ENG 346
Data Structures and Algorithms for Artificial Intelligence
Recursion

Dr. Mehmet PEKMEZCİ

mpekmezci@gtu.edu.tr

https://github.com/mehmetpekmezci/GTU-ENG-346

ENG-346 Teams code is **0uv7jlm**

Agenda

- Define: Recursion
- Examples: Factorial, Fibonacci Numbers
- Designing Recursive Algorithms
- Example: Towers of Hanoi
- Tail Recursion

Recursion

- When a function calls itself…We have *recursion*.
- Such function/algorithm is called *recursive* function/algorithm.

- Base case(s)
  - Values of the input variables for which we perform no recursive calls are called base cases (there should be at least one base case).
  - Every possible chain of recursive calls must eventually reach a base case.
- Recursive calls
  - Calls to the current method.
  - Each recursive call should be defined so that it makes progress towards a base case.

Example: Factorial

$$f(n) = \begin{cases} 1, & n = 0 \\ n \times f(n-1), & otherwise \end{cases}$$

```
def f(n):
    if n == 0: return 1
    else: return n * f(n-1)
```

Example: Factorial – continued

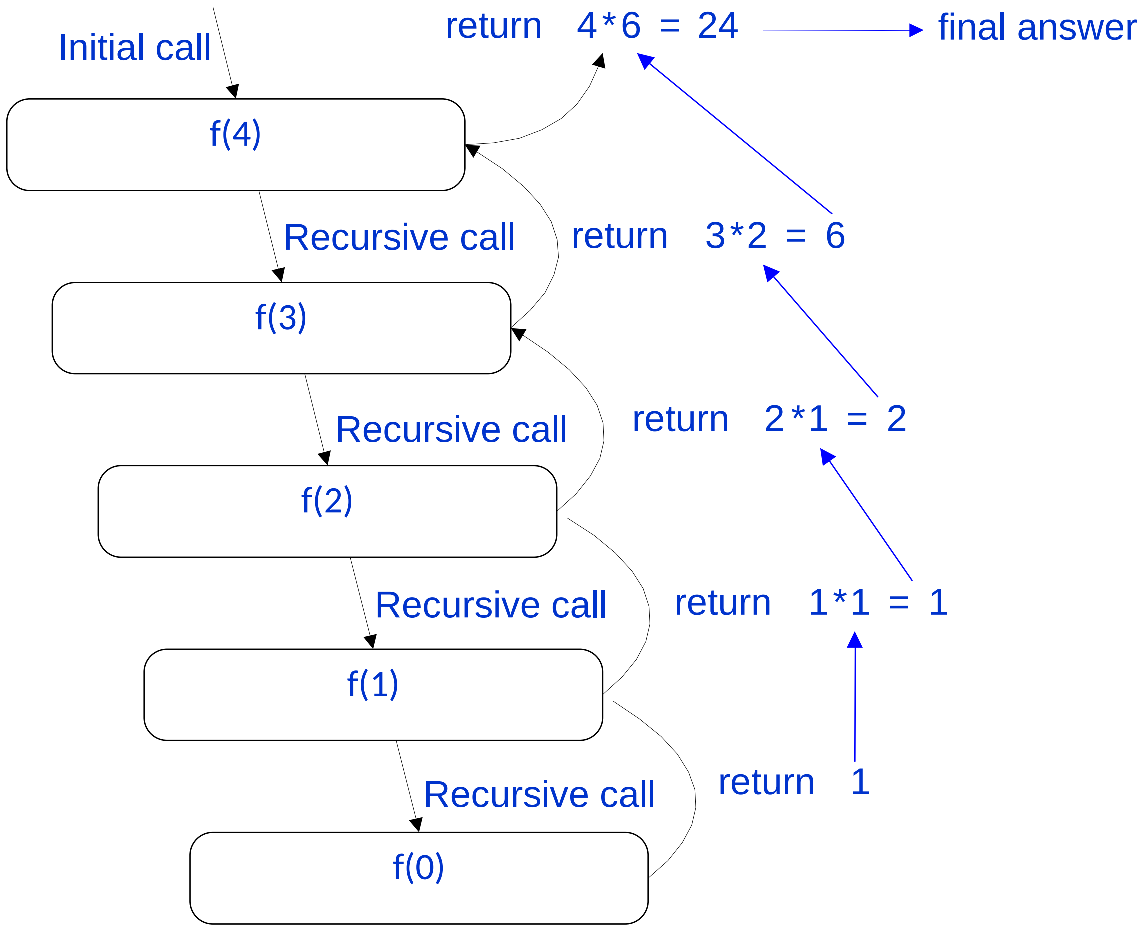
```python
def f(n):
    print ("Called with n=", n)
    if n == 0: return 1
    else: return n * f(n-1)

f(4) # 4!
```

