Introduction to Programming

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Recursion

The Concept of Recursion

CSC1

Hussein Suleman
Department of Computer Science

Recursion

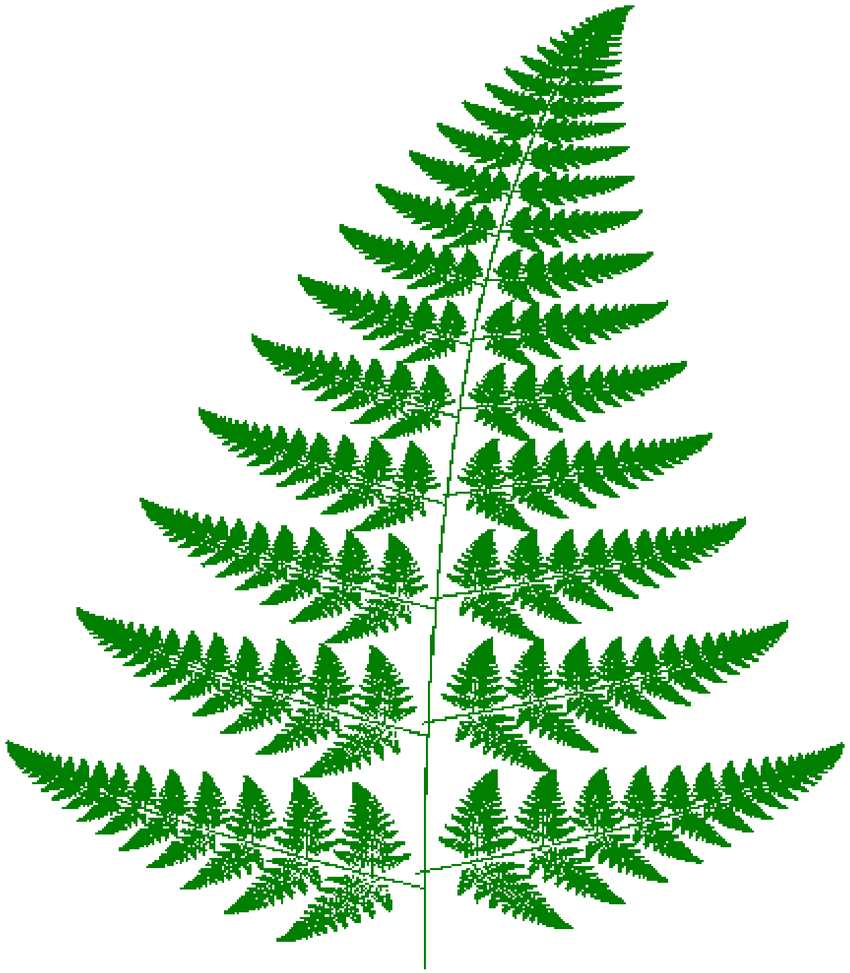
Hussein Suleman
Department of Computer Science
University of Cape Town

Concept: Recursion

- *It was a dark and stormy night, and the head of the brigands said to Antonio:“Antonio, tell us a tale”. And so Antonio began:*
  - “It was a dark and stormy night and the head of the brigands said to Antonio, “Antonio, tell us a tale”. And so Antonio began:
    - “It was a dark and stormy night and the head of the brigands said to Antonio, “Antonio, tell us a tale”. And so Antonio began:
      - “It was a dark and stormy night and the head of the brigands said to Antonio, “Antonio, tell us a tale”. And so Antonio began:
        - “It was a dark and stormy night and the head of the brigands said to Antonio, “Antonio, tell us a tale”. And so Antonio began:
          - “It was a dark and stormy night and ….

Concept: Recursion

- The fern leaf



Concept: Recursion

- Selfies with mirrors

Recursive definitions:

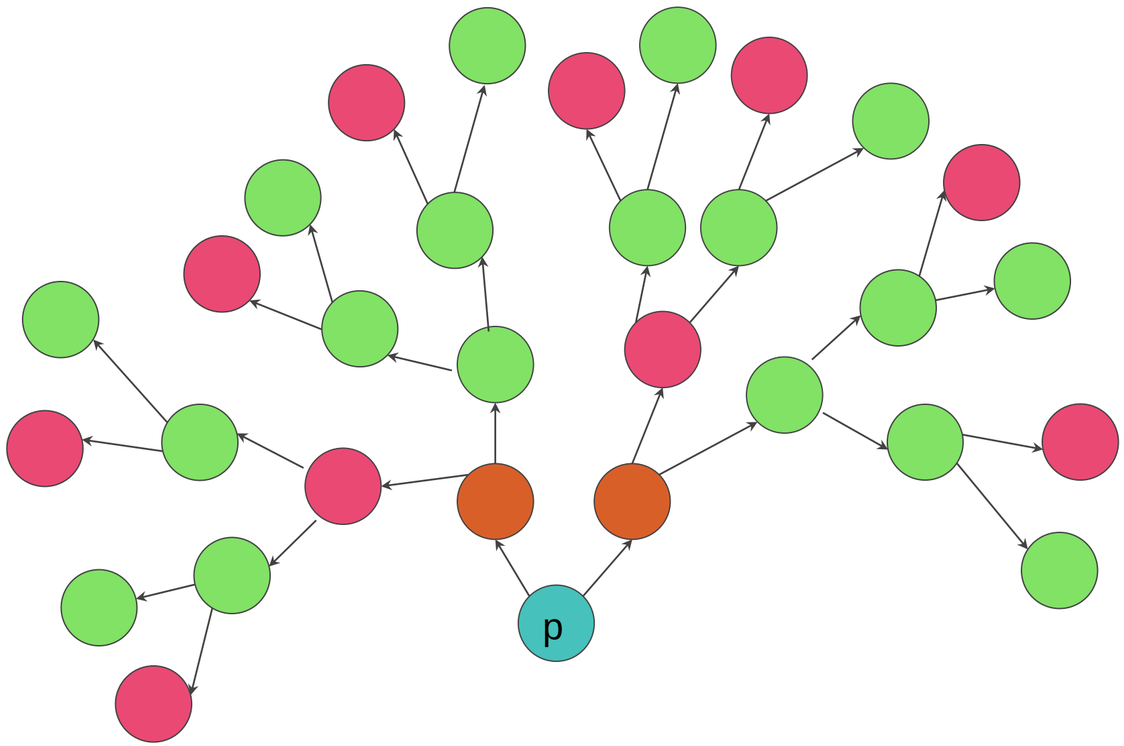
A description of something that refers to itself is called a **recursive definition**.

e.g.

