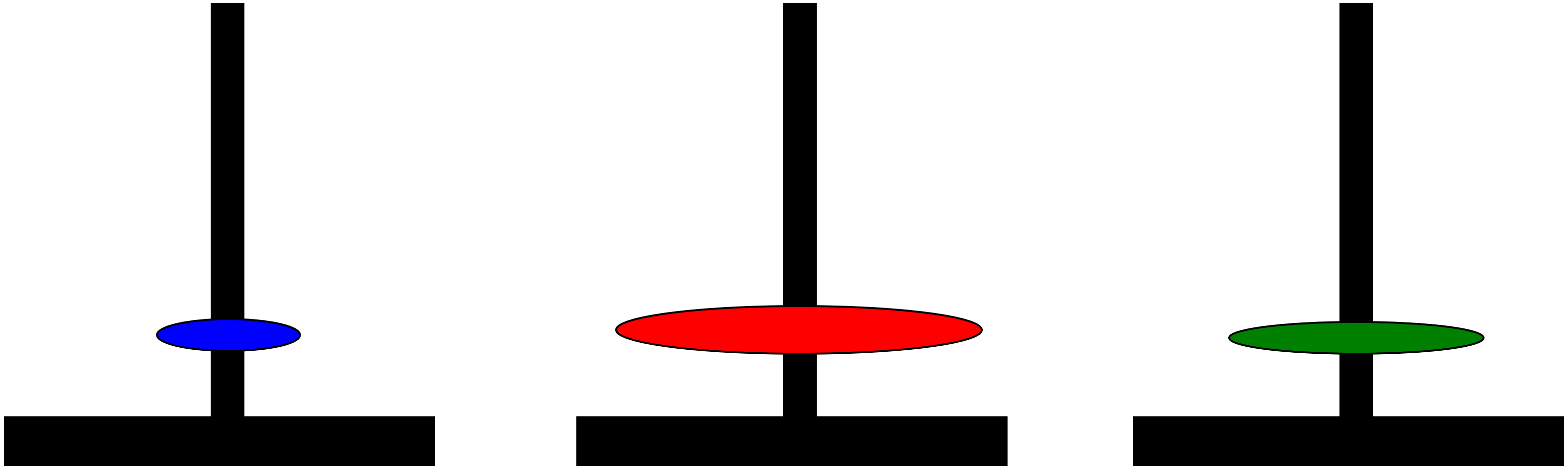
Tower of Hanoi



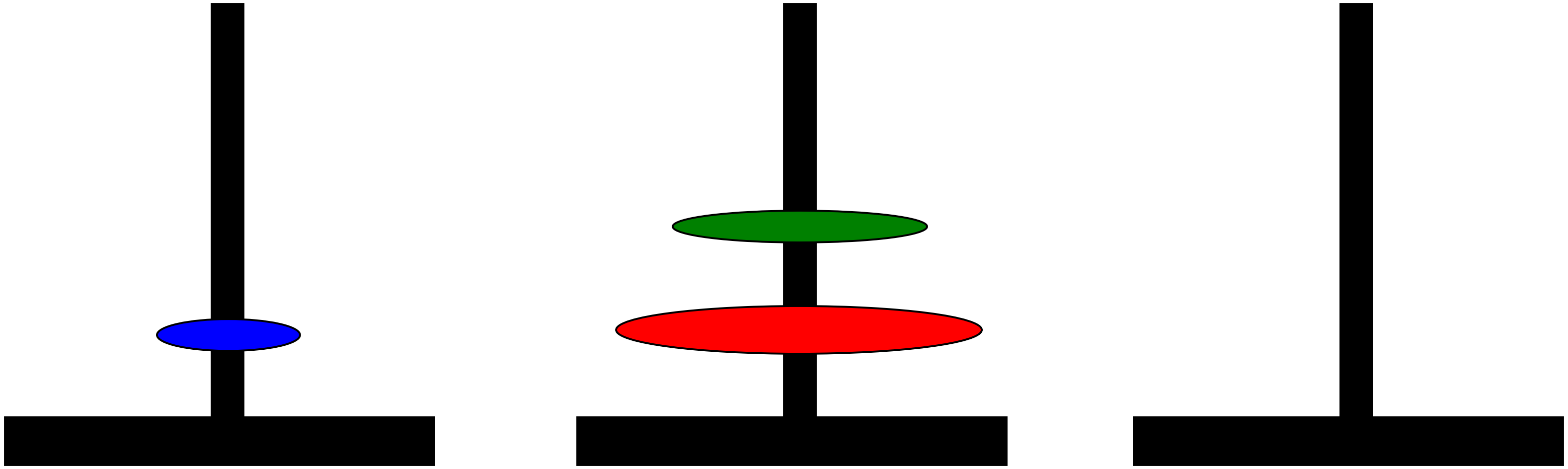
Tower of Hanoi



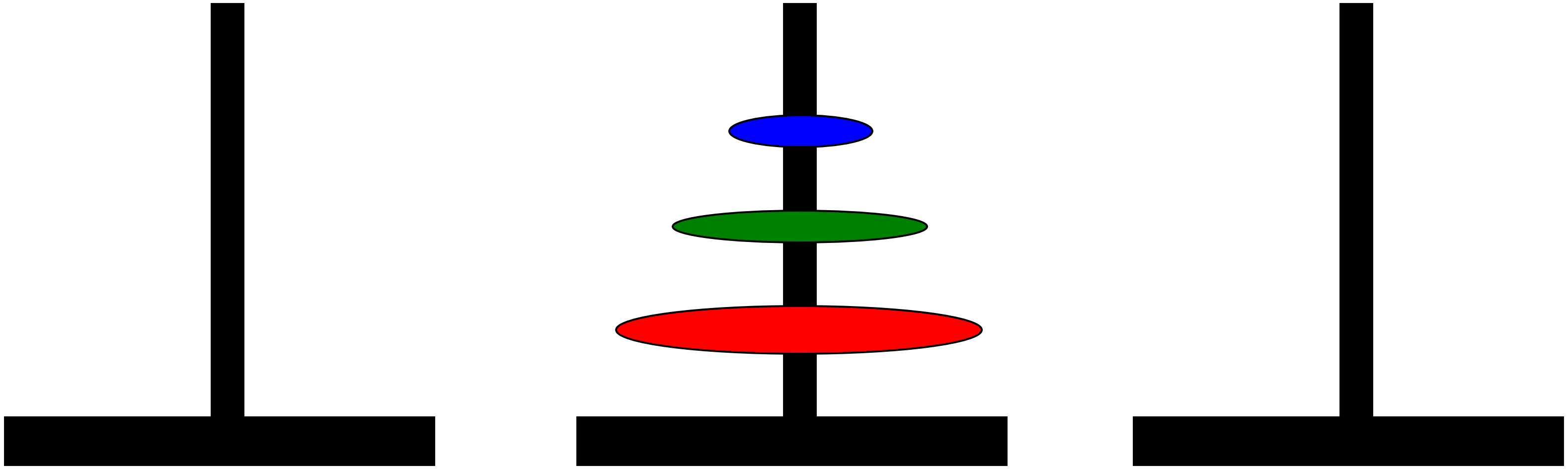
Tower of Hanoi



Tower of Hanoi



Tower of Hanoi



Analysis of Towers of Hanoi

- Hypothesis --- it takes $2^n - 1$ moves to perform TowerHanoi(n,a,b,c) for all positive n.

- Proof:
- Basis: P(1) – we can do it using move(a,b) i.e., $2^1 - 1 = 1$
- Inductive Hypothesis: P(n) - it takes $2^n - 1$ moves to perform TowerHanoi(n,a,b,c)
- Inductive Step: In order to perform TowerHanoi(n+1,a,b,c)
- we do: TowerHanoi(n,a,c,b), move(a,c), and TowerHanoi(n,c,b,a);
- Assuming the IH this all takes $2^n - 1 + 1 + 2^n - 1 = 2 \times 2^n - 1 = 2^{(n+1)} - 1$

N = 64 **Note: $(2^{64}) - 1 = 1.84467441 \times 10^{19}$**

QED

Recursion and Iteration

- A recursive definition expresses the value of a function at a positive integer in terms of the values of the function at smaller integers.

- But, instead of successively reducing the computation to the evaluation of the function at smaller integers, we can start by considering the base cases and successively apply the recursive definition to find values of the function at successive larger integers.

