Writing Recursive Functions
Announcements

• Lab 9 due Saturday at midnight
• Quiz 5 on Friday
• Ninja session tonight, 7-10pm
Today’s Plan

• Topics for quiz 5
• Review Monday’s class
• More recursion
  - Stack diagrams for recursive functions
  - Practice with recursion
  - Recursive rules of thumb
Quiz 5

- Linear search: $O(n)$

- Binary search: $O(\log n)$
  - Be able to do a trace, filling in chart with “low”, “mid”, and “high”

- Analysis of algorithms: categorize algorithms into $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, or $O(n^2)$ run time
Quiz 5

• Top-down design

• Finding extreme value pattern
Review

• Merge sort is an $O(n \log n)$ sorting algorithm that can be implemented using recursion.

• A recursive function is one that calls itself in its own definition.

• You call a recursive function once, say in main(), and it repeatedly calls itself until its base case is reached. At this point the stack has several “copies” of the recursive function, each with its own frame.
Iterative vs. Recursive

- Recursion often replaces a loop:

```python
def sumToNum(n):
    if n == 1:
        return 1
    else:
        return n + sumToNum(n-1)
```

```python
def sumToNumIterative(n):
    total = 0
    for num in range(1, n+1):
        total += num
    return total
```
• Alternative approach:

```python
# Does the recursive sum
def sumToNumHelper(n, accum):
    ...

    Purpose: Returns the sum of the numbers from 1 to n, plus the accumulated value
    Parameters: n – int value to sum to
                accum – accumulated value to include in sum
    Returns: 1 + ... + n + accum
    ...

    if n == 1:
        return 1 + accum
    else:
        return sumToNumHelper(n-1, n+accum)

# More convenient wrapper around sumToNumHelper
def sumToNum(n):
    return sumToNumHelper(n, 0)
```
How to write a recursive function

• Start with the **base case**, a “small” version of the problem that can be solved immediately.

• Move to the **general case** (or cases). Take a leap of faith and assume you can solve smaller versions of the problem. Break the general case down into a smaller sub-problem.

• As always, think about whether your recursive function is being called for its return value or its side effects.
cs.swarthmore.edu/~mauskop/cs21/s17/practice/recursion-practice.html
Recursive Rules of Thumb

• When recursing on an integer (#1, #4):
  - Typical base case: 0 or 1
  - Typical general case: Use fn(n-1) to solve fn(n)

• On a list (#2, #3, #9, #10):
  - Typical base case: len(L) == 0 or len(L) == 1
  - Typical general case: Use fn(L[1:]) to solve fn(L)

• On a string (#6, #7, #8):
  - Typical base case: s == “”
  - Typical general case: Use fn(s[1:]) to solve fn(s)
Recursive Rules of Thumb

• If you are asked to write a function, \texttt{foo}, that recurses over a list, \texttt{L}:

  - Solve the problem directly for a list of length 0
  
  - Imagine you have a working version of \texttt{foo}. Ask yourself what \texttt{foo(L[1:]}) will return and whether you can use this return value to calculate the return value for \texttt{foo(L)}. 

Good luck on quiz 5!