GNU: GNU is Not Unix

Recursive definitions

- A recursive definition of the ancestors of person ***p***:
  - p's parents are p's ancestors (***base case***);
  - The parents of any ancestors of ***p*** are also the ancestors of p (***recursive step***).



Recursive definitions:

A description of something that refers to itself is called a **recursive definition**.

e.g.

- The **set of prime numbers** can be defined as the unique set of positive integers satisfying:
  - 1 is not a prime number
  - any other positive integer is a prime number if and only if it is not divisible by any **prime number** smaller than itself.

- But how is this useful to computer science …

Recursive Functions

- A function can **call itself**.
- A function that does this is a **recursive function**.

```
def brigand():
    print("It was a dark and stormy night, and the head of
the brigands said to Antonio:“Antonio, tell us a tale”. And
so Antonio began:")
    brigand()

brigand()
```

Recursive Functions

A ***recursive*** function is a function that includes a call to itself, based on the general problem-solving technique of breaking down a task into subtasks.

Recursion can be used whenever **one subtask is a smaller version of the original task**.

Defining recursive functions

- A recursive function **calls itself.**

- Recursive functions have 2 key elements:
  - one or more **recursive calls.**
  - **stopping condition**, or **base case**, where no recursion is required.

- Recursion is the equivalent of mathematical induction!

Example: Recursive Functions

- Now, with a **base case**

```
def brigand(n):
    if n==0:
        print(“and he immediately finished.")
    else:
        print("It was a dark and stormy night, and the head
of the brigands said to Antonio:“Antonio, tell us a tale”.
And so Antonio began:")
        brigand(n-1)
```

UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

SCHOOL OF IT

Introduction to Programming

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Recursion

Factorial

CSC1

Hussein Suleman
Department of Computer Science

Factorial

Hussein Suleman
Department of Computer Science
University of Cape Town

SCHOOL OF IT

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Poll

Why is recursion important?

- A – `define problems in terms of smaller problems`
- B – `elegant and natural solutions`
- C – `analogous to mathematical induction`
- D – `all of the above`

Solution

Why is recursion important?

- A – `define problems in terms of smaller problems`
- B – `elegant and natural solutions`
- C – `analogous to mathematical induction`
- D – `all of the above` 

Poll

What are the features present in most recursive functions?

- A – `function that is nested, nonlocal`
- B – `function that calls itself, base case`
- C – `global variables, main calling functions`
- D –`modules, def main ()`

Solution

What are the features present in most recursive functions?

- A – `function that is nested, nonlocal`
- B – `function that calls itself, base case` 
- C – `global variables, main calling functions`
- D –`modules, def main ()`

Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n$$

recursive call

$$n! = n \times (n-1)! \quad \text{for n>=1}$$

$$0! = 1$$

stopping condition, or base case

Recursion: Factorial - derivation

We know:

n!= 1 x 2 x 3 x 4 … (n-2) x (n-1) x n

Then,

(n-1)!= 1 x 2 x 3 x 4 … (n-2) x (n-1)

Thus, n! = n x (n-1)!

This is a recursive definition in terms of a smaller problem.

All, we then need is a stopping condition, and n=0 works since factorials are defined for natural numbers.

Recursive Factorial function

```
def factRec(n):
    if n==0:
        return 1  #base case - ends recursion
    else:
        return n*factRec(n-1)
           #recursive call - does a little work and uses the
results from smaller version of same problem
```

function definition – must have parameter

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n$$ for n>=1

$$n! = (n-1)! \times n$$ # recursive call

$$0! = 1$$ # stopping condition, or base case

5!=5*4*3*2*1*1

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n \qquad \text{for n>=1}$$

$$n! = (n-1)! \times n \qquad \text{\# recursive call}$$

$$0! = 1 \qquad \text{\# stopping condition, or base case}$$

5!=5*4*3*2*1*1

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n \qquad \text{for n>=1}$$

$$n! = (n-1)! \times n \qquad \text{\# recursive call}$$

$$0! = 1 \qquad \text{\# stopping condition, or base case}$$

5!=5* 4* 3* 2*1

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n$$ for n>=1

$$n! = (n-1)! \times n$$ # recursive call

$$0! = 1$$ # stopping condition, or base case

5!=5* 4* 3*2

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n \qquad \text{for n>=1}$$

$$n! = (n-1)! \times n \qquad \text{\# recursive call}$$

$$0! = 1 \qquad \text{\# stopping condition, or base case}$$

5!=5* 4*6

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n$$ for n>=1

$$n! = (n-1)! \times n$$ # recursive call

$$0! = 1$$ # stopping condition, or base case

5!=5*24

Further explanation - Recursion: Factorial

Classic introductory example - Factorial function:

$$n! = 1 \times \ldots \times (n-1) \times n$$ for n>=1

$$n! = (n-1)! \times n$$ # recursive call

$$0! = 1$$ # stopping condition, or base case

120

Iterative solution: Factorial function

```
def fact(n):
    f=1
    for i in range(1,n+1):
        f=f*i
    return f
```

UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

SCHOOL OF IT

Introduction to Programming

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Recursion

Recursion Problems

CSC1

Hussein Suleman
Department of Computer Science

Recursion Problems

Hussein Suleman
Department of Computer Science
University of Cape Town

Problem 1

Write a recursive function to sum the first n positive integers.

```
#fill in the code for this function
def sum (n):
  #code goes here
  #what is the base case?
  #what is the recursive step?
```

More Recursion Problems

- Problem 2: Calculate $\mathbf{x}^{n}$
- Problem 3: Count the number of characters in a string.
- Problem 4: Count the number of words in a list.
- Problem 5: Search/replace characters in a string.
- Problem 6: Calculate Greatest Common Divisor (GCD).
  - Euclid's Algorithm.
- Problem 7: Draw a triangle of height n.
- Problem 8: Print out a list of values.
- Problem 9: Reverse a string.
- Problem 10: Find the sum of integers from ***m*** to ***n***.

Fun with recursion

100 bottles of beer on the wall,
100 bottles of beer.
If one of those bottles should happen to fall,
99 bottles of beer on the wall.

99 bottles of beer on the wall,
99 bottles of beer.
If one of those bottles should happen to fall,
98 bottles of beer on the wall.

98 bottles of beer on the wall,
98 bottles of beer.
If one of those bottles should happen to fall,
97 bottles of beer on the wall.

97 bottles of beer on the wall,
97 bottles of beer.
If one of those bottles should happen to fall,
96 bottles of beer on the wall…

Pitfall: Infinite Recursion

In our examples, the series of recursive calls eventually reached a call of the method that did not involve recursion (a stopping case).

If instead, every recursive call had produced another recursive call, then a call to that method would, in theory, run forever.

- This is called ***infinite recursion.***
- In practice, such a method runs until the computer runs out of resources, and the program terminates abnormally.

Infinite recursion example

```
def sumRec(n):
    if n==0:
        return 0
    else:
        return sumRec(n)+n    #logic error here
```

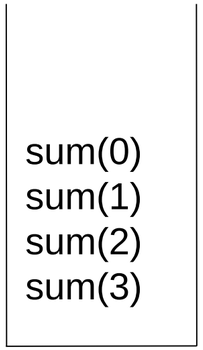
A Closer Look at Recursion

- When the computer encounters a recursive call, it must temporarily suspend its execution of a function
  - It does this because *it must know the result of the recursive call before it can proceed*
  - It saves all the information it needs to continue the computation later on, when it returns from the recursive call

- Ultimately, this entire process terminates when one of the recursive calls does not depend upon recursion to return.

What's Happening Inside Python

- Python keeps track of every function that has been called in a part of memory called a stack.
- This allows Python to return to the next point after a function call.
- The same holds for recursive functions.
  - e.g., sum(3) calls sum(2) calls sum(1) calls sum(0)



UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

SCHOOL OF IT

Introduction to Programming

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Recursion

Towers

CSC1

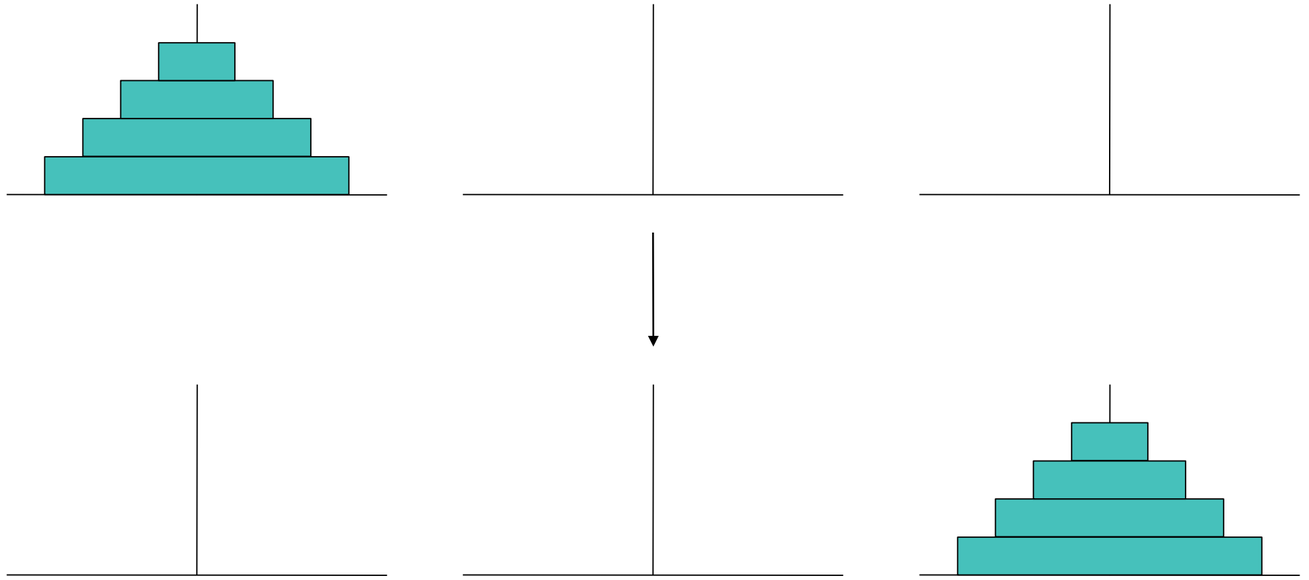
Hussein Suleman
Department of Computer Science