- Output:

  Called with n= 4
  Called with n= 3
  Called with n= 2
  Called with n= 1
  Called with n= 0
  24

Example: Factorial – continued

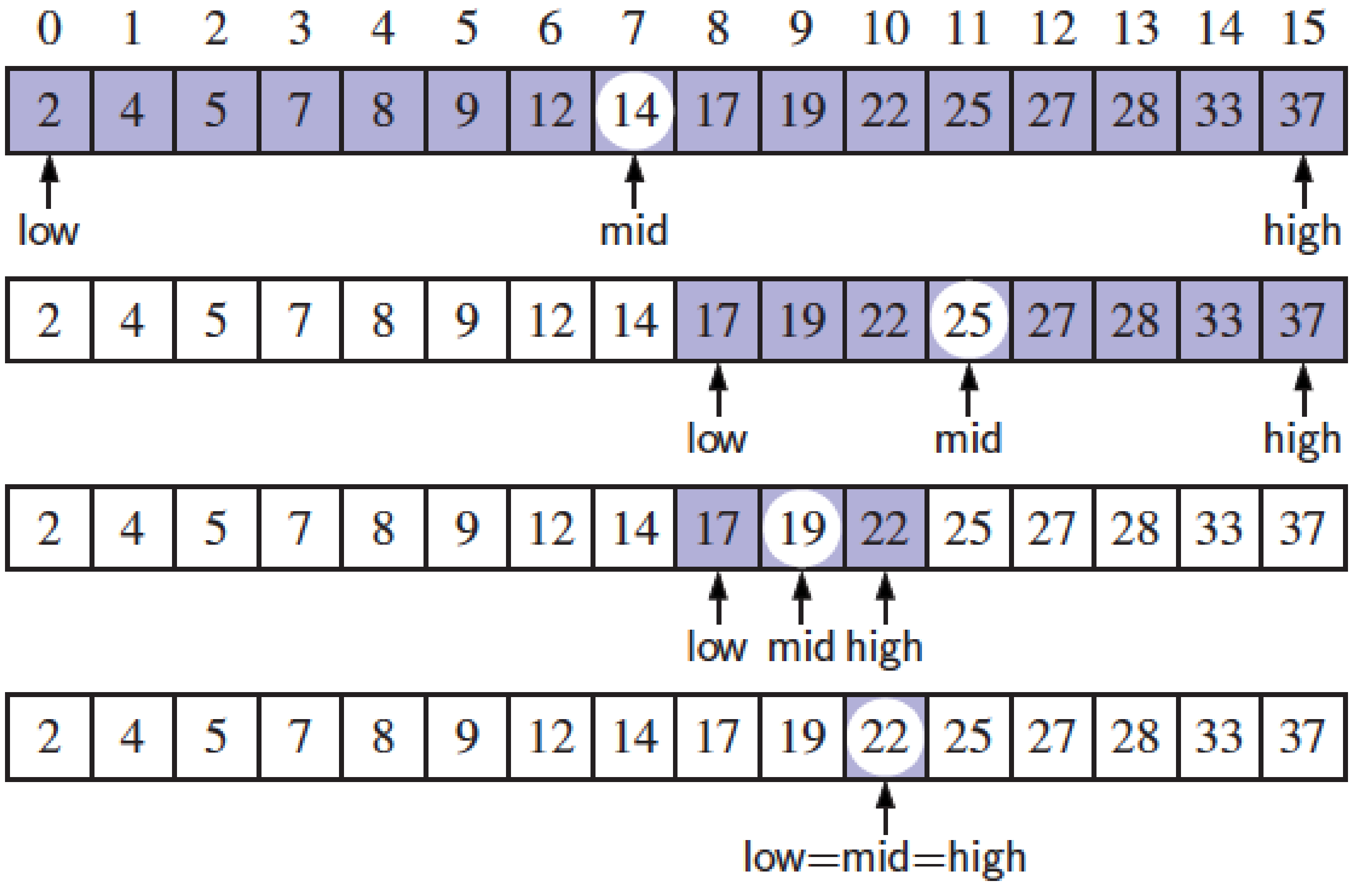


Example: Number Guess

- Pick a number between 1 and 100.

Example: Binary Search

- Search for an integer “target” in an ordered list.



Example: Binary Search – continued

```
def binary_search(data, target, low, high):
  """Return True if target is found in indicated portion of a Python list.

  The search only considers the portion from data[low] to data[high] inclusive.
  """
  if low > high:
    return False                              # interval is empty; no match
  else:
    mid = (low + high) // 2
    if target == data[mid]:                   # found a match
      return True
    elif target < data[mid]:
      # recur on the portion left of the middle
      return binary_search(data, target, low, mid − 1)
    else:
      # recur on the portion right of the middle
