Slide 1

Administrivia

- Reminder: Homework 4 due today. Accepted without penalty through 5pm tomorrow.
- Homework 5 to be on the Web soon, due the Tuesday after the break. This will be a “not accepted late” so I can distribute a solution.

Slide 2

Proofs by Induction, Revisited

- Based on the answers on the quiz, many people are still having trouble with proofs by induction (though there were some very nice answers).
- Keep in mind — with proofs in general, goal is not so much to produce an answer (as in other kinds of math problems) as convince a human reader. So some prose may be useful. For inductive proofs, good to explicitly identify base case, hypothesis and desired conclusion for inductive step.

Slide 3

Proofs — A Rant

- If you are to show that $a = b$ — you're not solving an equation, so you don't write down $a = b$ and operate on both sides!
- Instead, build up a chain of equalities with intermediate results c_i:

$$\begin{aligned} a &= c_1 \\ c_1 &= c_2 \\ &\cdots \\ c_n &= b \end{aligned}$$

Okay to omit LHS (left-hand side) if same as previous step's RHS. To prove $a < b$, replace one or more $=$ with $<$.

Slide 4

Analysis of Algorithms — Review/Recap

- Often useful to be able to estimate algorithm's execution time as a function of "problem size".

  Customary to skip over housekeeping operations and count only "important stuff" — arithmetic operations, comparisons, etc.
- Useful in comparing efficiency of different algorithms for same problem. Also useful in determining feasibility of single algorithm. (E.g., something that requires evaluating $N!$ possibilities will not be practical for large N.)

Slide 5

Analysis of Algorithms, Examples (Recap)

- Example — computing a sum of N numbers. How many additions? ($N - 1$)
- Example — sequential search of array of size N. How many comparisons (worst case)? (N)
- Example — binary search of sorted array of size N. How many comparisons (worst case)? (about $\log_2 N$)

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)
        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

Proving Program Correctness — Preview

- Once you've written a program, want to have some confidence that "it works".
- What do you mean "it works"? Informally? Formally, "meets its specification" (more later).
- How do you show it works? As a grad-school colleague wrote:

  To reduce the number of errors in a program, or to increase one's confidence in a program, one can *test* the program on a given test suite. If the program is observed to behave correctly for these test cases, the program is shipped to the customer. One then hopes there will be other cases that customers try for which the program also behaves correctly.
- Is there another way to "increase your confidence" in the program? "Formal methods" . . .

Slide 13

Proving Program Correctness, Continued

- Idea of formal methods is to give a mathematical proof that a program does what it's supposed to do.
- For non-trivial programs, this is usually a lot of work, though if the program is "important" enough, might be worthwhile.
- We will do mostly trivial examples — mostly because they're all we can do in the time we have. Keep in mind, though:
  - How to make this practical, and/or how to have it done by a smart program, are subjects of ongoing research.
  - In my opinion/experience, applying these ideas informally helps you "reason about programs" — which is how careful programmers work, consciously or not.

Slide 14

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

Slide 15

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`.)