Intro to Algo: **More recursion**

Contents

Lab

Fred sells bunches of flowers at the local shopping centre. One day Fred's boss, Joe, tells Fred that at any time during the day he (Joe) will need to know:

- how many bunches of flowers have been sold
- what was the value of the most expensive bunch sold
- what was the value of the least expensive bunch sold
- what is the average value of bunches sold

Lab

Processes:

- Sell a bunch of flowers
- Maintain a counter of bunches sold
- Store the value of the most expensive bunch
- Store the value of the least expensive bunch
- Maintain an average value
- and in order to find the average price we will need to keep a total of the values

Decisions:

- Find_Highest
- Find_Lowest

Variables:

- Bunches_Sold
- Price
- Highest_Price
- Lowest_Price
- Average_Price
- Total_value

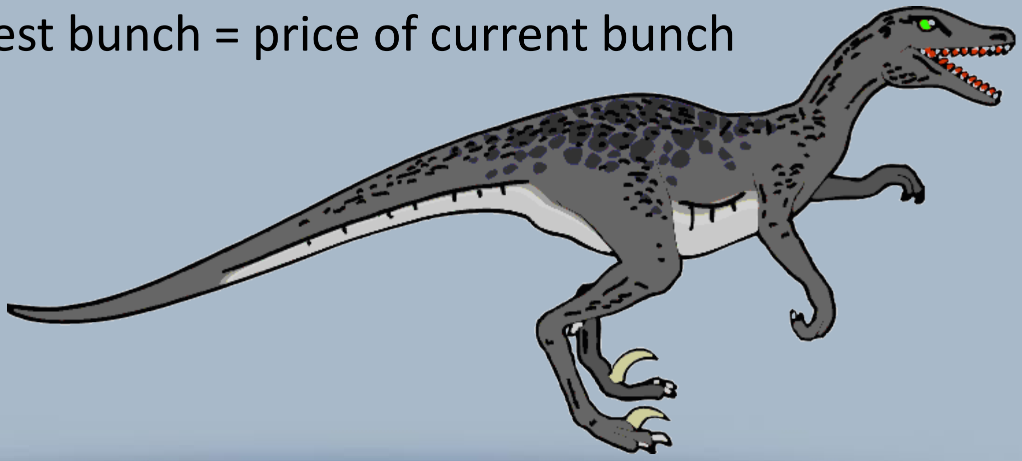
Loops:

- Selling flowers

Lab

While there are bunches of flowers to sell

1. sell a bunch
2. increment the bunches sold counter
3. calculate the average so far
4. if price of current bunch is greater than price of highest bunch then make price of highest bunch = price of current bunch
5. if price of current bunch is less than price of cheapest bunch then make price of cheapest bunch = price of current bunch



end of while

Reminder: What is Recursion?

When one function calls ITSELF directly or indirectly.

When should I use Recursion?

- If the algorithm has a **base case**
- If a problem is iterative
- If the problem gets progressively smaller

Reminder: Recursive Factorial

- What is the algorithm?

Reminder: Recursive Factorial

- Remember this …. 4!

```
Factorial (n)
   if n=1 or n=0
         return 1
   else
         return n*Factorial(n-1)
```

Call Stack

<Pop!> 1	1
2*Factorial (1)	2*1
3*Factorial (2)	3*2
4*Factorial (3)	4*6
Factorial (4)	= 24

Recursive Power

- Write the recursive algorithm for Power.
- Illustrate the call stack for 2^5

Recursive X ^ Y

```
Power(x,y)
   if (y=0) then
      return 1
   else
      return x*Power(x,y-1)
```

Stack power(2,5) :-

~~power(2,0)~~	*pop* return 1
2*~~power(2,0)~~	=2*(1)=2
2*~~power(2,1)~~	=2*(2)=4
2*~~power(2,2)~~	=2*(4)=8
2*~~power(2,3)~~	=2*(8)=16
2*~~power(2,4)~~	=2*(16)=32
return (32)	

Recursive Euclid's Algorithm

```
gcd(a, b)
  if (b = 0) then
     return a
else
   return gcd(b,a mod b)
```

Stack `gcd(72,30) :-`

~~gcd(6,0)~~ *pop* return (6)
~~gcd(12,6)~~ = 6
~~gcd(30,12)~~ = 6
return (6)

Illustrate this recursive algorithm using a call stack – **gcd (72, 30)**

Write an *iterative* GCD algorithm

An iterative solution ….

```
gcd(a, b)
if b=0 then
     return a
else
     while b!=0
          rem = a mod b
          If rem=0
               return b
          else
               a=b
               b=rem
```

Does this work?
Test it with (72, 30)

Euclid’s Algorithm

Or…

```
function gcd(a, b)
  while a mod b>0
  do
    R := a mod b
    a := b
    b := R
  done
  return b
End function
```

Fibonacci

- The Fibonacci Sequence is the series of numbers:
  - 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- The next number is found by adding up the two numbers before it.