Towers of Hanoi

```
void hanoi(int from, int to, int num)
{
    int temp = 6 - from - to; //find the temporary
                              //storage column
    if (num == 1){
        cout << "move disc 1 from " << from
             << " to " << to << endl;
  }
    else {
        hanoi(from, temp, num - 1);
        cout << "move disc " << num << " from " << from
             << " to " << to << endl;
        hanoi(temp, to, num - 1);
    }
}
```

Towers of Hanoi

```
int main() {
    int num_disc;  //number of discs

    cout << "Please enter a positive number (0 to quit)";
    cin >> num_disc;

    while (num_disc > 0){
        hanoi(1, 3, num_disc);
        cout << "Please enter a positive number ";
        cin >> num_disc;
    }
    return 0;
}
```

Is there any other way ?????

Non recursive

```
void hanoi(int nDisk, char start, char temp, char finish){
  int max = nDisk;
  char dest = finish;
  int disk = max;
  while(true){
    while(disk > 0){
      if(moving disk succeeds)
      {
        if(disk == max){
          max--;
          if(max == 0)
            return;
        }
        dest = the final place of max;
      }
      else
        dest = the alternative place between dest and the current place of disk;
      disk--;
    }
    p and q = the places different of dest;
    disk = the smaller of the disks on top of p and q;
    dest = the place between p and q with greater disk on top;
  }
}
```

Fibonacci

- **The Fibonacci sequence is one of the most famous formulas in mathematics.**

Each number in the sequence is the sum of the two numbers that precede it.

So, the sequence goes: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, and so on.

The mathematical equation describing it is $X_{n+2} = X_{n+1} + X_n$

The next number is found by adding up the two numbers before it.

- The 2 is found by adding the two numbers before it (1+1)
- The 3 is found by adding the two numbers before it (1+2),
- And the 5 is (2+3),
- and so on!

Fibonacci

1 1 2 3 5 8 13 21

Fibonacci Numbers

- Fibonacci numbers can also be represented by the following formula.

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Recursive Fibonacci

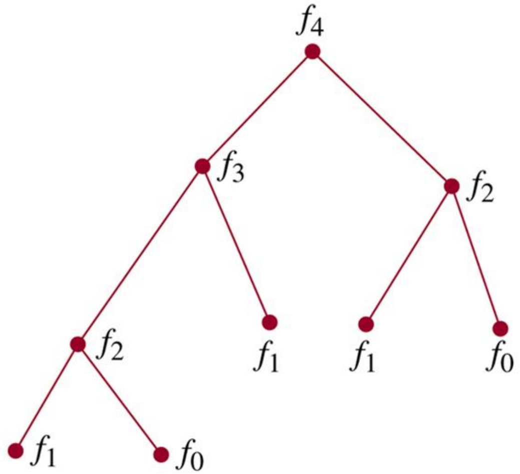
procedure *fibonacci* (n: nonnegative integer)

if n = 0 **then** *fibonacci*(0) :=0

else if n =1 **then** *fibonacci*(1) := 1

else *fibonacci*(n): = *fibonacci*(n-1) + *fibonacci*(n-2)

What's the "problem"
with this algorithm?