Towers

Hussein Suleman
Department of Computer Science
University of Cape Town

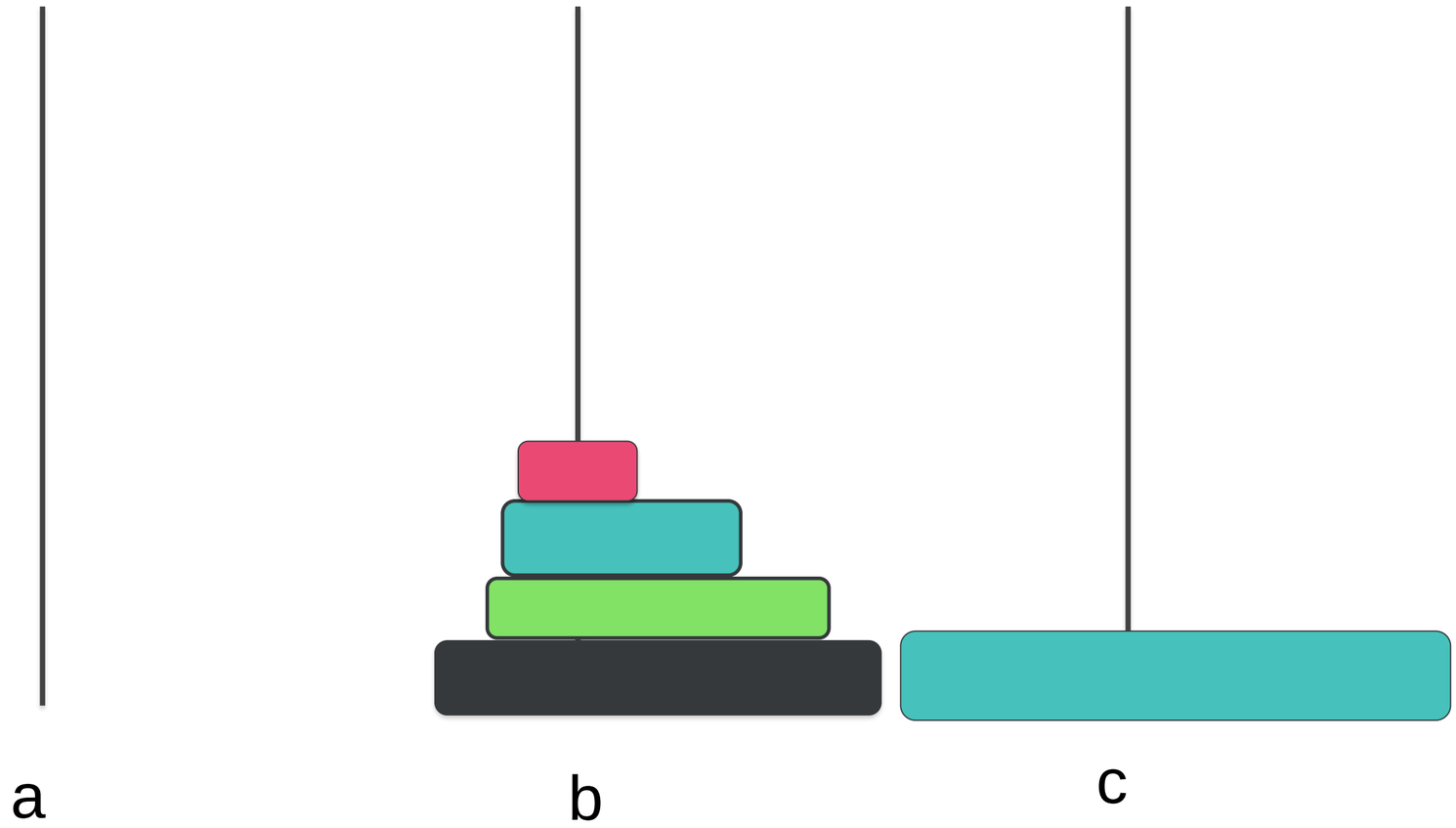
Towers of Hanoi

- Problem: Move a stack of discs one disc at a time from one tower to another, such that no disc may be placed on a larger disc.

