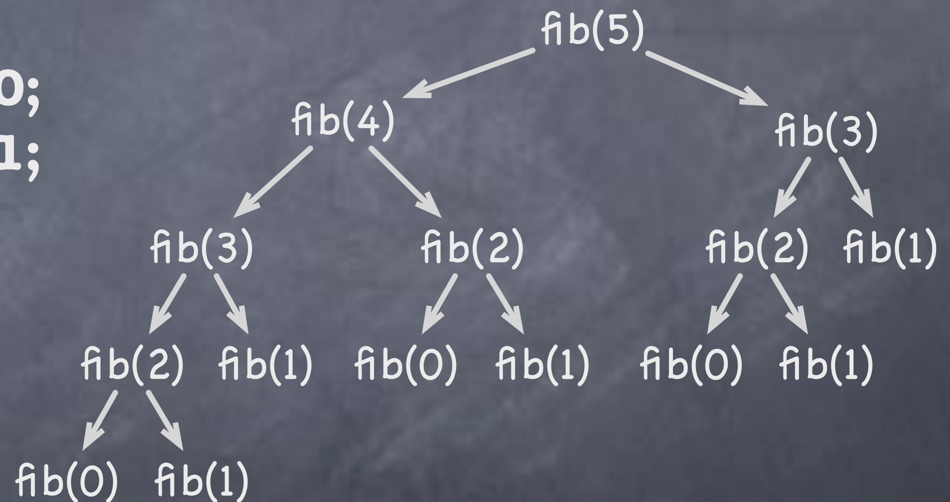
Basic Analysis of Algorithms

Curt Clifton
Rose-Hulman Institute of Technology

Recursive Fibonacci

- ```
  long fib(int n) {
      if (n <= 0) return 0;
      if (n == 1) return 1;
      return fib(n-1) +
                fib(n-2);
  }
  ```

- Why so slow?



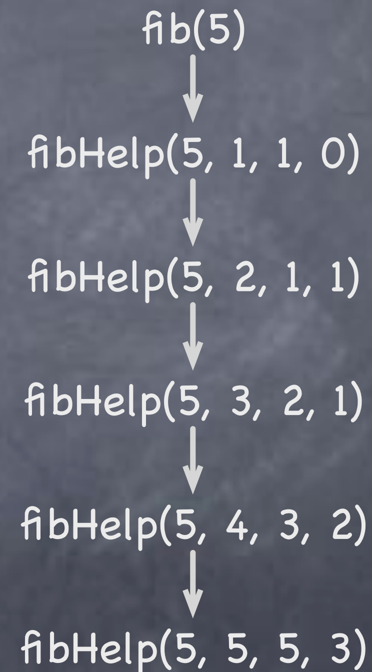
Shlemiel the Painter: http://www.joelonsoftware.com/articles/fog0000000319.html
  ```

Tail-recursive Fibonacci

- ```
  long fib(int n) {
      return fibHelp(n, 1, 1,
  0);
  }
  ```

- ```
  long fibHelp(int n, int m,
  long fm, long fmm1) {
      if (n < m) return 0;
      if (n == m) return fm;
      return fibHelp(n, m+1,
              fm + fmm1, fm);
  }
  ```

- Why so much better?



long maxes out at fib(92) = 7,540,113,804,746,346,429

Can we improve on this?

- ```
  static long fibLoop(int n) {
      if (n <= 0) return 0;
      if (n == 1) return 1;
      int m = 1; long fm = 1; long fmm1 = 0;
      while(m < n) {
          m++;
          long nextFM = fm + fmm1;
          fmm1 = fm;
          fm = nextFM;
      }
      return fm;
  }
  ```
- How much better?
  ```

Iteration vs. Recursion

- Loops often harder to understand than recursive implementations
- Engineering tradeoff:
  - Maintainability vs. efficiency
- “To iterate is human, to recurse divine.”
  — L. Peter Deutsch

Cartoon of the Day

Analysis of Algorithms

- A technique for **predicting** the **approximate** run-time performance of some code
- Helps in deciding whether efficiency improvement is worthwhile

Algorithm

- A well-defined computational procedure that:
  - take some value(s) as **input** and
  - produces some value(s) as **output**
- An algorithm is a tool for solving a **computational problem**