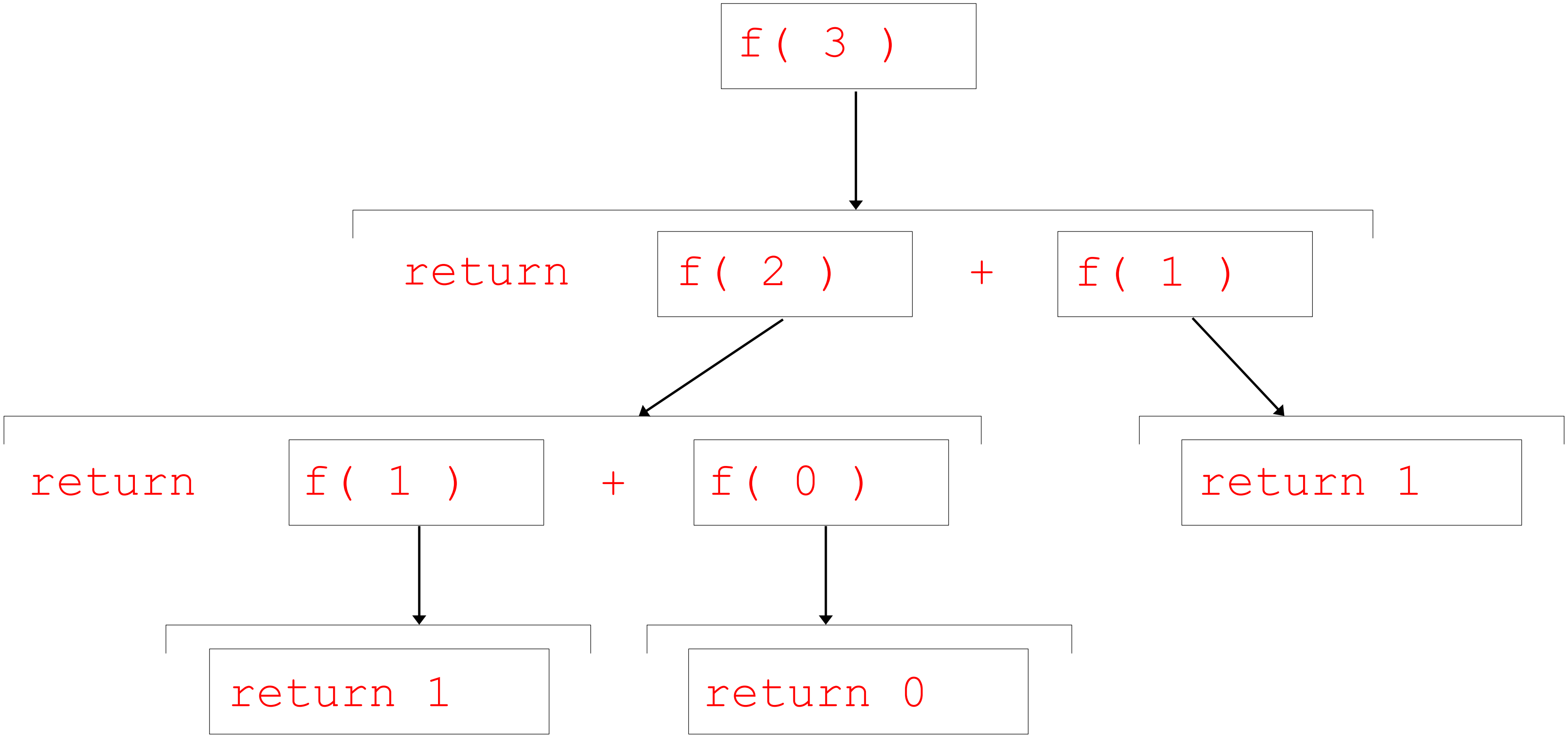


Iterative Fibonacci

```
procedure iterativefibonacci(n: nonnegative integer)
if n=0 then y:= 0
else
begin
        x := 0
        y := 1
        for i := 1 to (n-1)
        begin
                z := x + y
                x := y
                y := z
        end
end
{y is the nth Fibonacci number}
```

$$f(n) = \begin{cases} 0 & \text{if} \quad n = 0 \\ 1 & \text{if} \quad n = 1 \\ f(n-1) + f(n-2) & \text{if} \quad n > 1 \end{cases}$$

Example Using Recursion: Fibonacci Series



Fibbonacci in C++

```
#include <stdio.h>
START
Procedure Fibonacci(n)
   declare f0, f1, fib, loop
      set f0 to 0
   set f1 to 1
      display f0, f1
      for loop ← 1 to n
         fib ← f0 + f1
         f0 ← f1
         f1 ← fib
       display fib
   end for
 END
```