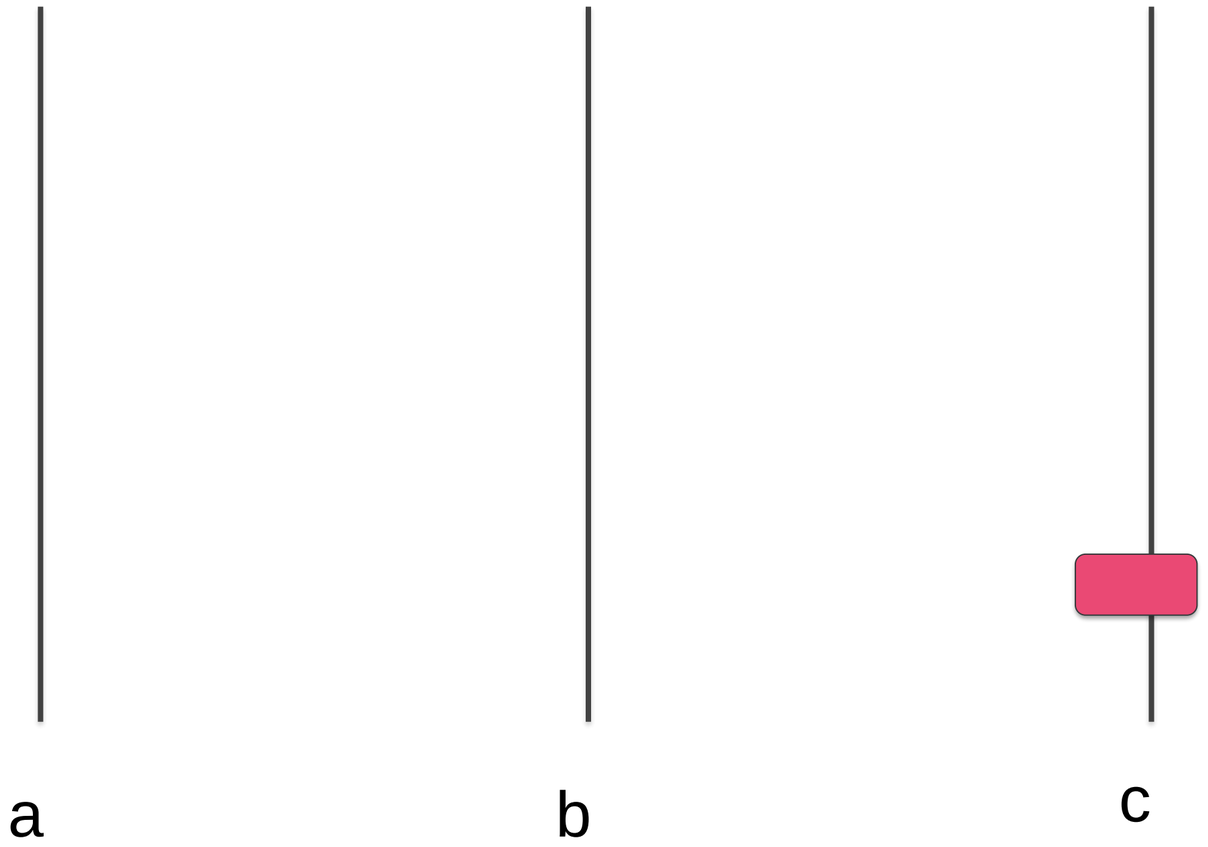


puzzle game invented in the late 1800s

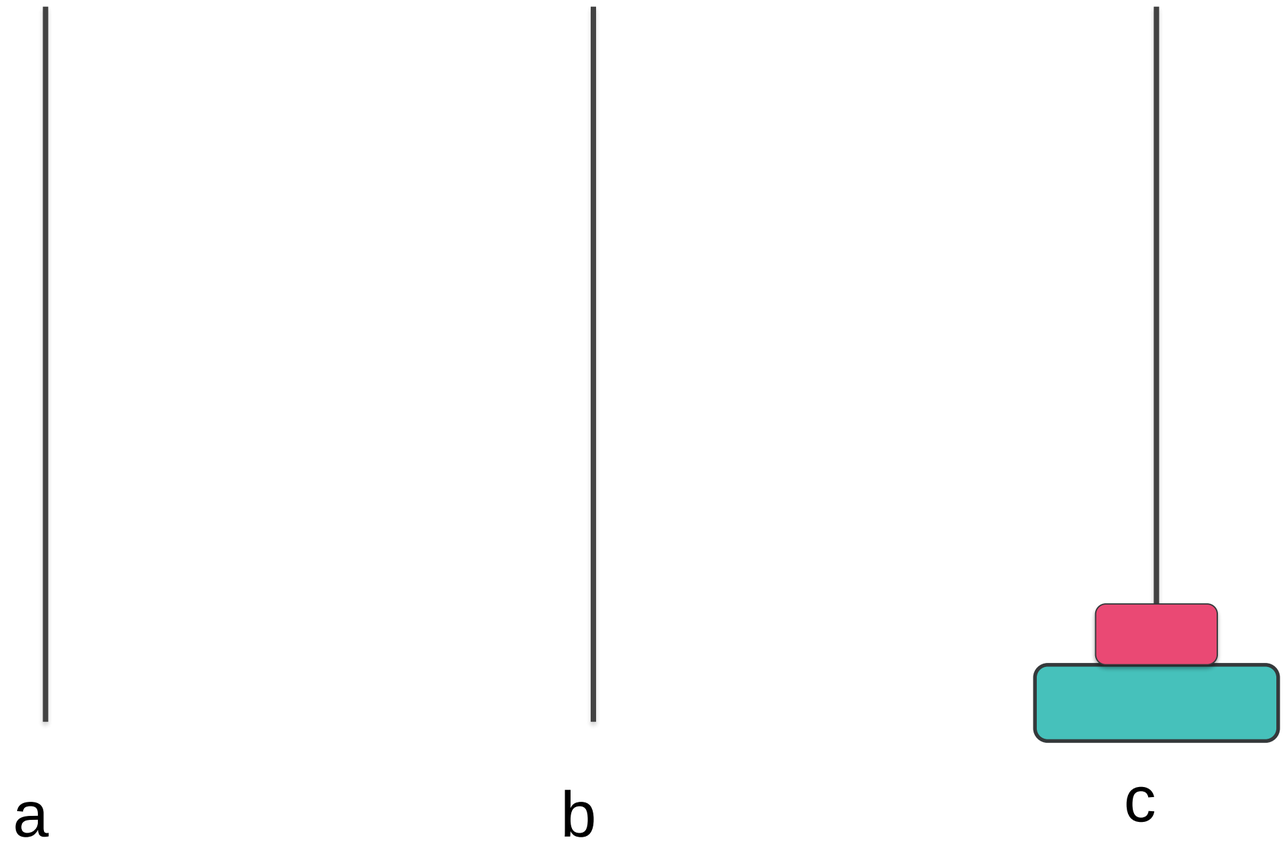
Hanoi



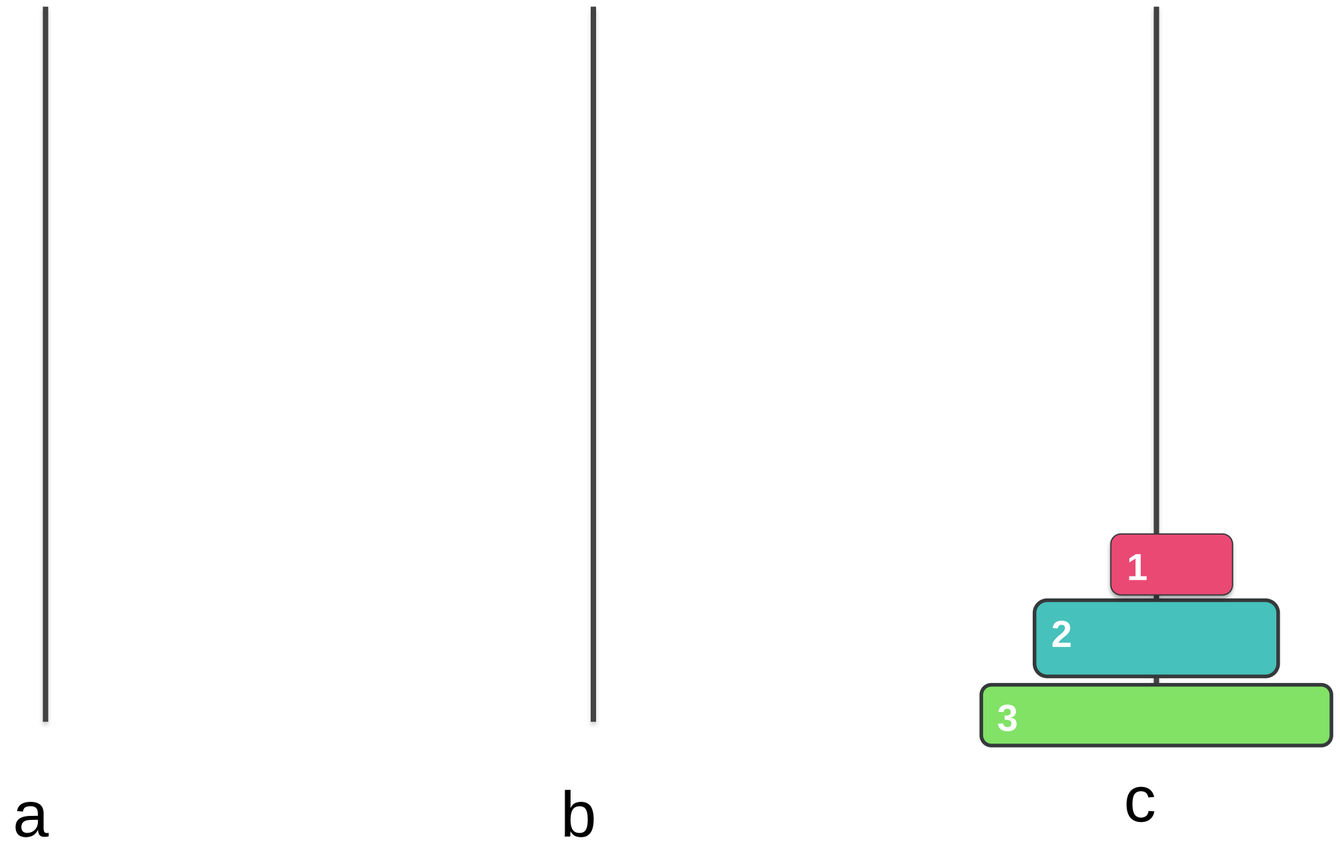
Hanoi: 1 disk



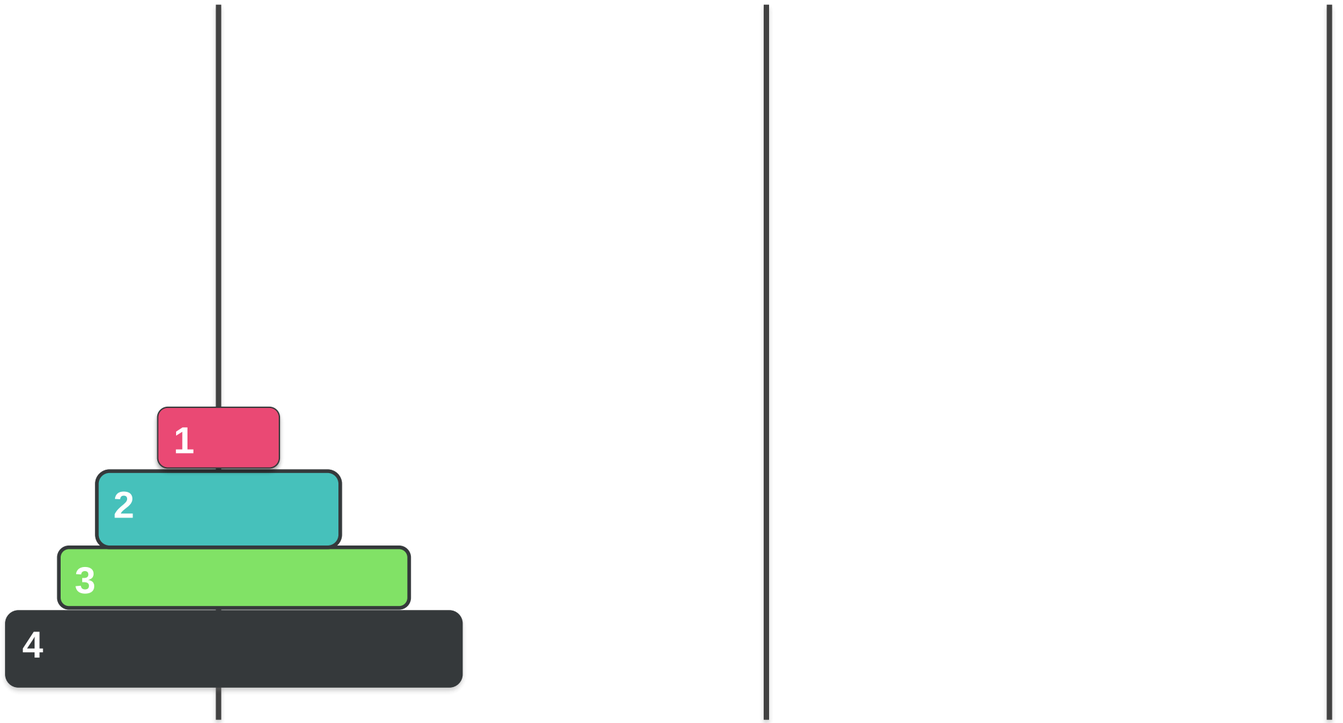
Hanoi: 2 disks



Hanoi: 3 disks



Hanoi: 4 disks



Towers of Hanoi

- Algorithm:
  - Move n-1 discs from source to spare tower
  - Move nth disc from source to destination tower
  - Move n-1 discs from spare to destination tower

  - Stop when no more discs … or one disc

https://www.youtube.com/watch?v=rVPuzFYlfYE

Exercise

- Write a program to show the steps in the Towers of Hanoi solution.

Exercise: what does this function compute?

```
def mystery(x):
    if x==1:
            return 2
