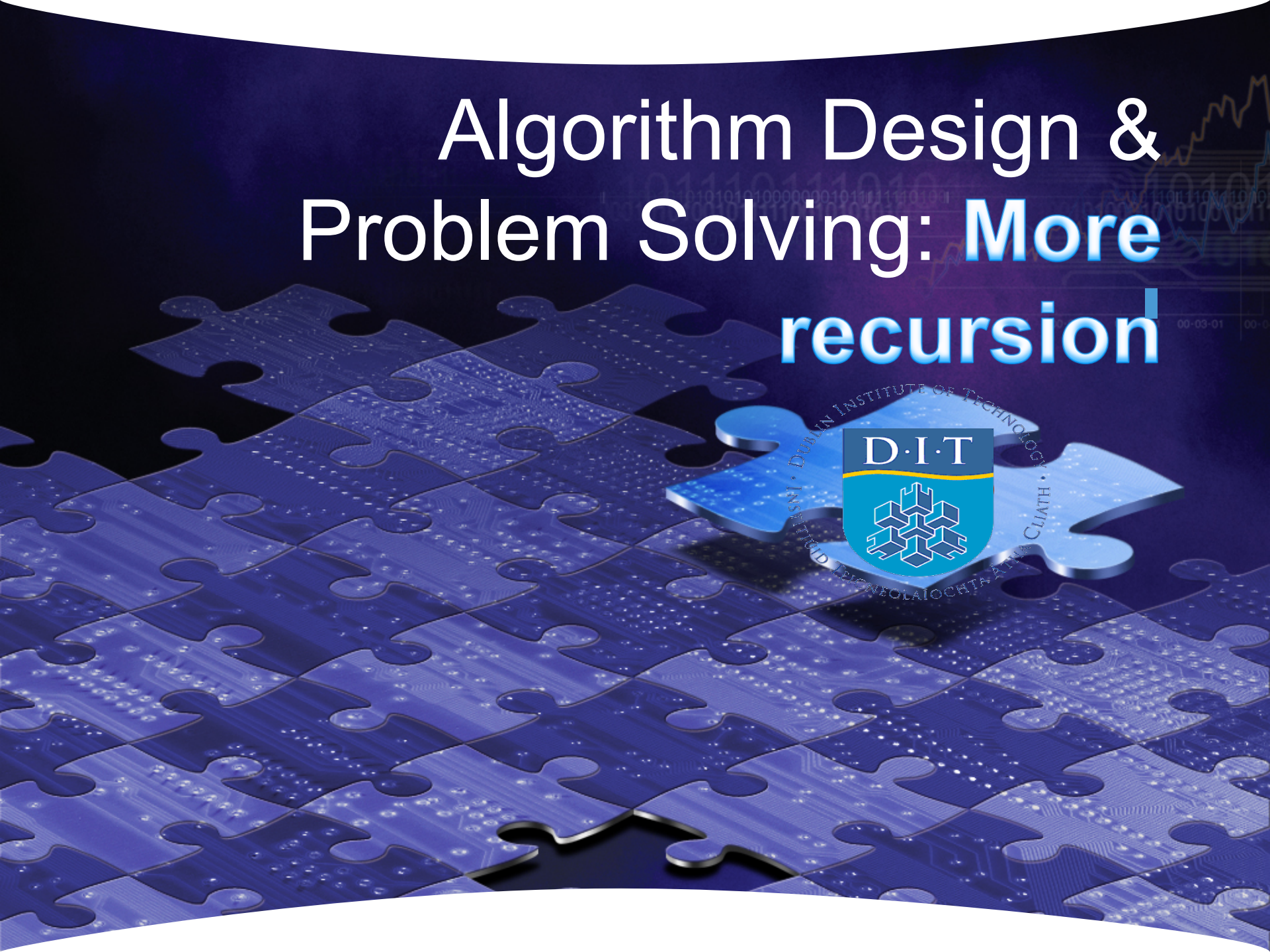
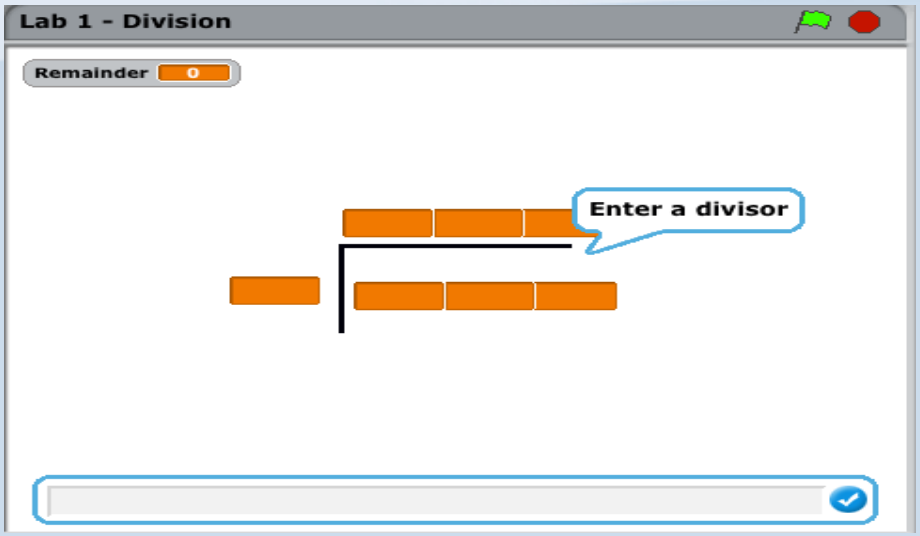
Algorithm Design & Problem Solving: More recursion

