COSC 2306
Data Programming

Recursion

Survey with Bonus Credit

- **Reminder**: this is an advanced course about problem solving skills (algorithm, design, optimization), **NOT** an intro level coding course
- **Change 1**: Increase # of in-class practices and code together to emulate lab session (pause bonus opportunity for in-class activity, faster going through the algorithm concepts)
- **Change 2**: More frequent assignments (less questions per assignment)
- **Change 3**: Add no-grade assignments

Python implementation

```
def largest(l):
    if len(l) == 1:
        return l[0]
    else:
        max = largest(l[1:])
        if l[0] >= max:
            return l[0]
        else:
            return max
```

Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34 . . .

What are the patterns above?

Fibonacci Sequence

- Sequence: 1, 1, 2, 3, 5, 8, 13, 21, 34 . . .
- Given first two numbers (a_1 and a_2)
  - n^{th} number a_n, $n >= 3$, of sequence given by: $a_n = a_{n-1} + a_{n-2}$
- Recursive function: `rFibNum`
  - Determines desired Fibonacci number
  - Parameters: three numbers representing first two numbers of the Fibonacci sequence and a number *n*, the desired *n*th Fibonacci number
  - Returns the *n*th Fibonacci number in the sequence

Fibonacci Number

- Third Fibonacci number
  - Sum of first two Fibonacci numbers
- Fourth Fibonacci number in a sequence
  - Sum of second and third Fibonacci numbers
- Calculating fourth Fibonacci number
  - Add second Fibonacci number and third Fibonacci number

Fibonacci Number

- Recursive algorithm to calculate *n*th Fibonacci number
  - *a* denotes first Fibonacci number
  - *b* denotes second Fibonacci number
  - *n* denotes *n*th Fibonacci number

$$rFibNum(a, b, n) = \begin{cases} a & \text{if } n = 1 \\ b & \text{if } n = 2 \\ rFibNum(a, b, n-1) + & \\ rFibNum(a, b, n-2) & \text{if } n > 2. \end{cases}$$

Fibonacci Number

Recursive function implementing algorithm

```
def rFibNum(a, b, n):
    if n == 1:
        return a
    elif n == 2:
        return b
    else:
        return rFibNum(a, b, n-1) + rFibNum(a, b, n-2)
```

```
for n in range (1, 11):
   print(n, ":", rFibNum(1, 1, n))
```

```
1 : 1
2 : 1
3 : 2
4 : 3
5 : 5
6 : 8
7 : 13
8 : 21
9 : 34
10 : 55
```

