Recursion

CMSC 132, Summer 2015
Object-Oriented Programming II

Anwar Mamat

Recursion

- When one function calls itself directly or indirectly
- a method where the solution to a problem depends on solutions to smaller instances of the same problem.

Factorial

Definition:

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0 \end{cases}$$

5! =?

5! = 5 * 4!

4 = 4 * 3!

3! = 3 * 2!

2! = 2 * 1!

1! = 1

5 * 24 = 120

4 * 6

3 * 2

2 * 1

Greatest Common Divisor

- gcd: Find largest integer d that evenly divides into p and q **Euclid's algorithm. [300 BCE]**

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \% q) & \text{otherwise} \end{cases}$$

← base case

← reduction step, converges to base case

```
gcd(4032, 1272)  = gcd(1272, 216)
                 = gcd(216, 192)
                 = gcd(192, 24)
                 = gcd(24, 0)
                 = 24.
```

4032 = 3 × 1272 + 216

Greatest Common Divisor

- gcd: Find largest integer d that evenly divides into p and q **Euclid's algorithm. [300 BCE]**

$$\gcd(p, q) = \begin{cases} p & \text{if } q = 0 \\ \gcd(q, p \% q) & \text{otherwise} \end{cases}$$

← base case

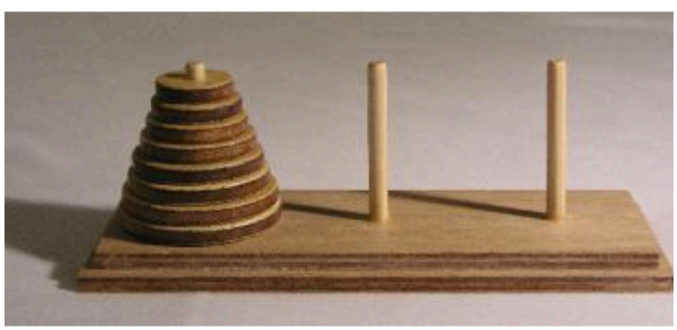
← reduction step, converges to base case

Java implementation.

```
public static int gcd(int p, int q) {
    if (q == 0) return p;
    else return gcd(q, p % q);
}
```

← base case
← reduction step

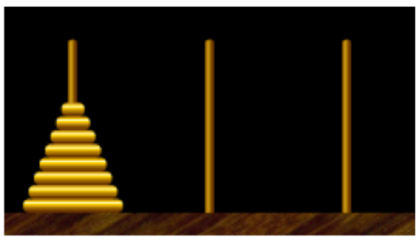
Towers of Hanoi



http://en.wikipedia.org/wiki/Image:Hanoiklein.jpg

Towers of Hanoi

- Move all the discs from the leftmost peg to the rightmost one.
  - Only one disc may be moved at a time.
  - A disc can be placed either on empty peg or on top of a larger disc



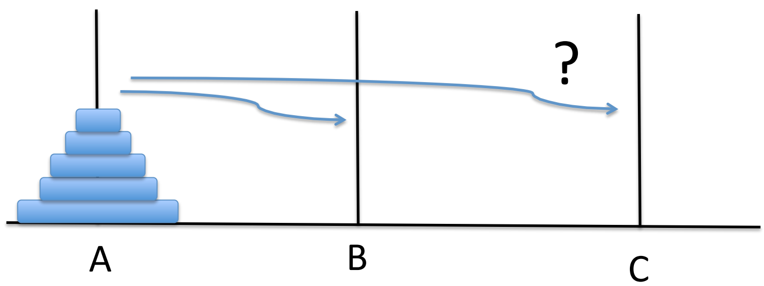
start

finish

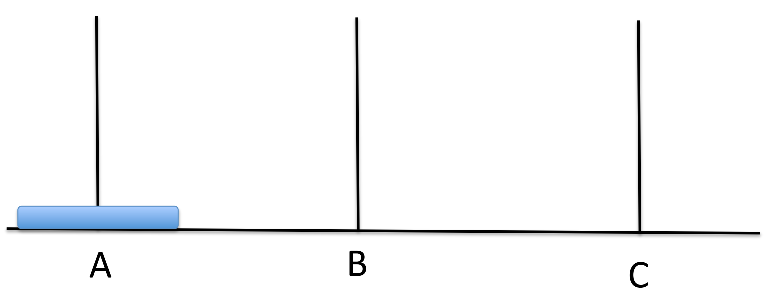
Towers of Hanoi Legend

- Q. Is world going to end (according to legend)?
  - 64 golden discs on 3 diamond pegs.
  - World ends when certain group of monks accomplish task.
- Q. Will computer algorithms help