Fibonacci

$n =$	0	1	2	3	4	5	6
$x_n =$	0	1	1	2	3	5	8

Example: term 6 would be calculated like this:

$$\mathbf{x_6 = x_{6-1} + x_{6-2} = x_5 + x_4 = 5 + 3 = 8}$$

- Can you calculate the following?
  - Term 7
  - Term 9

Fibonacci

- What is the base case?
- What is the recursive call?

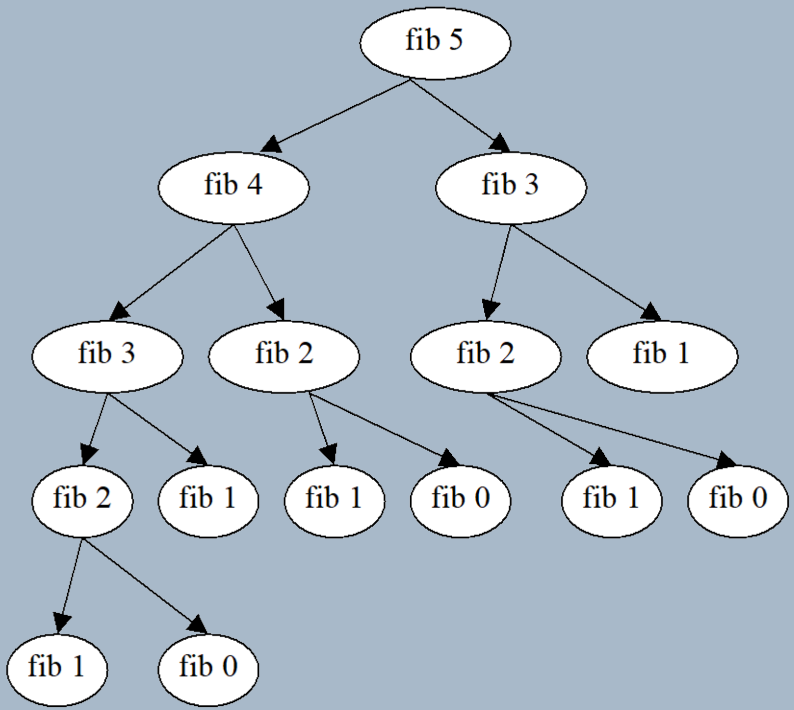
Recursive Fibonacci

- Let’s try this with n=5

```
fibonacci(n)
if (n=0 or n=1)
    return n
else
    return fibonacci(n-1) + fibonacci(n-2)
```