      return binary_search(data, target, mid + 1, high)
```

Fibonacci Numbers

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

- fib(0) = 0
- fib(1) = 1
- fib(2) = 1 + 0 = 1
- fib(3) = 1 + 1 = 2
- fib(4) = 2 + 1 = 3
- fib(5) = 3 + 2 = 5
- fib(6) = 5 + 3 = 8
- fib(7) = 8 + 5 = 13
- fib(8) = 13 + 8 = 21

Fibonacci Numbers – continued

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

- Calls for fib(0) = 1
- Calls for fib(1) = 1
- Calls for fib(2) = 1 + 1 + 1 = 3
- Calls for fib(3) = 1 + 3 + 1 = 5
- Calls for fib(4) = 1 + 5 + 3 = 9
- Calls for fib(5) = 1 + 9 + 5 = 15
- Calls for fib(6) = 1 + 15 + 9 = 25
- Calls for fib(7) = 1 + 25 + 15 = 41
- Calls for fib(8) = 1 + 41 + 25 = 67

- Basically:

  Call for fib(n) > 2^(n/2)

- Exponential runtime!

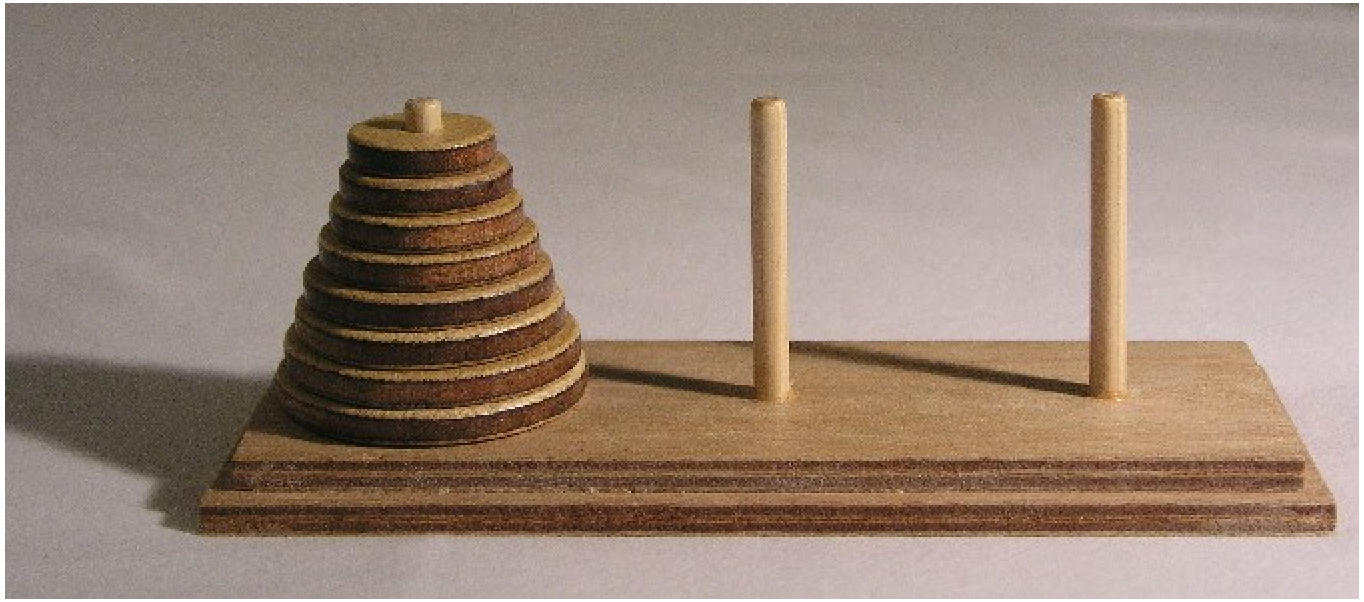
Fibonacci Numbers – continued

```
def fib2(n):
    """Return the nth Fibonacci
number."""
    if n <= 1: return (n,0)
    else:
        (a, b) = fib2(n-1)
        return (a+b, a)
```