    if x==0:
            return 1
  return 2*mystery(x-1)
```

Problem 11

- Draw a half-hourglass using a recursive function
  - Hourglass(“MARSUPIAL”)

```
MARSUPIAL
MARSUPIA
MARSUPI
MARSUP
MARSU
MARS
MAR
MA
M
MA
MAR
MARS
MARSU
MARSUP
MARSUPI
MARSUPIA
MARSUPIAL
```

UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

SCHOOL OF IT

Introduction to Programming

UNIVERSITY OF CAPE TOWN
IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

Recursion

Fibonacci and Performance

CSC1

Hussein Suleman
Department of Computer Science

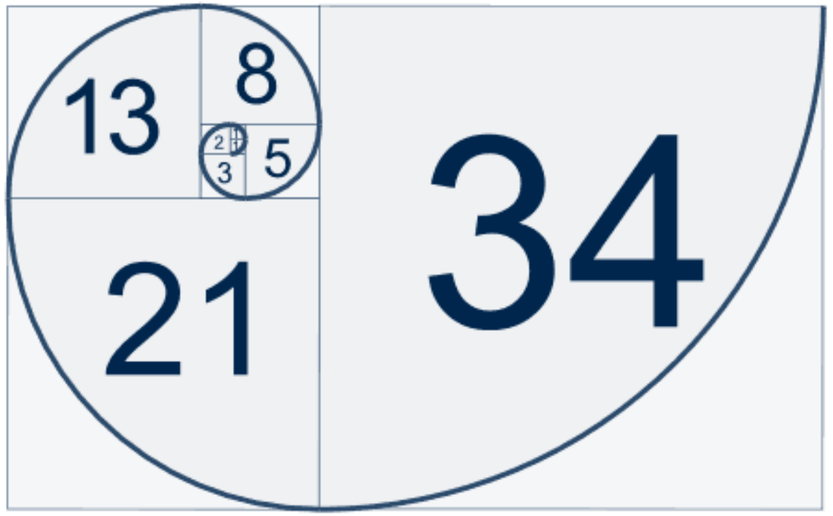
Fibonacci and Performance

Hussein Suleman
Department of Computer Science
University of Cape Town

Iterative Fibonacci numbers

- A Fibonacci number is the sum of the previous 2 Fibonacci numbers.
  - 0, 1, 1, 2, 3, 5, 8, 13, …

```
def fib(n):
    curr=1
    prev=1
    for i in range(n-2):
        curr,prev=curr+prev,curr
    return curr
```



Fibonacci in Australia

- In 1859, a farmer introduced 24 grey rabbits to remind him of home. At the time, the man wrote:
  - "The introduction of a few rabbits could do little harm and might provide a touch of home, in addition to a spot of hunting."
- For one pair, by 1900….(480 months)…
- Fib(480)

Problem 12: Recursive Fibonacci numbers

Recursive step:

Every Fibonacci number is the sum of the previous two numbers.

fib(n) = fib(n-2) + fib(n-1)

Base case:

The first 2 Fibonacci numbers are 0 and 1.

