Computing for Math and Stats

Lecture 21

Recursion

- That’s a fancy name for a very simple concept
- Recursion is when a function calls (invokes) itself
- There are algorithms that are inherently recursive
  - They are very awkward to implement without recursion
- There are algorithms that can be implemented both recursively and non-recursively (iteratively)
  - The non-recursive version is usually a bit faster, sometimes much faster
- One can design a language without for-loops or while-loops, as long as the language supports recursion

Recursion Basics

- If a function keeps calling itself it will never end
  - Actually it will end by crashing.
- All recursive functions work in more or less the following way
  - Examine if the problem is “basic” enough to be solved in one shot and if it is solve it.
  - Otherwise solve it by invoking self with a “simpler” version of the problem one or more times.

Recursive Fibonacci

$$f(x)=\begin{cases} x=1 & 1 \\ x=2 & 1 \\ o/w & f(x-1)+f(x-2) \end{cases}$$

Recursive Fibonacci

- The recursive implementation looks like the mathematical definition

```
function [ f ] = recfib( n )
%RECFIB recursive (and slow) Fibonacci
%   Returns the n_th Fibonacci

if (n==1)|(n==2)
    f = 1;
else
    f = recfib(n-1)+recfib(n-2);
end
end
```

Why so slow

- The implementation seems to be very slow for large numbers
- This is because we compute and recompute the same things over and over again
- This is an example where one should not use recursion... at least not this way

Quicksort

- This is much (much) faster than bubblesort on average
- It is also the fastest on average
- It is not perfect though
  - In some (rare) cases it is slow
  - Not very good for small inputs
  - Requires extra memory (not much)

Quicksort

- How it works
  - Pick an element p at “random”
  - Any element less than p goes to the top of the array
  - Any element greater than p goes to the bottom
  - Recursively call quicksort on the top half
  - Recursively call quicksort on the bottom half

How fast is Quicksort

- With some optimizations it is the fastest on average
- Occasionally is slow
- It needs a little extra memory due to the recursion

$$T(N) = T\left(\frac{N}{2}\right) + T\left(\frac{N}{2}\right) + N$$

$$T(N) = 2\,T\left(\frac{N}{2}\right) + N$$

$$T(N) = N \log N$$