Designing Recursive Algorithms

- Test for base cases:
  - There should be at least one.
  - Base case should not contain recursion.
- Recursive step:
  - Perform one or more recursive calls.
  - Input size (or length) should decrease with each recursive call.
  - Chain of recursive calls should reach to base case.

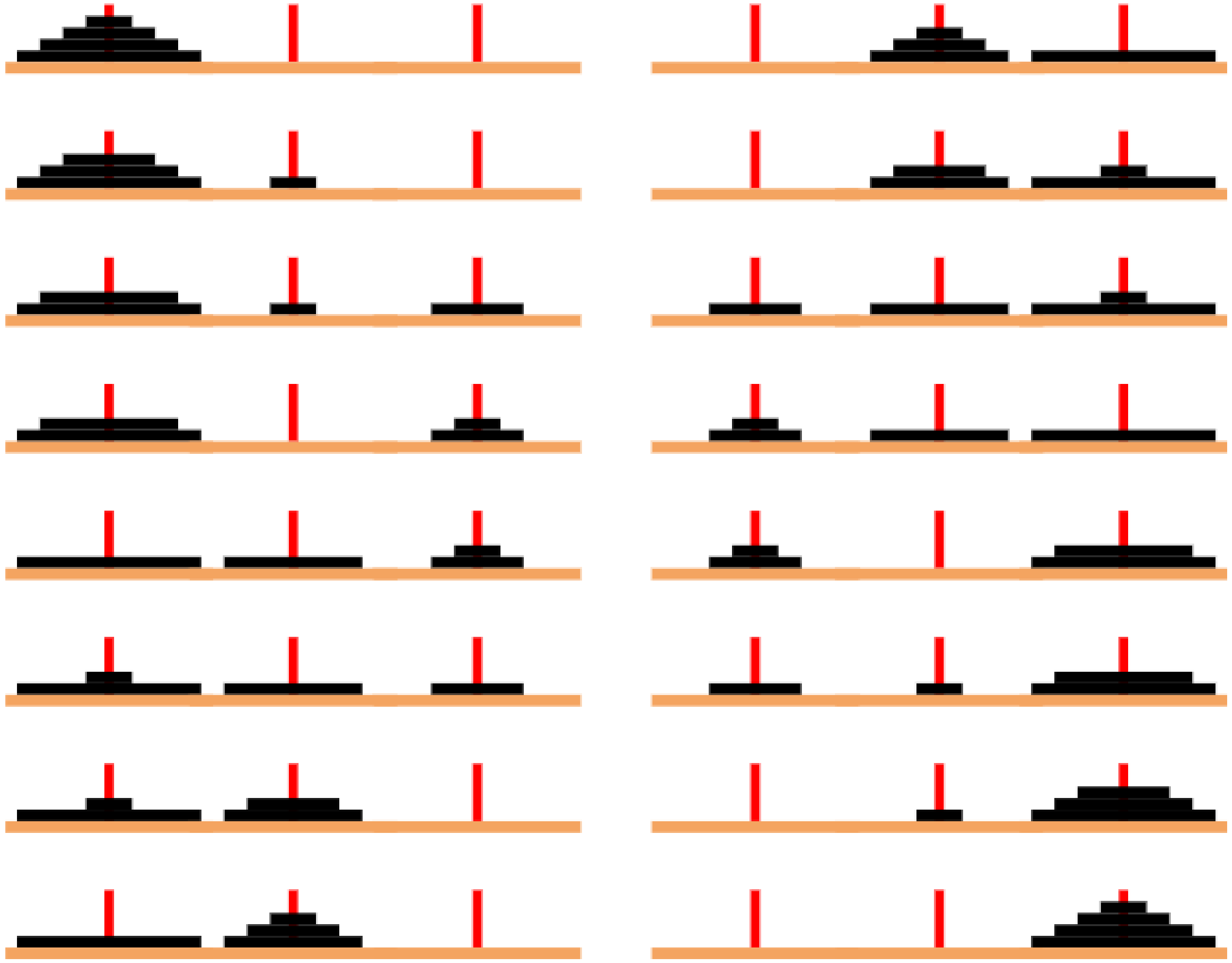
Example: Towers of Hanoi



- Game rules:
  - Only one disk may be moved at a time.
  - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
  - No disk may be placed on top of a disk that is smaller than it.

Example: Towers of Hanoi – continued

Example: Towers of Hanoi – continued



Example: Towers of Hanoi – continued

```
def TowerOfHanoi(n , source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to destination",destination)
        return
    TowerOfHanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to destination",destination)
    TowerOfHanoi(n-1, auxiliary, destination, source)
```

Tail Recursion

- Recursive call is the last step of the function.
- Function returns immediately after recursive call.

- Eliminating tail recursion will clear any overhead resulting from recursive function calls.

Iterative Binary Search

```
def binary_search_iterative(data, target):
    """Return True if target is found in the given Python
list."""
    low = 0
    high = len(data) - 1
    while low <= high:
        mid = (low + high) // 2
        if target == data[mid]: # found a match
            return True