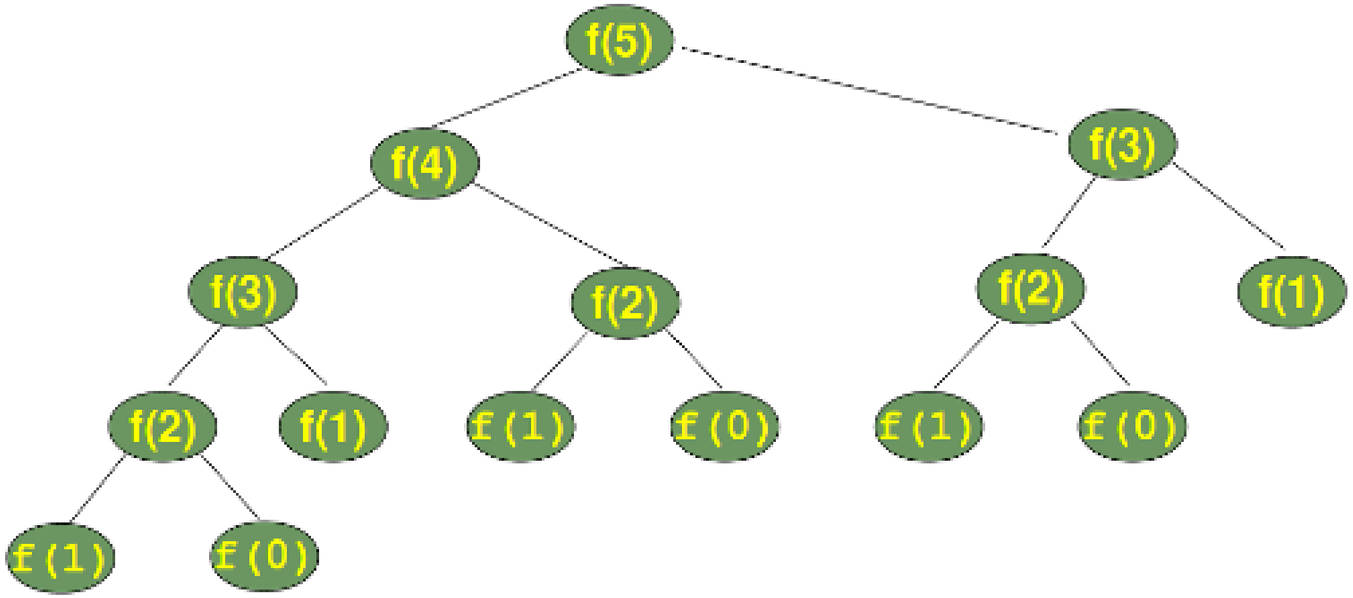
fib(0) = 0

fib(1) = 1

Recursive Fibonacci numbers

Elegant solution.

Not very efficient, because of many duplicate function calls



Recursion Versus Iteration

- Recursion is not absolutely necessary.
  - Any task that can be done using recursion can also be done in a non-recursive manner.
  - A non-recursive version of a method is called an ***iterative version***.
- An iteratively written method will typically use loops of some sort in place of recursion.
- A recursively written method can be simpler, but will usually run slower and use more storage than an equivalent iterative version.

Iterative version of prefix sum

```
#iterative definition
def  prefixSum(arr):
    tmp=[]
    for i in range(len(arr)):
        if i==0:
            tmp.append(arr[0]) #the first one is just a copy
        else:
            tmp.append(tmp[i-1]+arr[i]) #cummulative sum
    return tmp
```

Exercise: Recursive prefix sum

```
# recursive definition
def prefixSumRec(arr):
  # fill in the rest of the function

# test code
print (prefixSumRec ([1,5,2,7,4,6,5,3,6]))
```

Exercise: What does this function display?

```
def pattern(s,n):
    if n==0:
        return
    print(s)
    pattern('-'+s,n-1)
    print(s)
```

Problem 13

- A nested number list is a list whose elements are either:
  - numbers
  - nested number lists

e.g. `[1, 2, [11, 13], [8,[2,3]]]`

Write a recursive function to sum all the numbers in a nested number list

e.g. `r_sum([1, 2, [11, 13], [8,[2,3]]])` returns `40`

Recursive partitioning

Which is the more **efficient algorithm?**

```
def recPow( a,n):
    """raises a to power n"""
    if n==0: return 1
    else:
        return a*recPow(a,n-1)
```

```
def recPowAlt ( a,n):
    """raises a to power n"""
    if n==0: return 1
    else:
        factor=recPowAlt(a,n//2)
        if n%2==0:
            return factor*factor
        else:
            return factor*factor*a
```

Recursion - Justification

- **Recursion** is one of the most important ideas in computer science, but it's usually viewed as one of the harder parts of programming to grasp.
- We can work out very concise and elegant solutions to problems by **thinking recursively**.
- Basic approach traversing for non-linear data structures, such as trees.

Recursion – Justification (2)

- Also, there are problems whose solutions are **inherently recursive**, because they need to keep track of prior state. e.g.:
  - divide-and-conquer algorithms such as Quicksort (coming soon….)

- All of these algorithms can be implemented iteratively with the help of a stack, but the need for the stack arguably nullifies the advantages of the iterative solution.

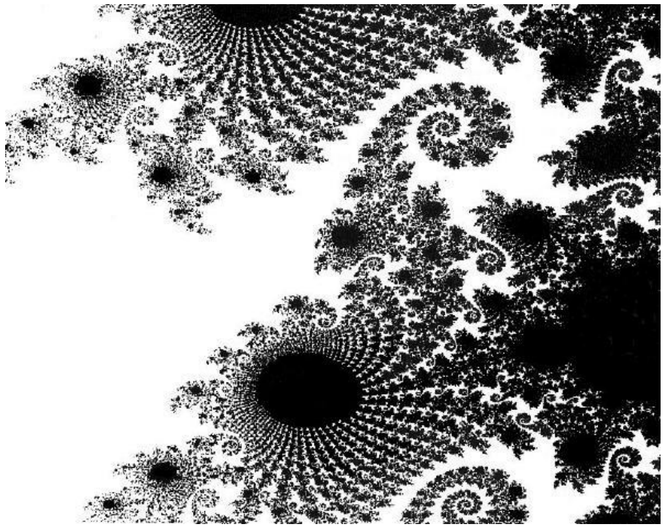
Recursion and Functional Languages

- Recursion is a key concept for functional programming languages, such as Haskell.
- Lisp is the second-oldest high-level programming language in widespread use today; only Fortran is older.
  - Recursion is a key component of language.
  - Closely related to AI and formal logic.
- Resurgence today, especially for parallel programming.

Thinking Recursively

1. There is no infinite recursion
   - Every chain of recursive calls must reach a stopping case
2. Each stopping case returns the correct value for that case
3. For the cases that involve recursion: *if* all recursive calls return the correct value, *then* the final value returned by the method is the correct value

These properties follow a technique also known as *mathematical induction*



UNIVERSITY OF CAPE TOWN

IYUNIVESITHI YASEKAPA • UNIVERSITEIT VAN KAAPSTAD

SCHOOL OF IT