Slide 1

Administrivia

- Homework 5 on the Web; due next Thursday.

Slide 2

Recurrence Relations — Review/Recap

- Previously we talked about defining sequences recursively, via base case(s) and recursive case. Can also call this a recurrence relation (recursive part) plus a basis step or initial conditions (base case(s)).
- "Solving" one of these amounts to finding an equivalent non-recursive ("closed-form") definition.
- One way to solve is to guess a solution and then prove it works by induction.
- Another way is to use one of the formulas from this chapter — *if* the relation has the right form. Last time we talked about two of these special cases. (Finish Fibonacci example?) One more . . .

Slide 3

Yet Another Special Case

- One more case for which there's a formula is one of interest in analysis of algorithms, especially those that take a "divide and conquer" approach (e.g., quicksort, mergesort, binary search). In math terms, the recursive part is

$$S(n) = cS(n/2) + g(n), \quad \text{for } n = 2^m, n > 1$$

- For problems that fit this case, the "expand, guess, verify" method produces the following:

$$S(n) = c^{\log n} S(1) + \sum_{i=\log n}^{n} (c^{\log n - i} g(2^i))$$

- Example: practice #25 in textbook.

Slide 4

Analysis of Algorithms, Overview

- Often there's more than one way to solve a given problem, i.e., more than one algorithm. Which one is "best"? Depends on what "best" means. If we mean "fastest":
- A useful measure of approximate execution time is worst-case (or sometimes average-case) execution time expressed as a function of "problem size" (e.g., for operations on array, size of array) — "time complexity" of algorithm. (Another measure is "space complexity".)
- Customary to skip over housekeeping operations and count only "important stuff" — arithmetic operations, comparisons, etc.

  Also customary to "round off" the estimate to an "order of magnitude" — for a problem of size N, we say an algorithm is $O(f(N))$ if execution time is somehow comparable to $f(N)$.

Slide 5

Analysis of Algorithms, Examples

- Example — computing a sum of N numbers. How many additions?
- Example — sequential search of array of size N. How many comparisons (worst case)?
- Example — binary search of sorted array of size N. How many comparisons (worst case)?

Slide 6

Analysis of Algorithms, Longer Example

- Look at several algorithms for computing a^b, for b a positive integer. First version:

```
double exp(double a, int b) {
    double temp = a;
    for (int i = 1; i < b; i+=1)
        temp *= a;
    return temp;
}
```

- How many multiplications needed?

Slide 7

Analysis of Algorithms, Longer Example Continued

- We could also express this recursively:

```
double exp(double a, int b) {
    if (b == 1)
        return a;
    else
        return a * exp(a, b-1);
}
```

Does this work? (Yes. Why?)

- How to figure out how many multiplications? Define and solve a recurrence relation.

Slide 8

Analysis of Algorithms, Longer Example Continued

- We could also express this recursively another way:

```
double exp(double a, int b) {
    if (b == 1)
        return a;
    else {
        double temp = exp(a, b/2);
        if (b % 2 == 0)     return temp * temp;
        else                return temp * temp * a;
    }
}
```

Does this work? (Yes. Why?)

- How to figure out how many multiplications? Define and solve a recurrence relation. (For now do this only for b a power of 2.)

Slide 9

Analysis of Algorithms, Continued

- More complicated (but faster) a^b algorithm — example of "divide and conquer" algorithms. General form:

```
if (base case)
    solve
else {
    split into subproblems
    solve subproblem(s)
    merge subsolutions
}
```

- In general, recurrence relation for work involved has the form

$$S(n) = cS(n/2) + g(n), \quad \text{for } n = 2^m, n > 1$$

for which we have a formula, right?

Slide 10

Analysis of Algorithms, Continued

- Example — recurrence relation for exponentiation algorithm:

$$\begin{aligned} M(1) &= 0 \\ M(n) &= 1 + M(n/2), \text{for } n = 2^m, n > 1 \end{aligned}$$

Slide 11

Analysis of Algorithms and "Big-Oh" Notation

- Often useful to further approximate time for algorithm using "order of magnitude" of function — e.g., $O(n)$, $O(n^2)$.
- We will talk about this more later (chapter on functions), but for now — idea is that all $O(g(n))$ algorithms are bounded above, for large n, by a multiple of $g(n)$, so they all have similar behavior as n increases.

Slide 12

Minute Essay

- How many comparisons are needed to sort an array of N elements using bubble sort?:

```
for (int i = 0; i < N-1; i+=1) {
    for (int j = 0; j < N-1-i; j+=1) {
        if (a[j+1] < a[j])
            swap(a[j+1], a[j]);
    }
}
```

Slide 13

Minute Essay Answer

- `N-1` + `N-2` + `N-3` + . . . + `0`, i.e., `(N-1)*N/2`. (One comparison per trip through the inner loop, and the number of inner-loop trips for each trip through the outer loop depends on the value of `i`.)