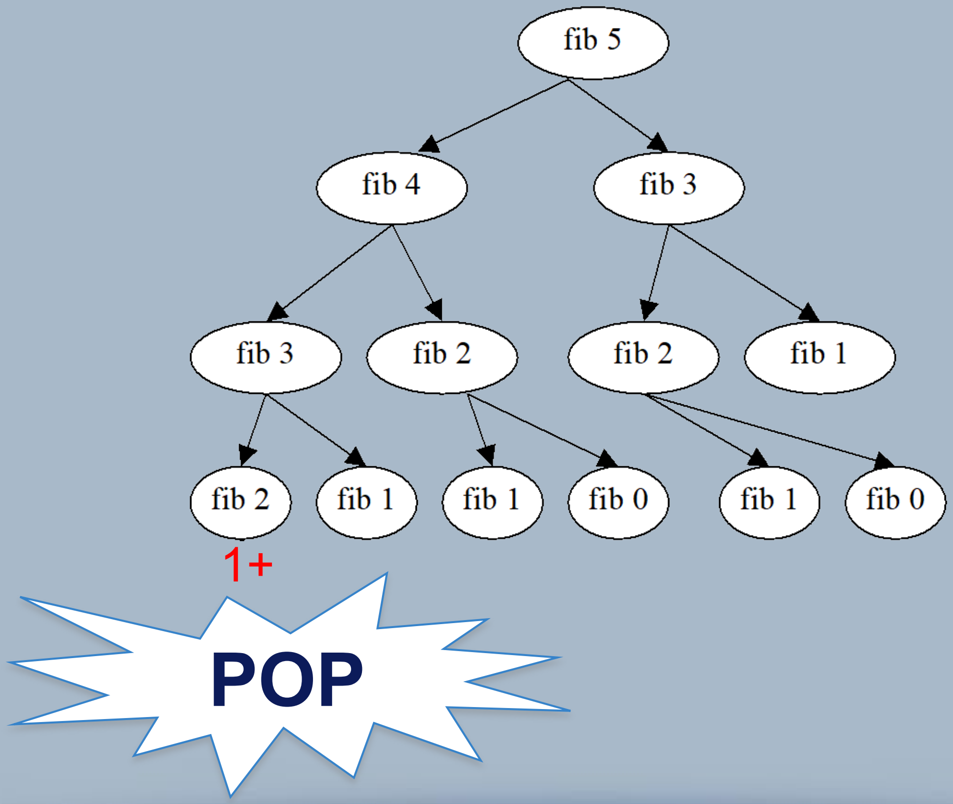
Fibonacci

- Let's try this with n=5



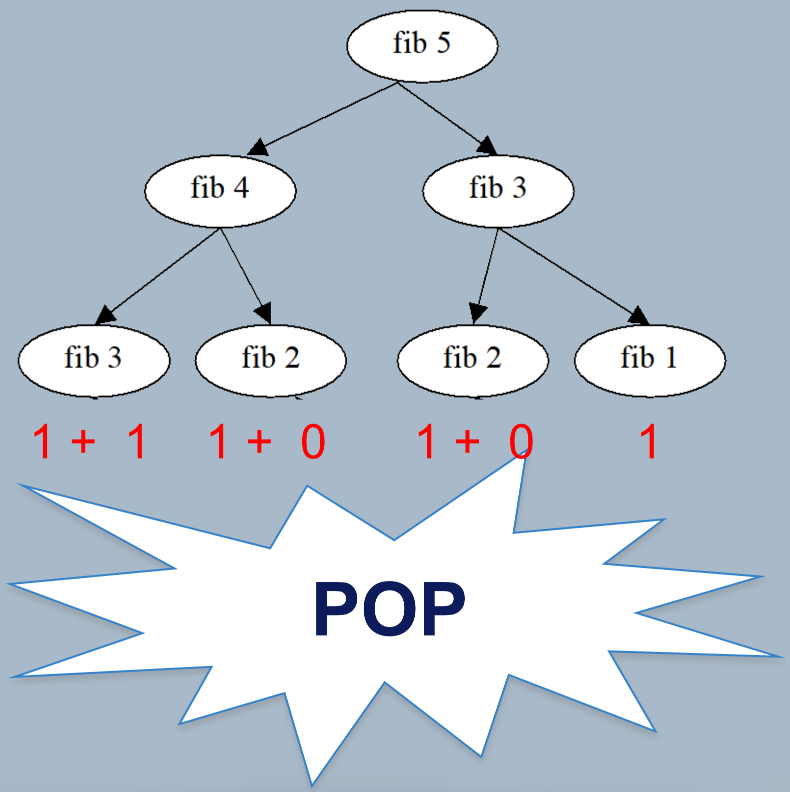
Fibonacci

- Let's try this with n=5