        elif target < data[mid]:
            high = mid - 1 # only consider values left of mid
        else:
            low = mid + 1 # only consider values right of mid

    return False # loop ended without success
```

Exercises

- **R-4.1** Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?
- **R-4.7** Describe a recursive function for converting a string of digits into the integer it represents. For example, “13531” represents the integer 13,531.

ENG 346
Data Structures and Algorithms for Artificial Intelligence
Recursion

Dr. Mehmet PEKMEZCİ

mpekmezci@gtu.edu.tr

https://github.com/mehmetpekmezci/GTU-ENG-346

ENG-346 Teams code is **0uv7jlm**

Agenda

- Define: Recursion
- Examples: Factorial, Fibonacci Numbers
- Designing Recursive Algorithms
- Example: Towers of Hanoi
- Tail Recursion

Recursion

- When a function calls itself…We have *recursion*.
- Such function/algorithm is called *recursive* function/algorithm.

- Base case(s)
  - Values of the input variables for which we perform no recursive calls are called base cases (there should be at least one base case).
  - Every possible chain of recursive calls must eventually reach a base case.
- Recursive calls
  - Calls to the current method.
  - Each recursive call should be defined so that it makes progress towards a base case.

Example: Factorial

$$f(n) = \begin{cases} 1, & n = 0 \\ n \times f(n-1), & otherwise \end{cases}$$

```
def f(n):
    if n == 0: return 1
    else: return n * f(n-1)
```