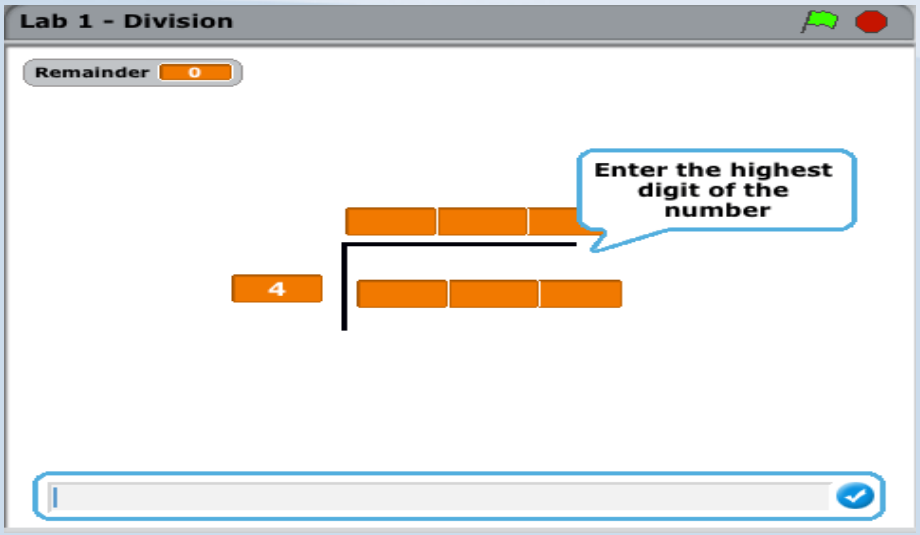


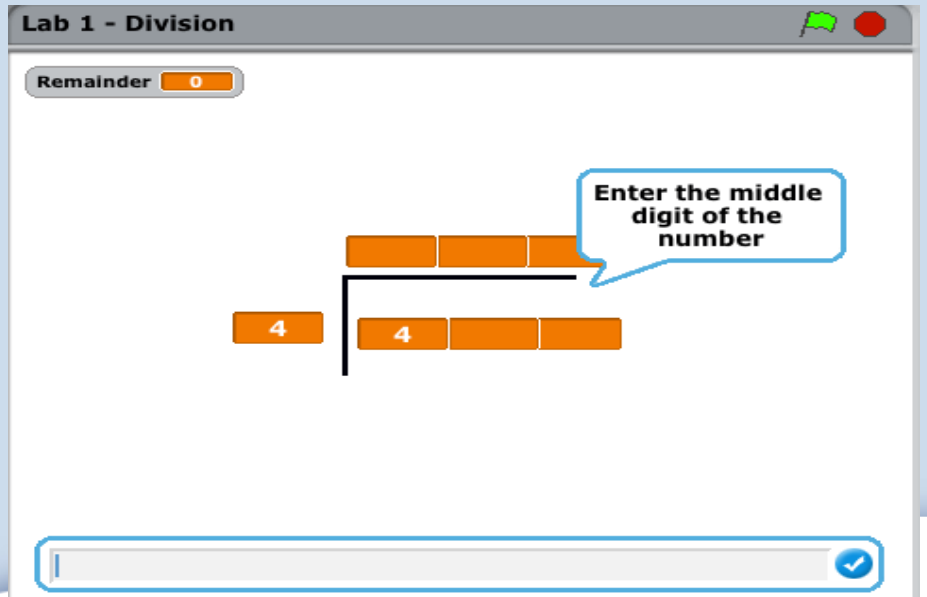
Contents

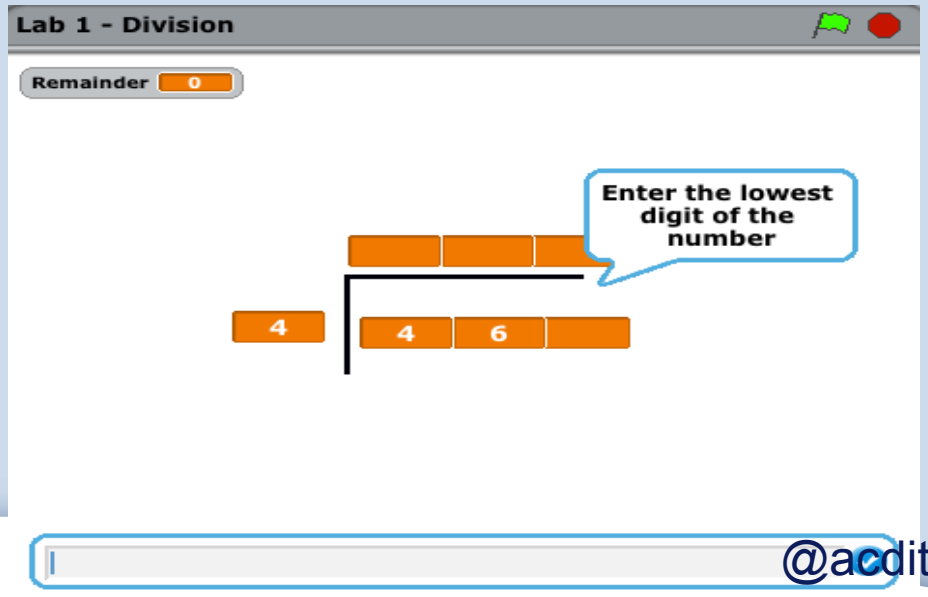
1. A look at Monday's lab
2. Review of recursion
3. Recursive Fibonacci
4. Recursive Tower of Hanoi

Last Monday's Lab









```
Enter divisor
Enter highest digit                                  *of number to be divided*
Enter middle digit                                   *of number to be divided*
Enter smallest digit                                 *of number to be divided*
If highest digit < divisor
        highest quotient = 0                         *the highest number of the answer*
        remainder = highest digit * 10
Else
        highest quotient = highest digit /divisor
        remainder = (highest digit mod divisor)*10
middle digit = (middle digit + remainder)
```

```
If middle digit < divisor
        middle quotient = 0                          *the middle number of the answer*
        remainder = middle digit * 10
Else
        middle quotient = middle digit /divisor
        remainder = (middle digit mod divisor)*10
smallest digit = (smallest digit + remainder)
```

```
If smallest digit < divisor
        smallest quotient = 0                        *the smallest number of the answer*
        remainder = smallest digit
Else
        smallest quotient = smallest digit /divisor
        remainder = smallest digit mod divisor
```

Last Monday's Lab

- Let's test with the 466/4:
  - Divisor =
  - Highest digit =
  - Middle digit =
  - Lowest digit =
- What is:
  - Highest quotient =
  - Middle quotient =
  - Lowest quotient =

Last Monday's Lab

- Any problems with this algorithm?
- Can it be made more efficient?
- How is Scratch as a programming language?