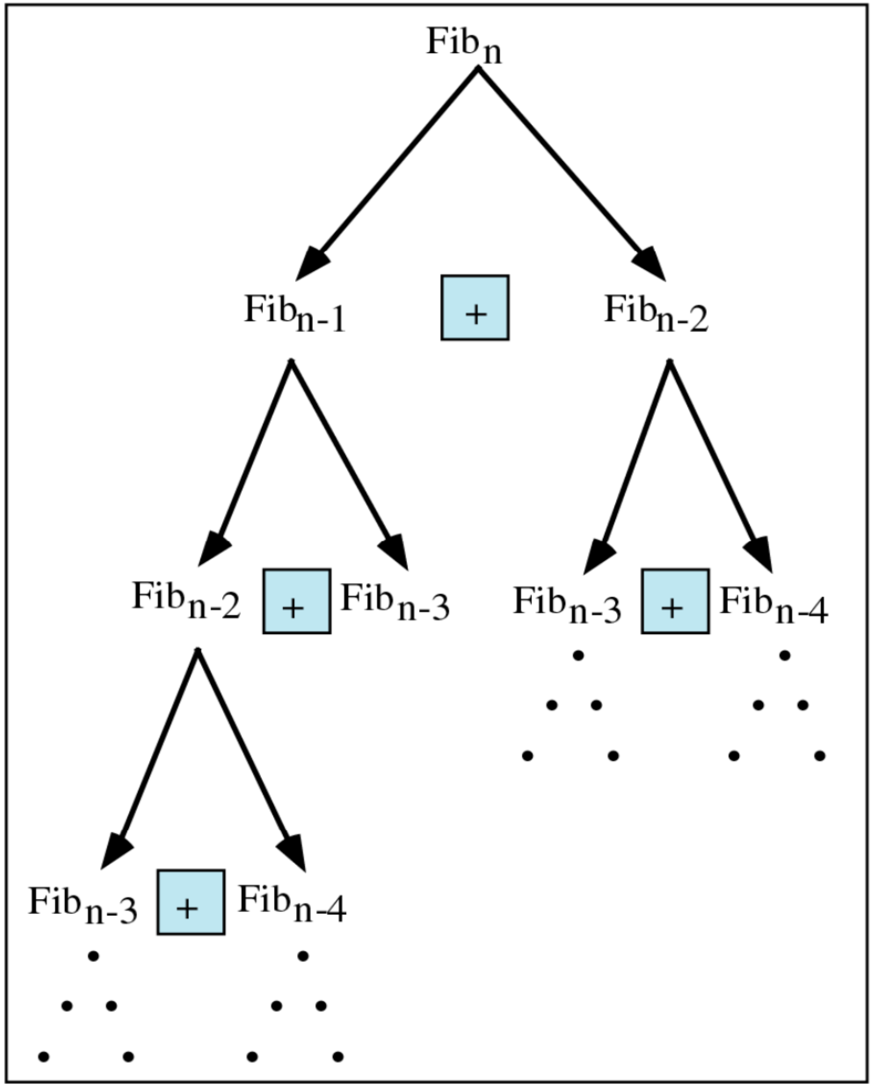
Factorial of 5: 120

Fibbonacci of 5: 0 1 1 2 3

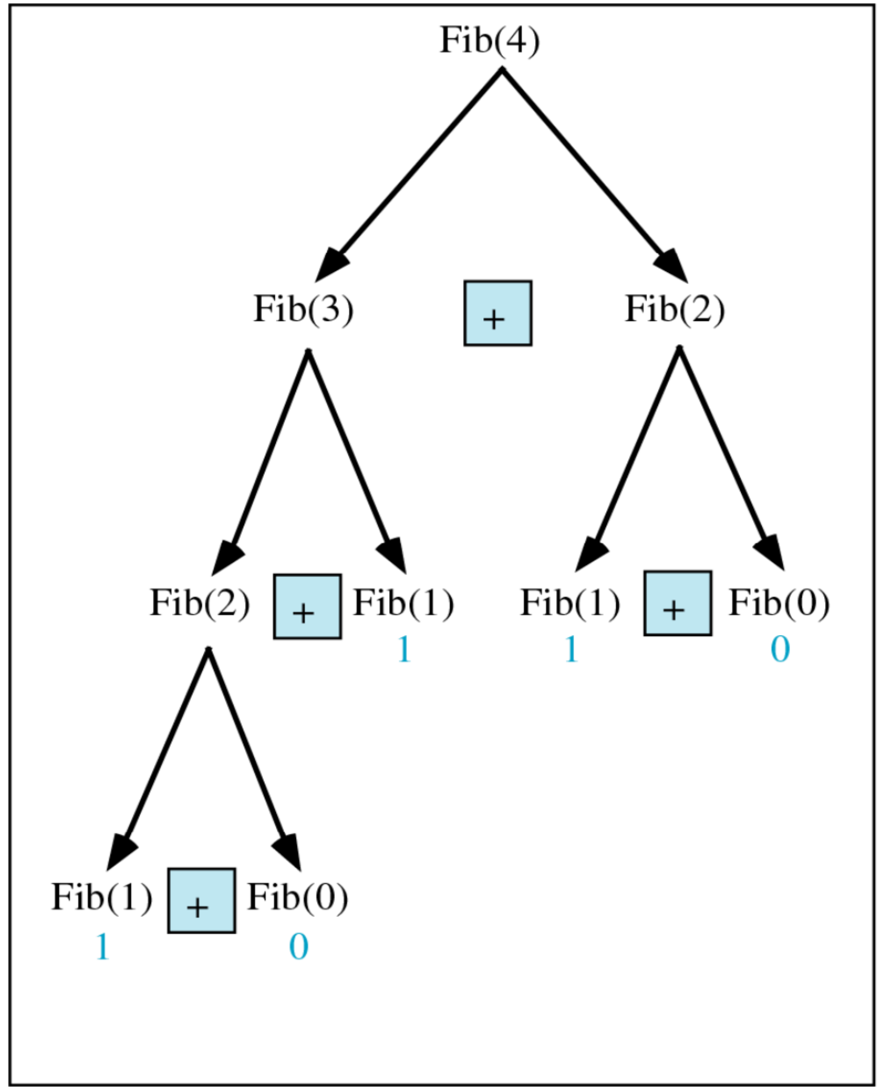
Fibonacci numbers

```
//Calculate Fibonacci numbers using recursive function.
//A very inefficient way, but illustrates recursion well
int fib(int number)
{
   if (number == 0) return 0;
   if (number == 1) return 1;
   return (fib(number-1) + fib(number-2));
}

int main(){          // driver function
    int inp_number;
    cout << "Please enter an integer: ";
    cin >> inp_number;
    cout << "The Fibonacci number for "<< inp_number
           << " is "<< fib(inp_number)<<endl;
  return 0;
}
```