Example: Factorial – continued

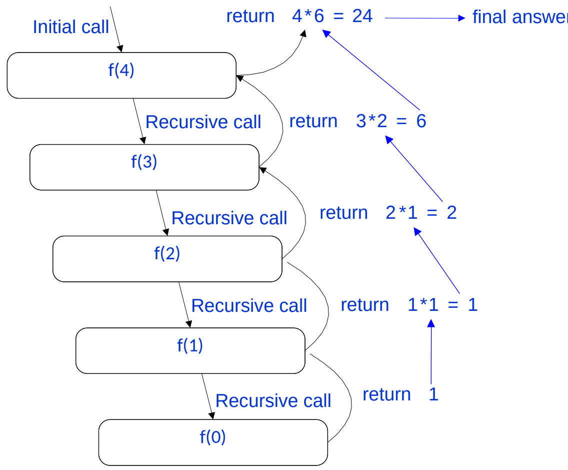
```
def f(n):
    print ("Called with n=", n)
    if n == 0: return 1
    else: return n * f(n-1)

f(4) # 4!
```

- Output:

  Called with n= 4
  Called with n= 3
  Called with n= 2
  Called with n= 1
  Called with n= 0
  24

Example: Factorial – continued

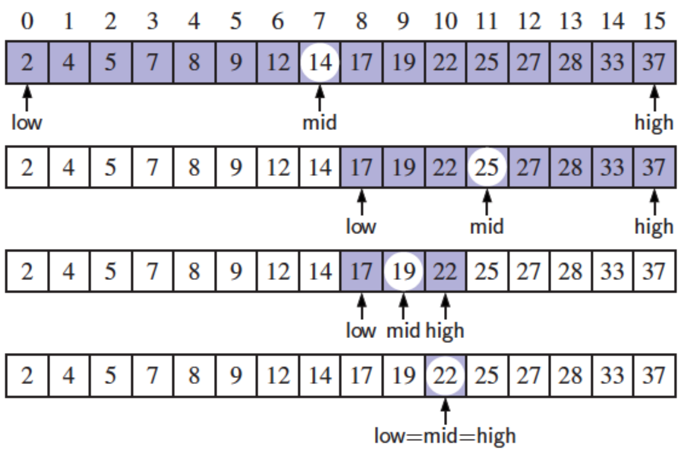


Example: Number Guess

- Pick a number between 1 and 100.

Example: Binary Search

- Search for an integer “target” in an ordered list.



Example: Binary Search – continued

GEBZE TECHNICAL UNIVERSITY

```
def binary_search(data, target, low, high):
  """Return True if target is found in indicated portion of a Python list.

  The search only considers the portion from data[low] to data[high] inclusive.
  """
  if low > high:
    return False                       # interval is empty; no match
  else:
    mid = (low + high) // 2
    if target == data[mid]:            # found a match
      return True
    elif target < data[mid]:
      # recur on the portion left of the middle
      return binary_search(data, target, low, mid − 1)
    else:
      # recur on the portion right of the middle
      return binary_search(data, target, mid + 1, high)
```

Fibonacci Numbers

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

- fib(0) = 0
- fib(1) = 1
- fib(2) = 1 + 0 = 1
- fib(3) = 1 + 1 = 2
- fib(4) = 2 + 1 = 3
- fib(5) = 3 + 2 = 5
- fib(6) = 5 + 3 = 8
- fib(7) = 8 + 5 = 13
- fib(8) = 13 + 8 = 21

Fibonacci Numbers – continued

$$fib(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ fib(n-1) + fib(n-2), & n > 1 \end{cases}$$

- Calls for fib(0) = 1
- Calls for fib(1) = 1
- Calls for fib(2) = 1 + 1 + 1 = 3
- Calls for fib(3) = 1 + 3 + 1 = 5
- Calls for fib(4) = 1 + 5 + 3 = 9
- Calls for fib(5) = 1 + 9 + 5 = 15
- Calls for fib(6) = 1 + 15 + 9 = 25
- Calls for fib(7) = 1 + 25 + 15 = 41
- Calls for fib(8) = 1 + 41 + 25 = 67

- Basically:

  Call for fib(n) > 2^(n/2)

- Exponential runtime!