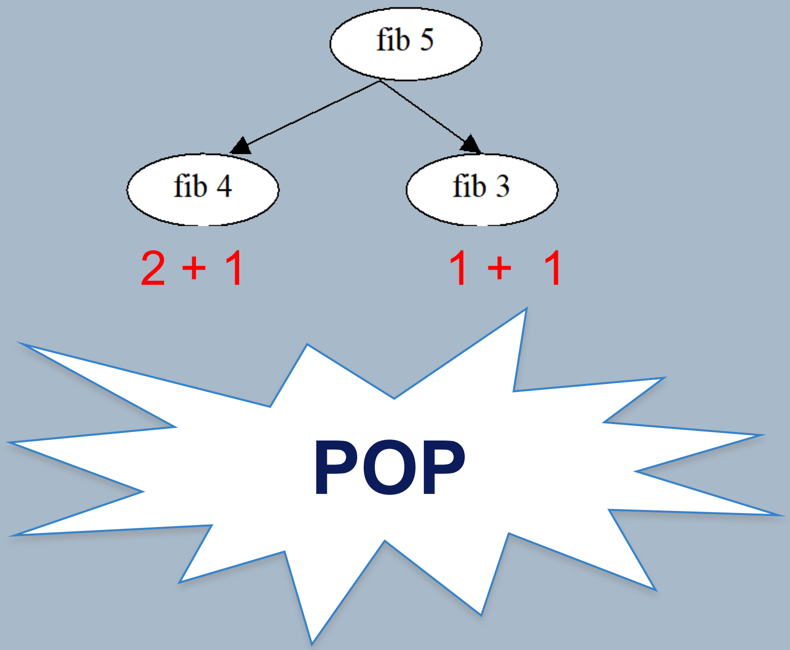
Fibonacci

- Let’s try this with n=5



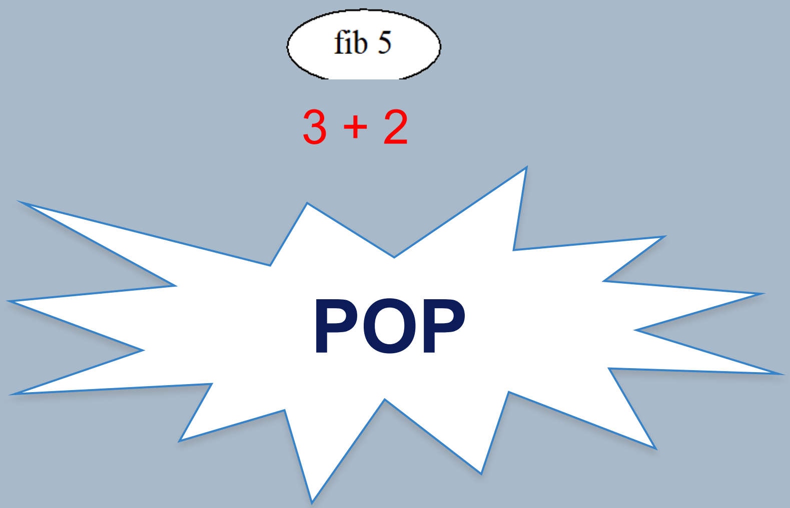
Fibonacci

- Let's try this with n=5