Last Monday's Lab

- How did you get rid of the decimal problem?

```
Remainder = number1 mod Divisor
Quot1 = (Number1-Remainder)/Divisor
Remainder = Remainder * 10
```

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

- Remember this …. 4!

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

Call Stack

BASE CASE REACHED 4*3*2*1
4*3*2*Factorial(1)
4*3*Factorial(2)
4*Factorial(3)
Factorial(4)

Recursive Factorial

- Remember this from last week …. 4!

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

4 * 3 * 2 * 1 = 24

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:

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

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

Fibonacci

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

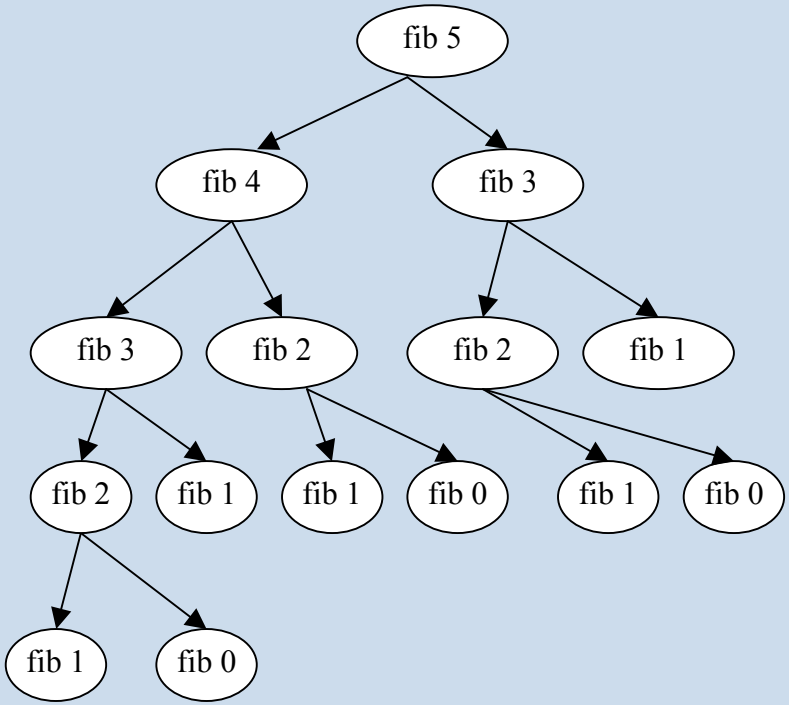
Recursive Fibonacci

- 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)
```

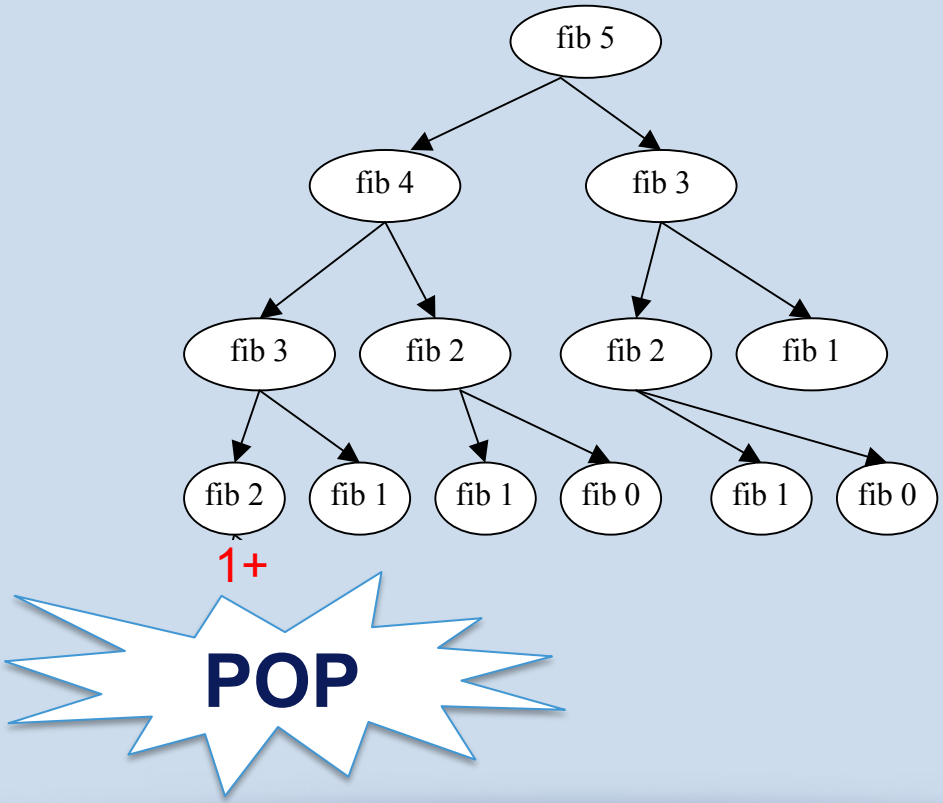
Fibonacci

- Let's try this with n=5



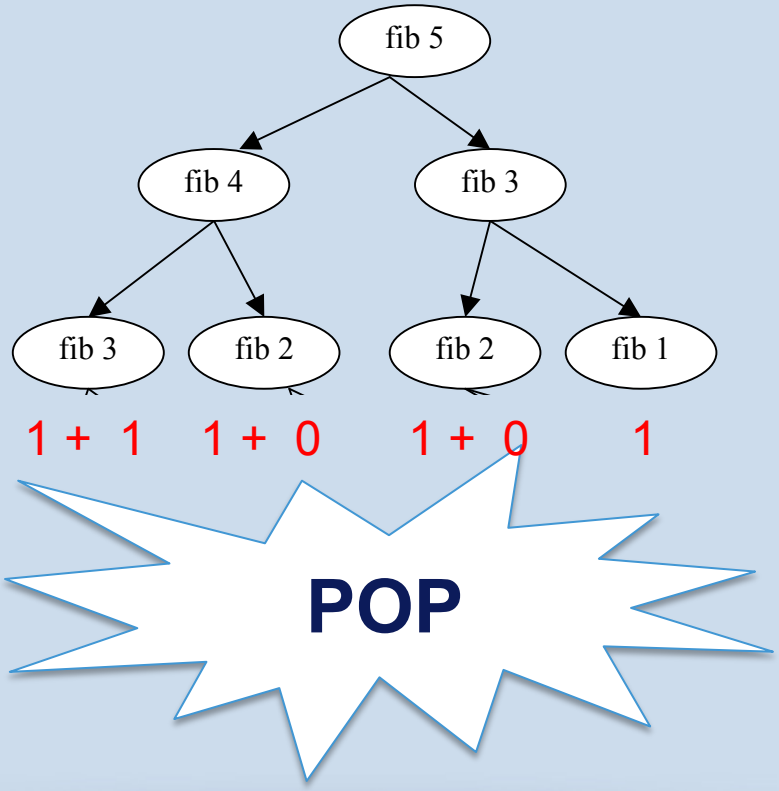
Fibonacci

- Let's try this with n=5



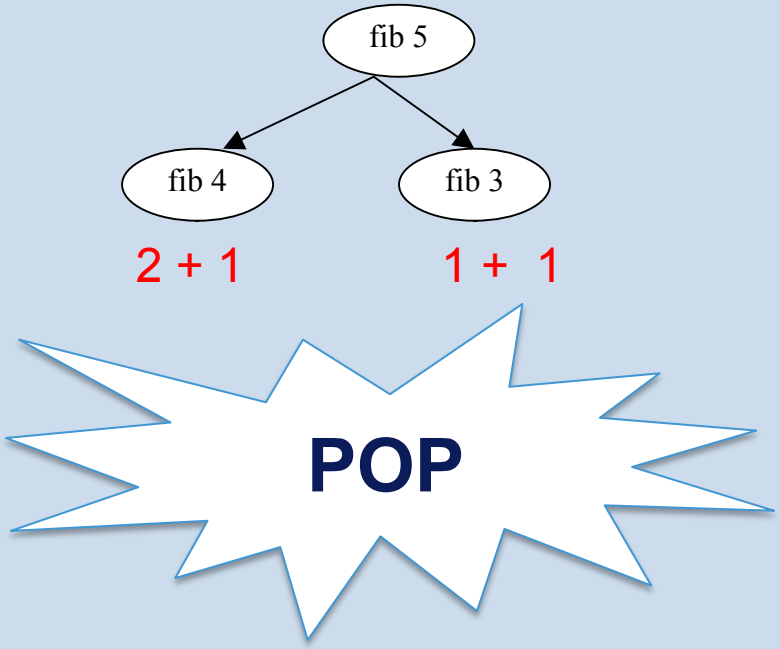
Fibonacci