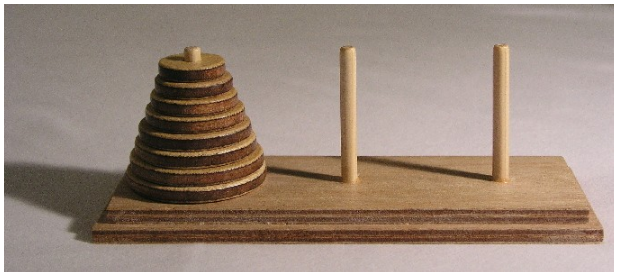
Fibonacci Numbers – continued

```
def fib2(n):
    """Return the nth Fibonacci
number."""
    if n <= 1: return (n,0)
    else:
        (a, b) = fib2(n-1)
        return (a+b, a)
```

Designing Recursive Algorithms

- Test for base cases:
  - There should be at least one.
  - Base case should not contain recursion.
- Recursive step:
  - Perform one or more recursive calls.
  - Input size (or length) should decrease with each recursive call.
  - Chain of recursive calls should reach to base case.

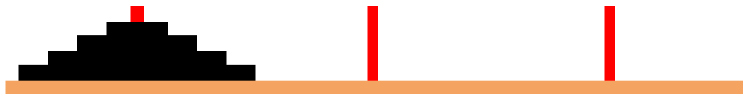
Example: Towers of Hanoi



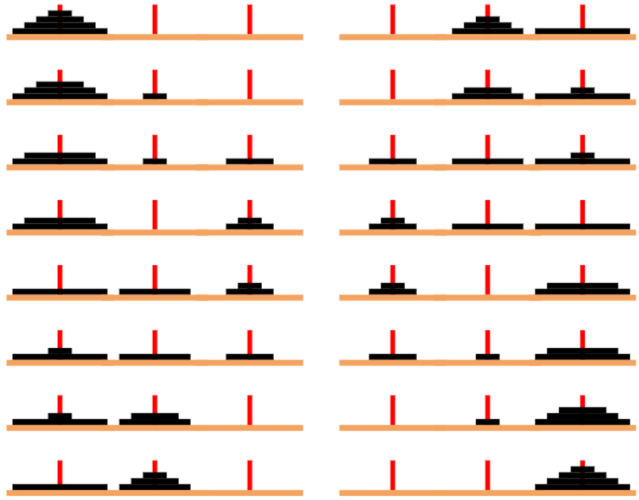
- Game rules:
  - Only one disk may be moved at a time.
  - Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack or on an empty rod.
  - No disk may be placed on top of a disk that is smaller than it.

Puzzle

Example: Towers of Hanoi – continued



Example: Towers of Hanoi – continued



Example: Towers of Hanoi – continued

```
def TowerOfHanoi(n , source, destination, auxiliary):
    if n==1:
        print ("Move disk 1 from source",source,"to 
destination",destination)
        return
    TowerOfHanoi(n-1, source, auxiliary, destination)
    print ("Move disk",n,"from source",source,"to 
destination",destination)
    TowerOfHanoi(n-1, auxiliary, destination, source)
```

Tail Recursion

- Recursive call is the last step of the function.
- Function returns immediately after recursive call.

- Eliminating tail recursion will clear any overhead resulting from recursive function calls.

Iterative Binary Search

```
def binary_search_iterative(data, target):
    """Return True if target is found in the given Python
list."""
    low = 0
    high = len(data) - 1
    while low <= high:
        mid = (low + high) // 2
        if target == data[mid]: # found a match
            return True
        elif target < data[mid]:
            high = mid - 1 # only consider values left of mid
        else:
            low = mid + 1 # only consider values right of mid

    return False # loop ended without success
```

Exercises

- **R-4.1** Describe a recursive algorithm for finding the maximum element in a sequence, S, of n elements. What is your running time and space usage?
- **R-4.7** Describe a recursive function for converting a string of digits into the integer it represents. For example, “13531” represents the integer 13,531.

R-4.1) Hint Don’t forget about the space used by the function stack.
R-4.1) Solution If the sequence has 1 element, that is the maximum. Otherwise, consider the bigger of the first element or the maximum of the
other $n-1$ elements. The running time and space usages is $O(n)$.

R-4.7) Hint Process the string left to right.
R-4.7) Solution Use a single-digit as the base case. For a multiple-digit
string, let $s' = sd$ for digit d. We have that $value(s') = d + 10 * value(s)$.