Fibonacci Number

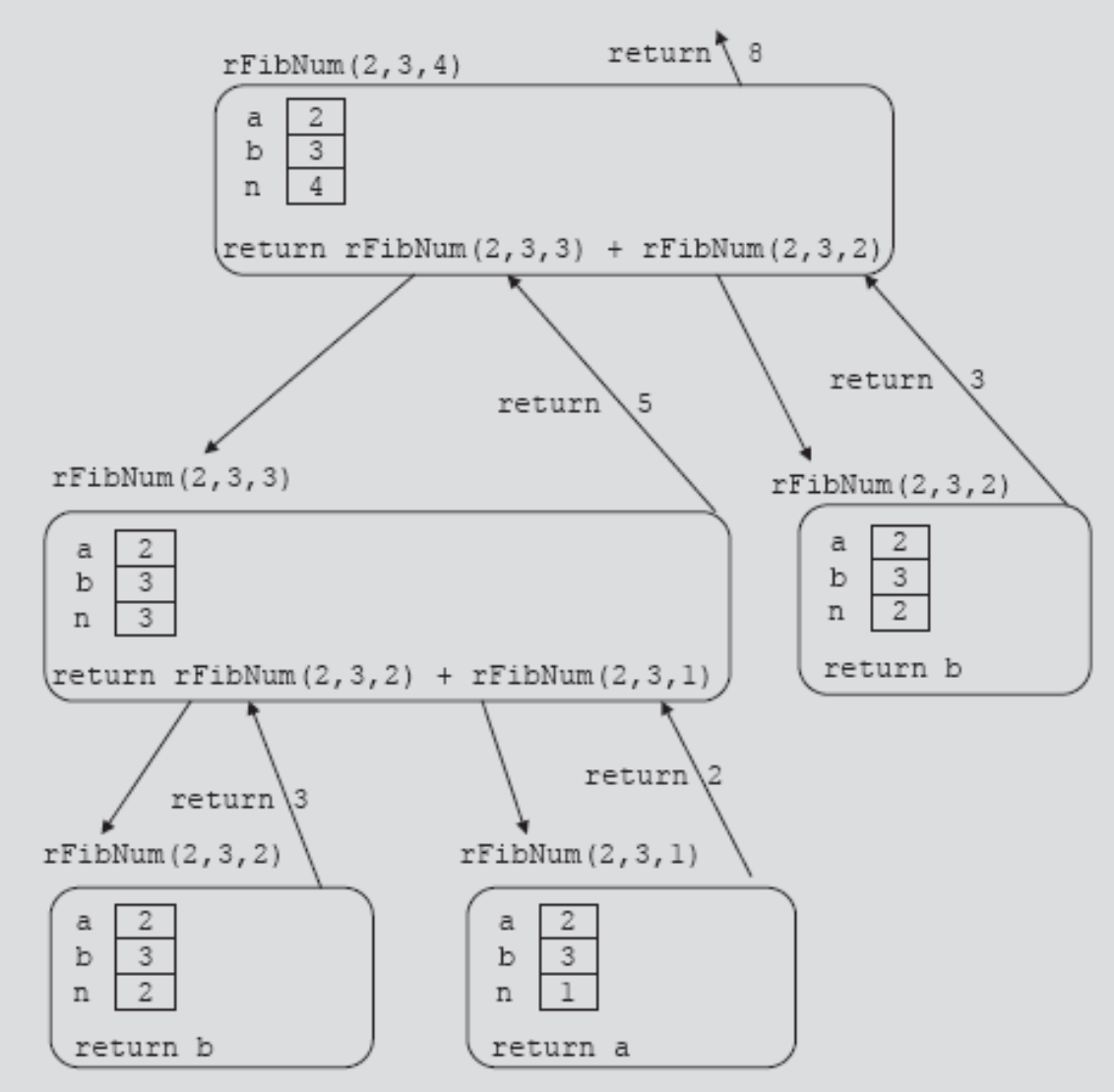

FIGURE 6-7 Execution of `rFibNum(2, 3, 4)`

Fibonacci Number

Recursive function implementing algorithm

```
def rFibNum(a, b, n):
    if n == 1:
        return a
    elif n == 2:
        return b
    else:
        return rFibNum(a, b, n-1) + rFibNum(a, b, n-2)
```

```
for n in range (1, 100):
   print(n, ":", rFibNum(1, 1, n))
```

f(5)=f(4) + f(3)
=f(3) + f(2) + f(2)+f(1)
=f(2)+f(1) + f(2) + f(2)+f(1)

Very slow: open too many function calls, requires lots of memory

Fibonacci Number

Use cache to avoid redundant function calls

```
rFibNum_cache = {}

def rFibNum(a, b, n):
    if n in rFibNum_cache:
        return rFibNum_cache[n]
    if n == 1:
        save = a
    elif n == 2:
        save = b
    else:
        save = rFibNum(a, b, n-1) + rFibNum(a, b, n-2)

    rFibNum_cache[n] = save
    return save
```

```
for n in range (1, 100):
   print(n, ":", rFibNum(1, 1, n))
```

In-class practice

- Write a recursive function sum_digits(n) for a recursive sum of digits

```
Base case: a single-digit number's digit sum
is the number itself

General case: extracts the last digit of n,
contributing that digit to the sum, and then
removes the last digit, producing the "rest of
the number"
```

Sum of digits

```
def sum_digits(n):
    if n < 10:
        return n #base case
    else:
        return n%10 + sum_digits(n//10)
                                #general case
```

In-class practice

- Write a recursive function reverse_string(s) to reverse the input string s.

```
Base case: if the string is empty, return it
unchanged -- an empty string reversed is still
empty

General case: each call moves the first
character to the end after reversing the rest
```

Reverse string

```
def reverse_string(s):
    if len(s) == 0:
        return s #base case
    else:
        return reverse_string(s[1:]) + s[0]
                                    #general case
```