- Let's try this with n=5



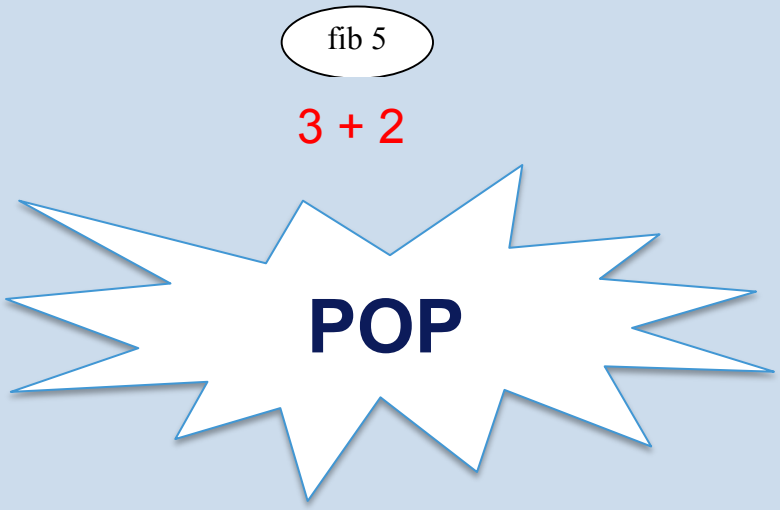
Fibonacci

- Let's try this with n=5



Fibonacci

- Let's try this with n=5



Recursive Euclid's Algorithm

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

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)

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*

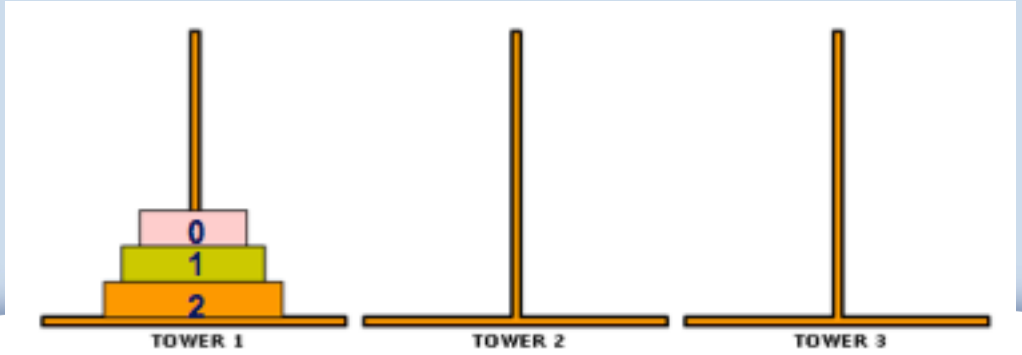
Move disk from source to dest ***moves smallest disk*

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 !


D·I·T
Dublin Institute of Technology
Institiúid Teicneolaíochta Átha Cliath