Introduction to Discrete Math

Felipe P. Vista IV

Course Outline

- Mathematical Thinking
  - Convincing Arguments, Find Example, Recursion, Logic, Invariants
- Probability & Combinatronics
  - Counting, Probability, Random Variables
- Graph Theory
  - Graphs (cycles, classes, parameters)
- Number Theory & Cryptography
  - Arithmetic in modular form
  - Intro to Cryptography

Mathematical Thinking – Recursion & Induction

RECURSION

- Recursion
- The Coin Problem
- Hanoi Towers

Line to process a paper

Frank

http://groups.csail.mit.edu/graphics/classes/6.837/F98/Lecture5/longline.gif

Compute Queue Length

1) Mary gets in line
2) She wonders how many people before her?
3) Mary asks Frank (in front of her)
   - *Could you please tell how many people ahead of you?*
4) Now, Frank has the same problem
   - But Frank was able to find out there are 8
5) Now Mary knows there are 9 people in front

Recursive Program

Algorithm:

numberOfPeopleInFront(F):

if there is no one before A:

return 0

F ← number of people before A

return *numberOfPeopleInFront*(F) + 1

Factorial Function

Definition:

- For a positive integer *n*, its factorial is the product of integers from 1 to n.
  - $5! = \underbrace{\overbrace{1 \text{ x } 2}^{2!} \text{ x } 3 \text{ x } 4}_{4!} \text{ x } 5 = 120 = 4! \text{ X } 5$

Recursive definition:

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

n! = x 4! = 4 (x3!)

Iterative Program

"iterate"

```
def factorial(n):
    assert(n > 0)
    result = 1
    for i in range(1, n+1)
        result *= i
    return result
```

→ result = result * i

res = res + 1 => res += 1

→ 1 * 2 * 3

res = res * 2

res = res * 3

res = res * n

Recursive Program

```
def factorial(n):
    assert(n > 0)
    if n == 1
        return 1
    else:
        return n * factorial(n - 1)
```

Termination

- Must make sure that recursive program (or definition) terminates after finite number of steps
- Achieved by decreasing some parameter until it reaches the base case
  - line length: line of the length decreases by 1 with each recursive call, until it becomes 1
  - Factorial: n decreases by 1

Example of Infinite Recursion

n = ~~3~~2

```
def infinite(n):
    if n == 1
        return 0
    return n * infinite(n+1)
```

if n==50, stop

- In theory:
  - will never stop, parameters increase to infinity
- In practice:
  - will cause error message
    - "Stack overload" or "Recursion depth exceeded"

More Examples of Infinite Recursion

- No base case:

```
def factorial(n):
    return n * factorial(n - 1)
```

- Parameter do not change:

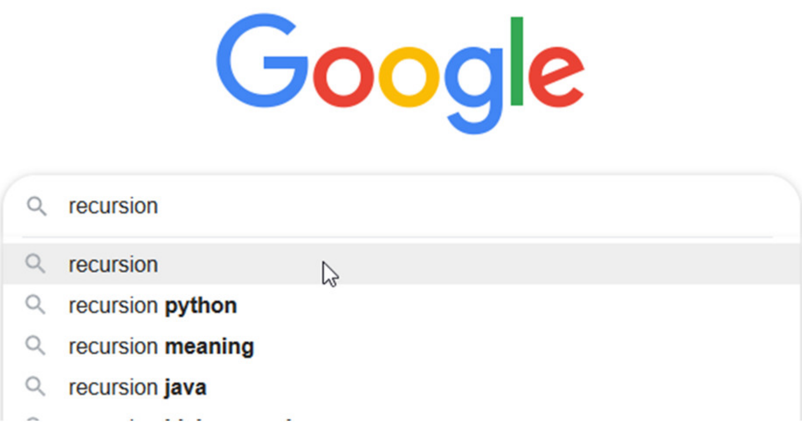
n = 1, n = 2

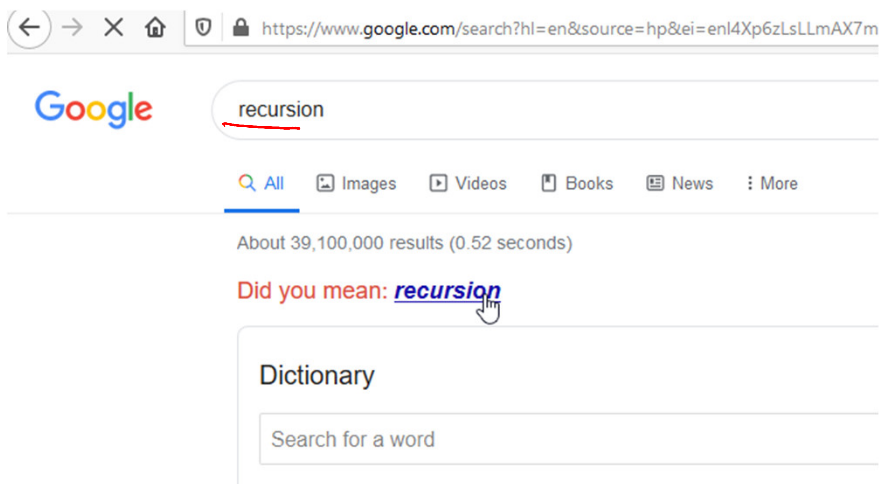
```
def infinite(n):
    if n == 1
        return 0
    return 1 + infinite(n)
```

More Examples (Not so serious :D)

- To understand recursion, one must first understand recursion

- Google is even in with the fun





- Recursion
- The Coin Problem
- Hanoi Towers

The Coin Problem

Problem

Prove that any amount starting from 8 **ewans** can be paid for using only 3 **ewans** and 5 **ewans.**

- $8 = 3+5$, $9=3+3+3$, $10=5+5$, $11=3+3+5$
  - looks promising, seems possible
- How to be sure it will always be possible?

Speculation

- 8 can be definitely ~~be~~ done
- 11 is also possible by adding one 3 ewan coin
  - 11 = 8 + 3
  - same principle for 14, 17, 20, etc…
- Same for 9, keep on adding a 3 ewan coin
  - will give 12, 15, 18, 21, etc
- Checking 10 ewans, still keep adding 3 ewan coins
  - we'll get 13, 16, 19, 22, etc

- *Speculation* – forming a theory without firm, solid concrete proof or evidence

Recursive Program

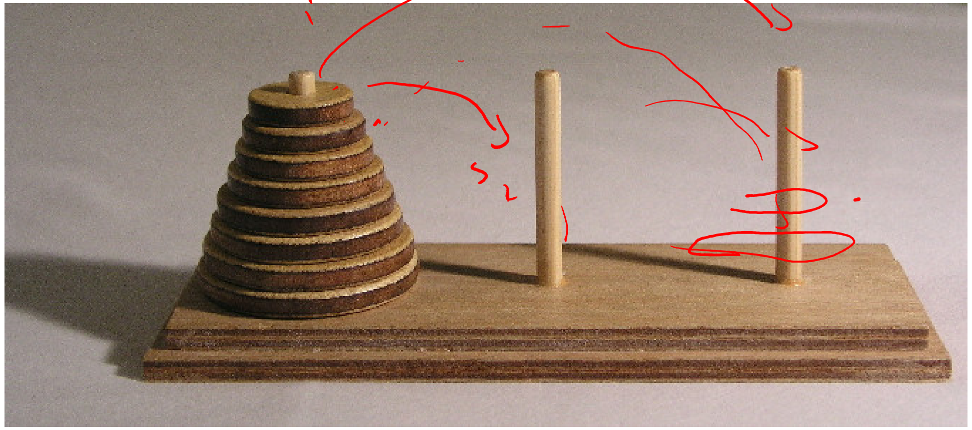
```
def change(amount):
    assert(amount >= 8)
    if amount == 8:
        return [3, 5]
    if amount == 9:
        return [3, 3, 3]
    if amount == 10:
        return [5, 5]

    coins = change(amount - 3)
    coins.append(3)
    return coins
```

- Recursion
- The Coin Problem
- Hanoi Towers

Hanoi Towers

Problem

There are 3 sticks with n discs sorted by size on one of the sticks. The goal is to move all n disks to another stick subject to 2 constraints: (1) move one disk at a time, and (2) don't put a larger disk over a smaller one.