(a) Fib(n)



(b) Fib(4)

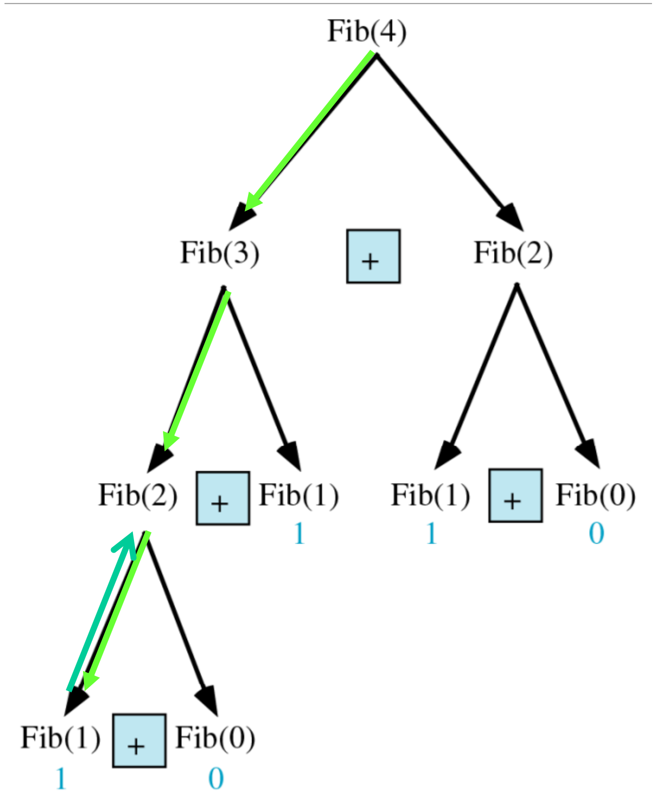
江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