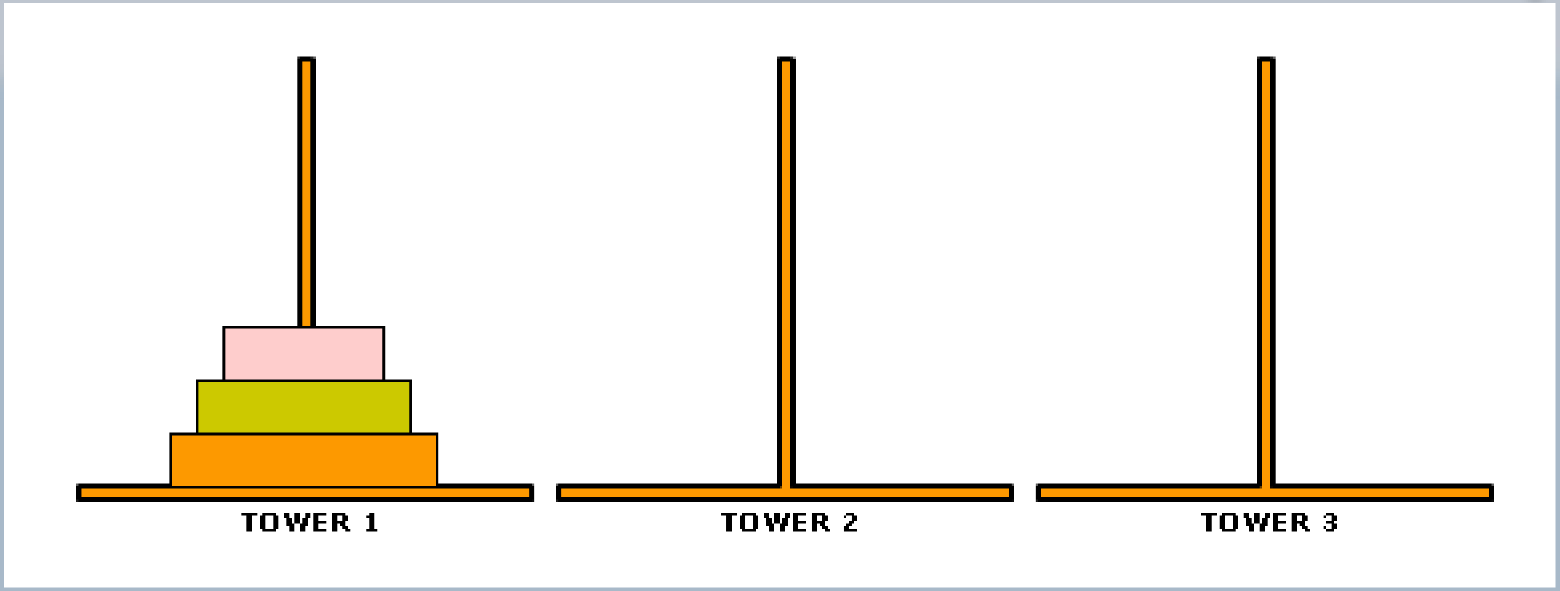
Fibonacci

- Let’s try this with n=5



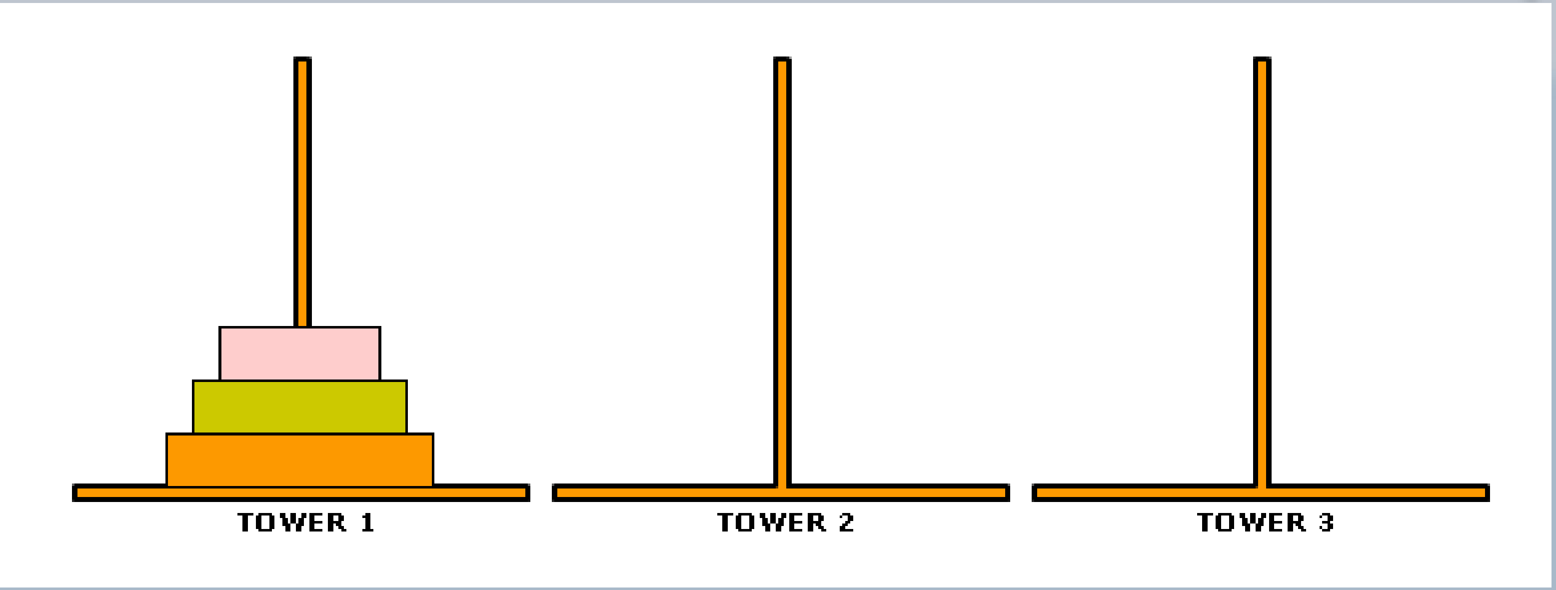
A more complicated use of recursion – *The Towers of Hanoi*