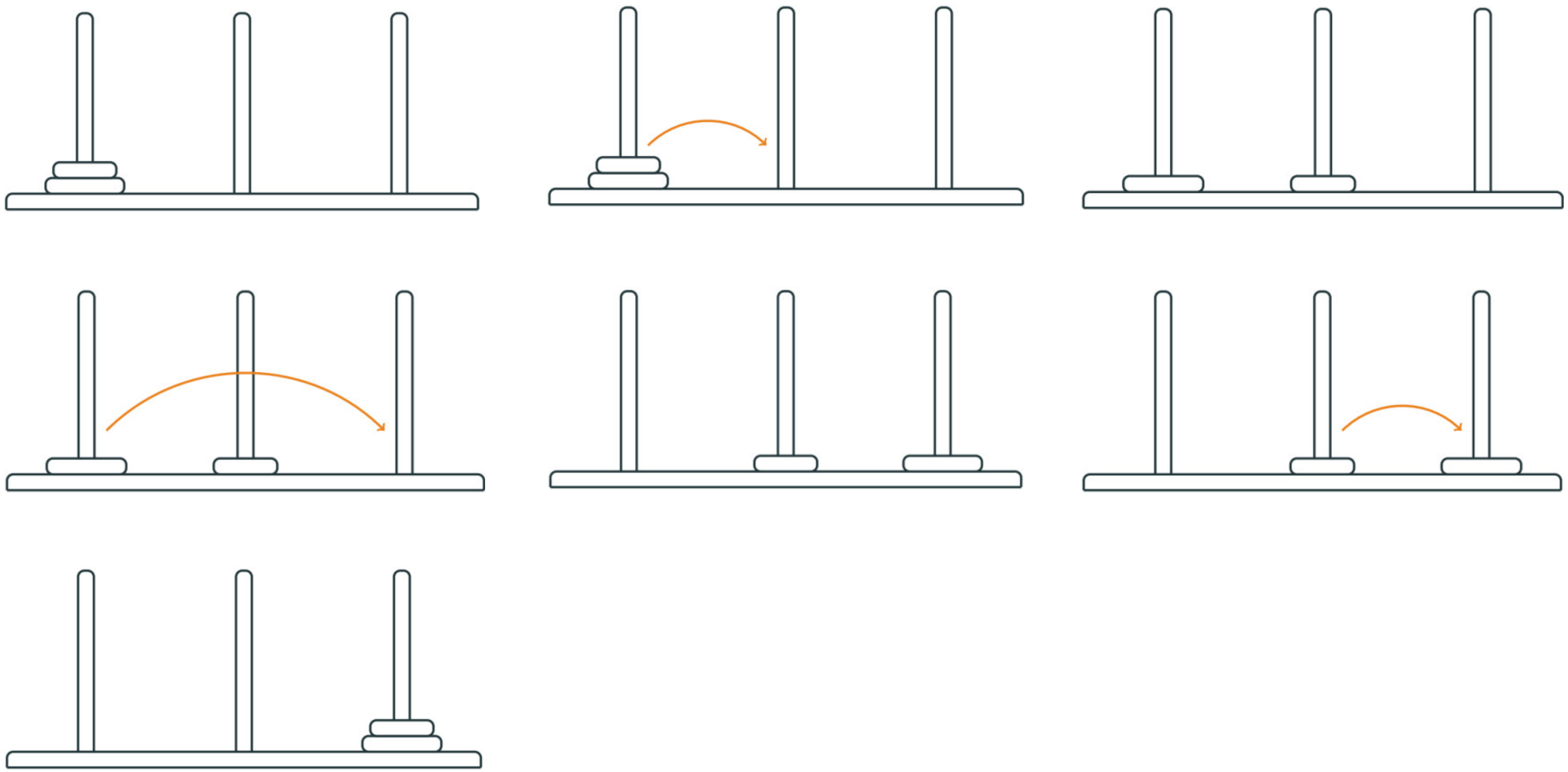
https://en.wikipedia.org/wiki/Tower_of_Hanoi#/media/File:Tower_of_Hanoi.jpeg

Can it be done?