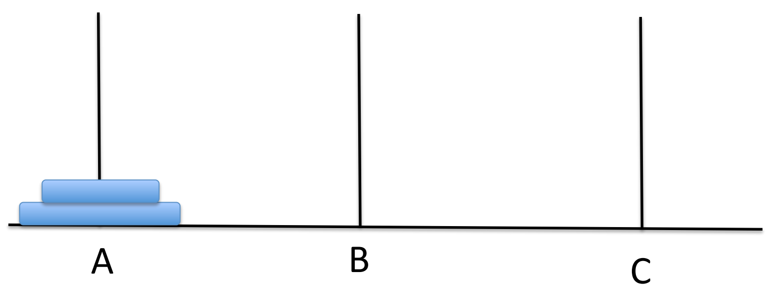
Let's move disks



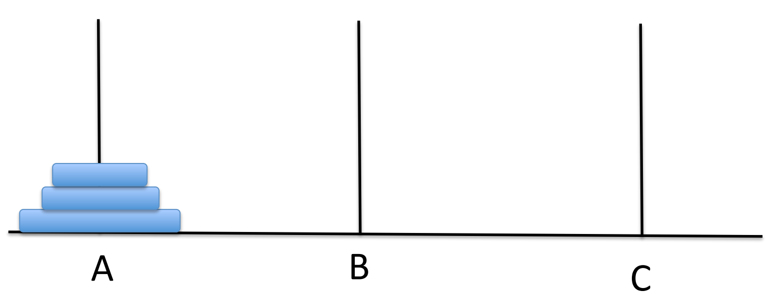
Move one disk



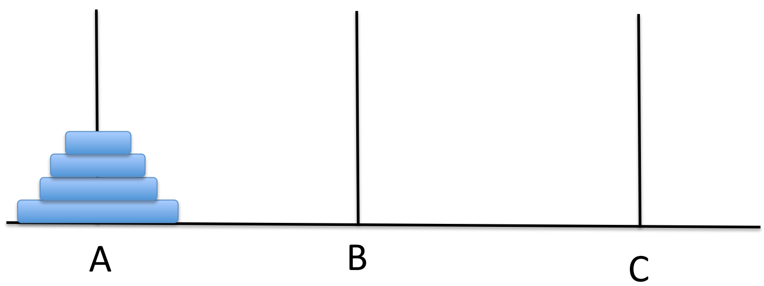
Move two disks



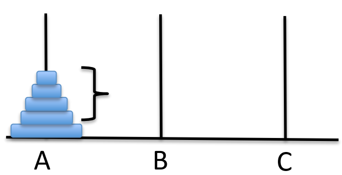
Move three disks



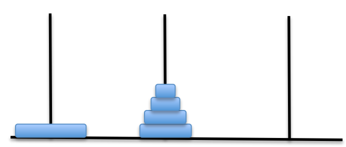
Move four disks



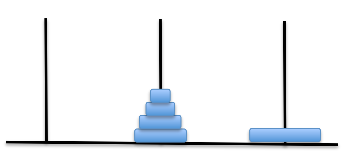
Move n disks



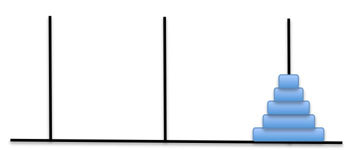
1. Move n-1 disks from A->B



2. Move the largest disk to C

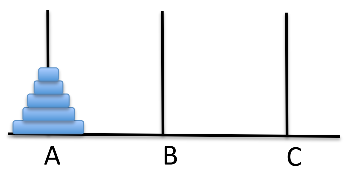


3. Move n-1 disks from B->C



Towers of Hanoi: Recursive Solution

```
public class TowersOfHanoi {
    public static void solve(int n, String A, String B, String C) {
       if (n == 1) {
           System.out.println(A + " -> " + C);
       } else {
           solve(n - 1, A, C, B);
           System.out.println(A + " -> " + C);
           solve(n - 1, B, A, C);
       }
    }
   public static void main(String[] args) {
        int discs = 3;
        solve(discs, "A", "B", "C");
   }
}
```



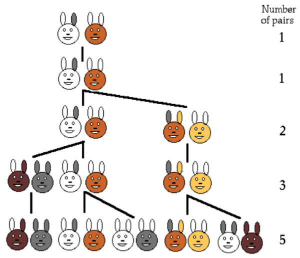
Remarkable properties of recursive solution

- Takes 2^n steps to solve n disc problem.
- Takes 585 billion years for n = 64(at rate of 1 disc per second).
- Reassuring fact: any solution takes at least this long

Fibonacci Number

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{otherwise} \end{cases}$$



Fibonacci rabbits

L. P. Fibonacci
(1170 - 1250)

Fibonacci Number

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34

A natural for recursion?

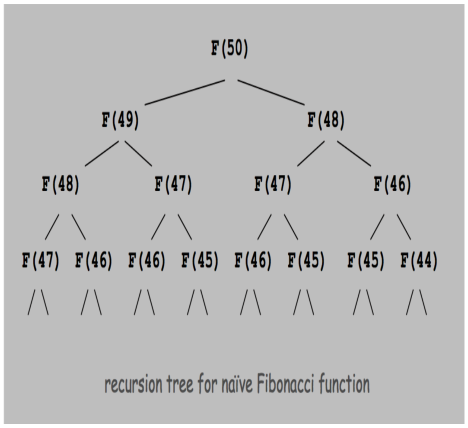
```
public static long F(int n) {
   if (n == 0) return 0;
   if (n == 1) return 1;
   return F(n-1) + F(n-2);
}
```

spectacularly inefficient code

Observation. It takes a really long time to compute `F(50)`.

Inefficient Recursion

Fibonacci numbers: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34



F(50) is called once.
F(49) is called once.
F(48) is called 2 times.
F(47) is called 3 times.
F(46) is called 5 times.
F(45) is called 8 times.
...
F(1) is called 12,586,269,025 times.

F(50)

Memoized Version of the Fibonacci

```
public Map<Integer, Integer> fibo;
public int fib(int n){
    int f1=0,f2=0;
    if( (n == 1)|| (n == 2)) return 1;
     else{
    if(fibo.containsKey(n-1)){
        f1 = fibo.get(n-1);
    }else{
        f1 = fib(n-1);
        fibo.put(n-1,f1);
    }
    if(fibo.containsKey(n-2)){
        f2 = fibo.get(n-2);
    }else{
        f2 = fib(n-2);
        fibo.put(n-2,f2);
    }
    return f2+f1;
    }
```