The Towers of Hanoi



- **Rules**
  - Move all disks to Tower 3
  - Only one disk can be moved at a time
  - A disk can never be put on a smaller disk

The Towers of Hanoi

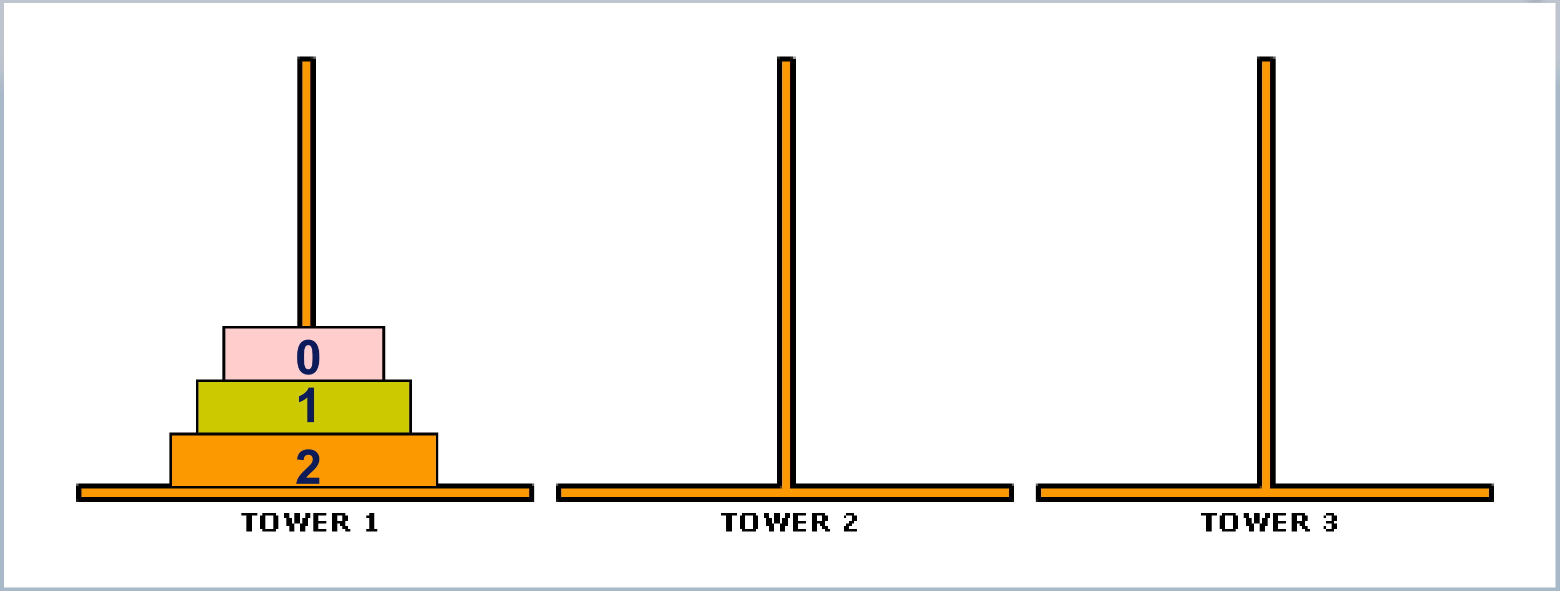


- What is the problem?
  - Move the largest disk, disk2, to Tower 3
  - Move the middle disk, disk1, to Tower 3
  - Move smallest disk, disk0, to Tower 3

The Towers of Hanoi

- Why is this suitable for **recursion**?
  - Because there is a base case
  - The problem is iteratively getting smaller

The Towers of Hanoi



Source

Spare

Dest

Disks = 2 (0,1,2)

The Towers of Hanoi: A recursive algorithm

moveTower (disks, source, dest, spare)

If disk = 0 ***if smallest disk*

Move disk from source to dest ***moves smallest disk*

else

moveTower (disk-1, source, spare, dest)

move disk from source to dest ***moves other 2 disks*

moveTower (disk-1, spare, dest, source)

Thank You !

OLLSCOIL TEICNEOLAÍOCHTA BHAILE ÁTHA CLIATH
TU DUBLIN
TECHNOLOGICAL UNIVERSITY DUBLIN