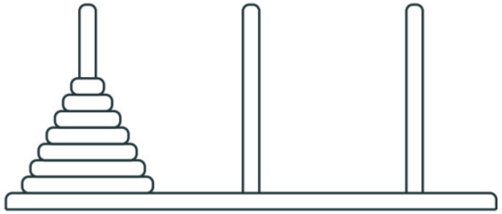
- For what value of *n* is this possible?
  - For all!
- How can we be sure?
  - Design a recursive program that will solve the puzzle for every value of *n*

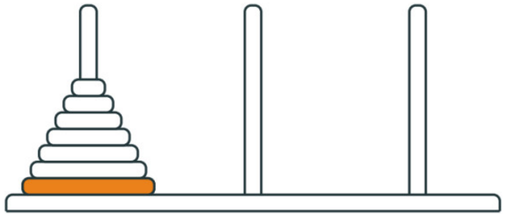
Simplest scenario, n=1 disk

n=2 Disks Scenario



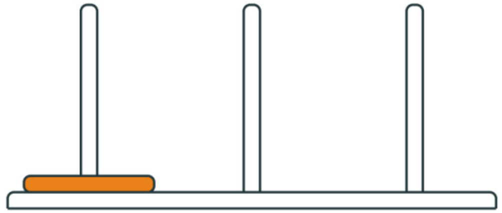
How about for *n* disks? Let's speculate



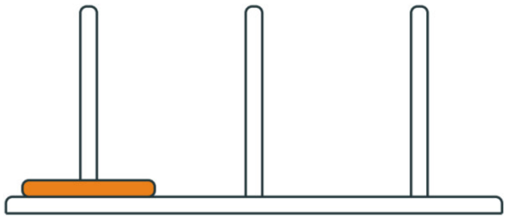


1

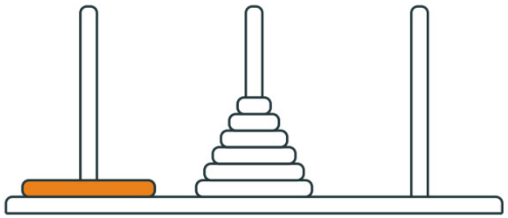
Consider largest disk



$C_1 \rightarrow$ When can we move it?
When there are no other disks
on the main stick



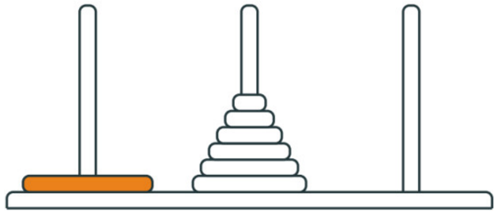
Where can we move it?
$C_2 \rightarrow$ Only to an empty stick



2

All other disks are on another
disk

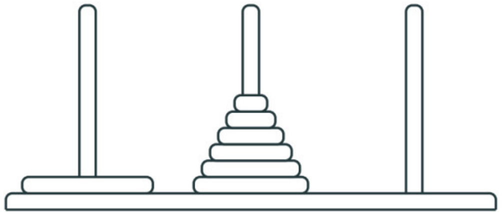
$C_1 \neq C_2$



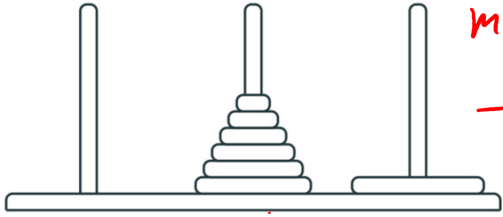
$n-1$

This is same problem,
just n-1 disks!

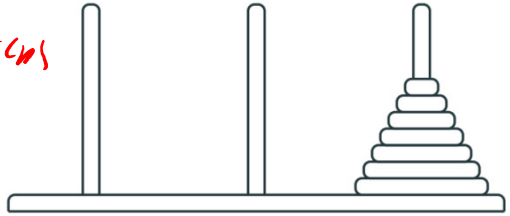
n-1 disks? Let's do it recursively



Move n-1 disks recursively



Move largest disk to free stick



Move n-1 disks recursively until done

Summary

- A solution has been proposed for all values of n:
  - Base scenario: possible for *disks = 1*
    - Therefore, it is possible for *disks = 2*
    - Therefore, it is possible for *disks = 3*, … until *disks = n*
  - Or putting it other way, It is possible to solve *n*, solve first for *n-1*
    - Therefore, it is possible for *n – 1*
    - Therefore, it is possible for *n – 2*; … until n = 1
- recursion & induction
  - *recursion*: method of defining/implementing something
  - *induction*: mathematical method of proving something

Thank you.