Trace a Fibonacci Number

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

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

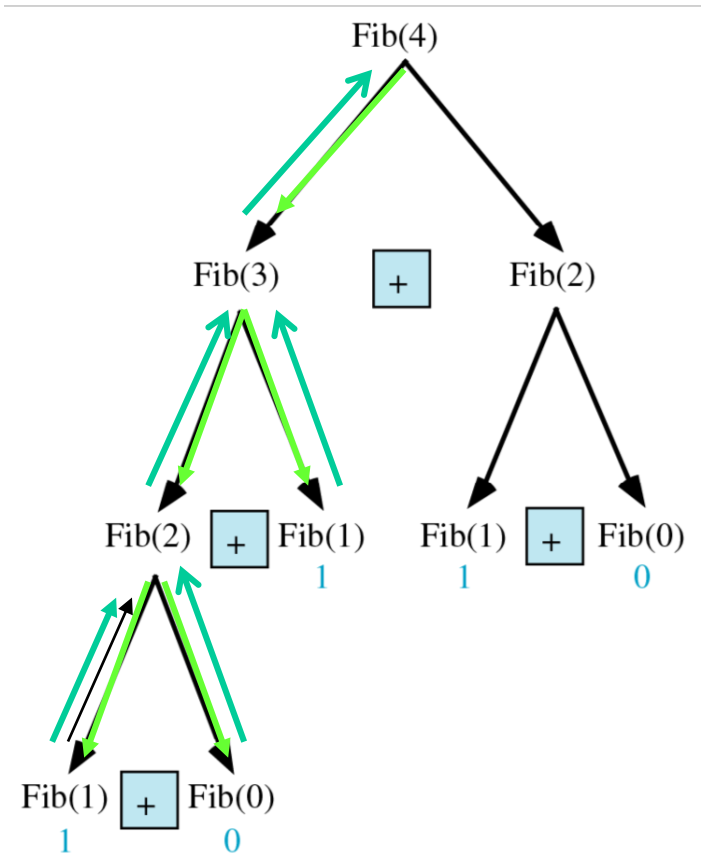
```
fib(4):
    4 == 0 ? No;    4 == 1?  No.
    fib(4) = fib(3) + fib(2)
    fib(3):
        3 == 0 ? No; 3 == 1? No.
        fib(3) = fib(2) + fib(1)
        fib(2):
            2 == 0? No; 2==1? No.
            fib(2) = fib(1)+fib(0)
           fib(1):
               1== 0 ? No; 1 == 1? Yes.
    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



江西理工大学
JIANGXI UNIVERSITY OF SCIENCE AND TECHNOLOGY

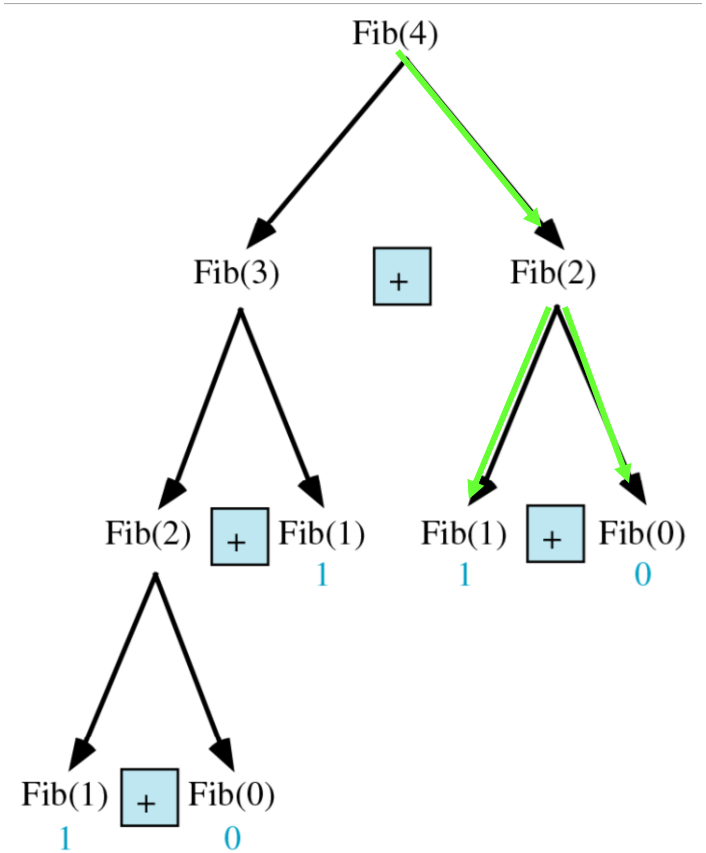
Trace a Fibonacci Number

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



Trace a Fibonacci Number

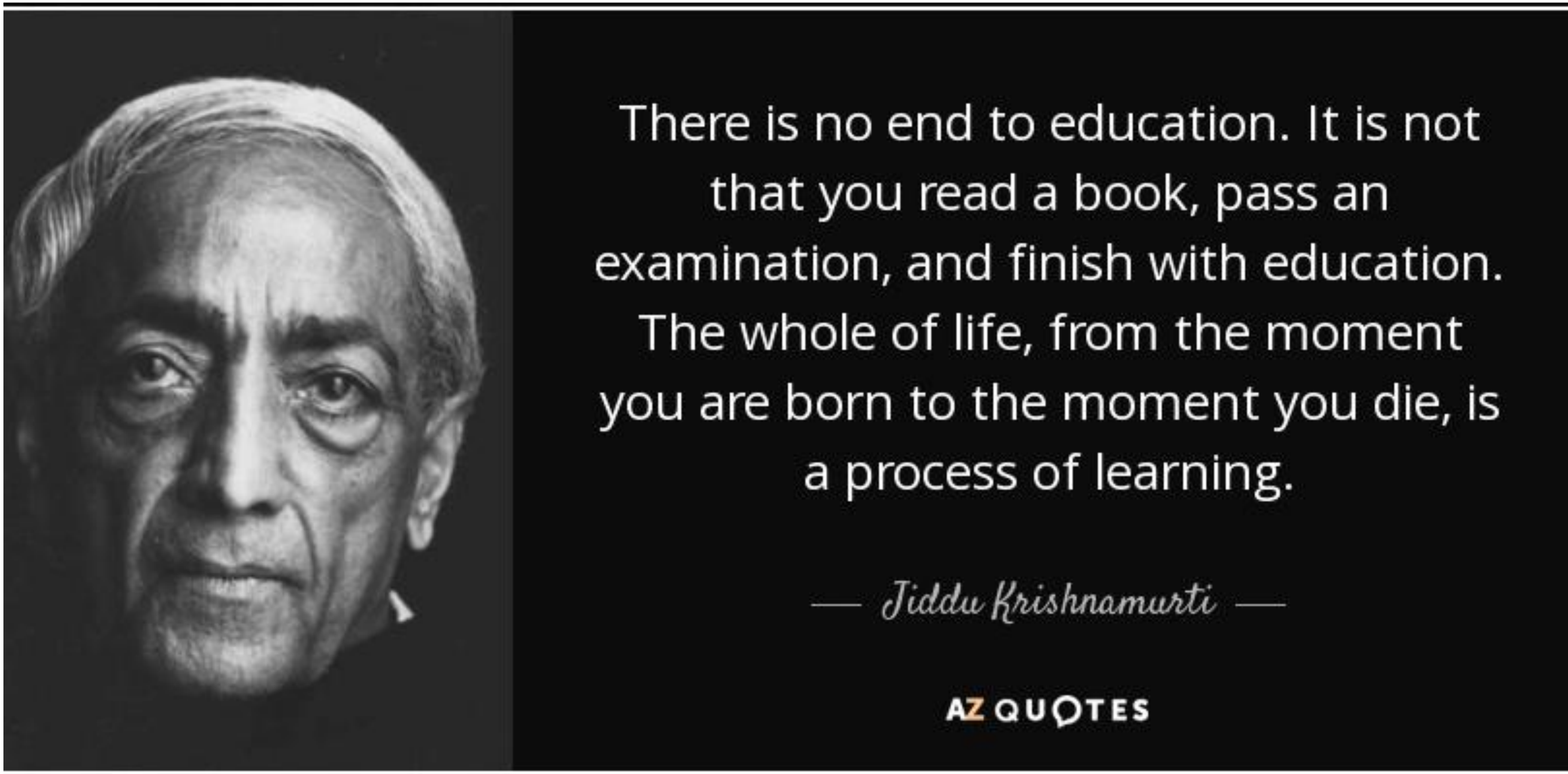
```
fib(2):
    2 == 0 ? No; 2 == 1?    No.
    fib(2) = fib(1) + fib(0)
    fib(1):
       1== 0 ? No; 1 == 1?  Yes.
        fib(1) = 1;
        return fib(1);
     fib(0):
        0 == 0 ?    Yes.
        fib(0) = 0;
        return fib(0);
     fib(2) = 1 + 0 = 1;
     return fib(2);
  fib(4) = fib(3)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



Example 4: Fibonacci number w/o recursion

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

int fib(int n)
{
  int f[100];
  f[0] = 0; f[1] = 1;
  for (int i=2; i<= n; i++)
      f[i] = f[i-1] + f[i-2];
  return f[n];
}
```


There is no end to education. It is not that you read a book, pass an examination, and finish with education. The whole of life, from the moment you are born to the moment you die, is a process of learning.
— Jiddu Krishnamurti —
AZ QUOTES