Reminder of slides based on [Cormen, Leiserson, and Rivest, 1990]

The Fibonacci Problem

- Input: a natural number **n**
- Output: fib(n) where fib is defined by

$$fib(n) = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ fib(n-1) + fib(n-2) & \text{otherwise} \end{cases}$$

The Array Search Problem

- Input:
  - A sorted array of integers a[0], ..., a[n-1]
  - and an integer m
- Output:
  - An index i such that a[i] == m
  - or -1 if no such i exists

Array Search Solution

- ```
  int search(int[] a, int m)
  {
      int n = a.length;
      for (int i=0; i < n; i++) {
          if (a[i] == m)
              return i;
      }
      return -1;
  }
  ```
- What things might we want to **predict** when **analyzing** this?
- Let a = {2, 3, 5, 7}
  - Runtime for m = 2
  - Runtime for m = 5
  - Runtime for m = 11
- Suppose a has 100 elements (n = 100)?
  ```

Approximating Runtime - Some Assumptions

- One processor
- Unlimited memory
- One operation at a time
- All individual operations take same amount of time

What is the Runtime of Linear Search

- In terms of the **size of the input**
- Best case?
- Worst case?
- Average case?
- Which case should we care about most?

Big-Oh Notation

Approximation

- Analysis of algorithms is concerned with **predicting** the **approximate** runtime cost
- We typically:
  - Just worry about significant differences between algorithms
  - Just worry about very large inputs

Example

- Suppose each execution of a fib method takes $5e^{-9}$ seconds, not counting recursive invocations
- What's the execution time of fib(5)...
  - for the simple recursive version? $\approx 75e^{-9}$ sec
  - for the tail-recursive version? $\approx 25e^{-9}$ sec
- What about fib(50)?

  ≈NNNNNNNe^{-9} sec vs. $\approx 250e^{-9}$ sec

“On the order”

- Recursive fib takes “on the order” of fib(n) steps
- Tail-recursive fib takes “on the order” of n steps

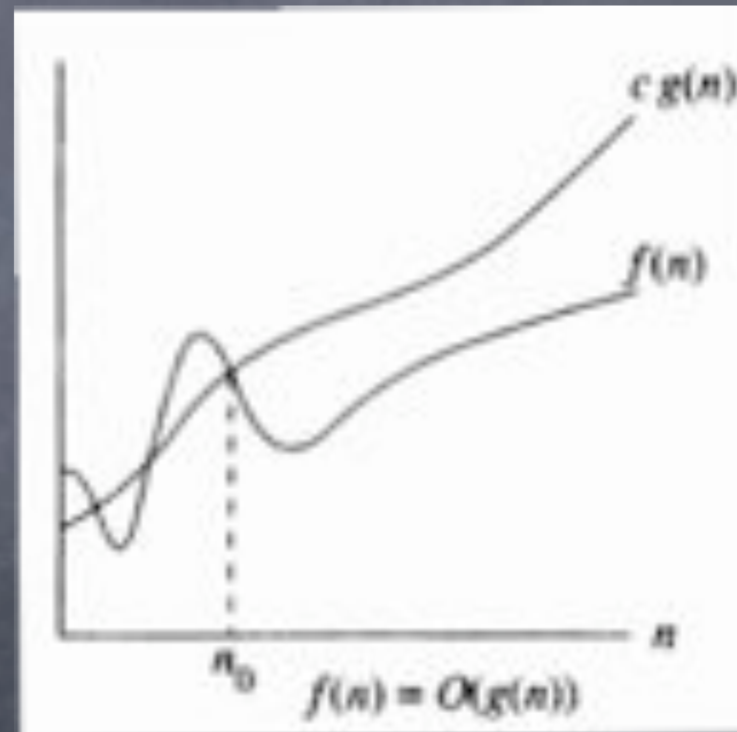
Big-Oh Notation

- A formal notation for “on the order of”
- Focuses on very large inputs
- Is asymptotic – provides a bound on the value for large numbers

Formally

g is a ceiling on f

- We write $f(n) = O(g(n))$,
- and say "f is big-oh of g"
- if there exists positive constants c and n_0 such that
- $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0$



Review for Exam 2