Problem Solving

Programming is a strenuous exercise in problem solving

G. Polya. *How to Solve It*, 2nd ed., Princeton University Press, 1957.

- **Understand the problem**
  - **What are its parts? unknown, data, condition**
  - **Does the problem make sense? Is it feasible?**
  - **Think about the problem, get a sense of what it needs**
- **Make a plan**
  - **Find the connection between givens and result**
  - **What kind of problem is it? Is it familiar?**
  - **Think about generalizations, specializations, variants**
- **Carry out the plan**
  - **Check each step**
- **Examine the result**
  - **Does it make sense?**

Back of the envelope calculations

Jon Bentley. *Programming Pearls*. 2nd edition. A-W, 2000.
http://www.vendian.org/envelope/

- Engineering technique to approximate and check answers
  - Two answers are better than one
  - Quick checks
  - Rules of thumb
  - Practice
- Ad claims that salesperson drove 100,000 miles in a year. True?
- Newspaper article states that a United States quarter dollar coin has "an average life of 30 years." How can you check that claim?

Why "back of the envelope" estimates?

- Often need to make rapid estimates
  - to eliminate candidate solutions
  - establish feasibility
  - sketch out potential trade-offs
- Most remember key numbers related to their field, not every detail
- Hence we need to estimate
  - *which* numbers are important
  - *values* of numbers needed
  - *how* to perform the calculation
- Emphasis is on "order of magnitude" estimates
  - to nearest factor of 10 (or 2)

Orders of Magnitude

- How far away is home? Is it more like 1, or 10, or 100 miles?
  - Probably do not know exactly
  - Is it approximately "a couple", or "a few", or "a lot”
  - Estimate based on powers rather than multiples of 10
- How tall is your dorm? More like 1, 10, 100, 1000 feet?
  - 1 foot tall is like a doll house, so that’s out
  - What do we know that is about 10 feet big? Hmm... People
  - If building is a couple of people high, 10 sounds good.
  - But that means 1000, would be 100 people high, so that’s out
  - So 10 or 100 depending on how many people tall the building is
- Use orders of magnitude as brackets to find reasonable range

Example: How many piano tuners in NYC

- Approximately how many people are in New York City?
  - 10,000,000
- Does every individual own a piano?
  - No
- Reasonable to assert "individuals do not own pianos; families do"?
  - Yes
- About how many families are there in a city of 10 million people?
  - Perhaps there are 2,000,000 families
- Does every family own a piano?
  - No
- Perhaps one out of every five does
  - That would mean there are about 400,000 pianos in NYC

Example: Piano Tuners continued

- How many piano tuners are needed for 400,000 pianos?
  - Some people never get around to tuning their piano
  - Some people tune their piano every month
  - Assume "on the average" every piano gets tuned once a year, then there are 400,000 every year
- How many piano tunings can one piano tuner do?
  - Assume that average piano tuner can tune four pianos a day
  - Assume that there are 200 working days per year
  - That means every tuner can tune about 800 pianos per year
- How many piano tuners are needed in NYC?
  - Number of tuners is approximately 400,000/800 or 500

Example: Piano Tuners summary

- "Back of the Envelope" estimates have
  - Formulas: provide roadmap to upcoming calculations
  - Estimates: brief justification of approximations in formula
  - Calculations: estimates and known facts are use in formula
- Piano Tuner example
  - Formula:
    # tuners = # pianos x # repairs / # repairs per day x # days
  - Estimates
    # pianos ~= 400,000 (20% of 2,000,000 families own pianos)
    # repairs ~= 1 per piano (some many, some none)
    # repairs per day ~= 4
    #working days ~= 200 (5 x 50 – vacation, sickness)
  - Calculation
    # tuners ~= (400,000 x 1) / (4 x 200) = 500

Estimation General Principles

- Recall Einstein's famous advice
  - Everything should be made as simple as possible, but no simpler
- Do not worry about constant factors of 2, π, etc.
  - Round to “easy” number or nearest order of magnitude
- Guess numbers you do not know
  - Within bounds of common sense (accuracy increases with experience)
- Adjust geometry, etc., to suit you
  - Assume a cow is spherical if it helps
- Extrapolate from what you do know
  - Use ratios to assume unknown value is similar to known quantity
- Apply a ‘plausibility’ filter
  - If answer seems unbelievable, it probably is
  - Can usually set range of reasonable values that indicates major mistake (e.g., speed cannot be faster than light!)

What's wrong with this algorithm?

(From back of shampoo bottle)

<u>Directions:</u>

Wet Hair
Apply a small amount of shampoo
Lather
Rinse
Repeat

Properties of good algorithms

- Good algorithms must be
  - Correct
  - Complete
  - Precise
  - Unambiguous
- And should be
  - Efficient
  - Simple
  - Contain levels of abstraction

An algorithm is an ordered set of unambiguous, executable steps, defining a terminating process.

Algorithms

- **Hand-waving not allowed!**
- **Specifying algorithms requires you to say what is really involved in making it work.**
- **Example:**
  - **How does a computer work?**
  - **Hand-wave: zeros & ones**
  - **Real answer: see later part of class.**
- **You learn to know when you don't know**
  - ***"I know nothing except the fact of my ignorance."***
  - **Socrates, from Diogenes Laertius, Lives of Eminent Philosophers**

Describing Algorithms

More easily expressed

Pictures

Natural language (English)

Pseudo-code

Specific high-level programming language

More precise

Pseudocode

- A shorthand for specifying algorithms
- Leaves out the implementation details
- Leaves in the essence of the algorithm

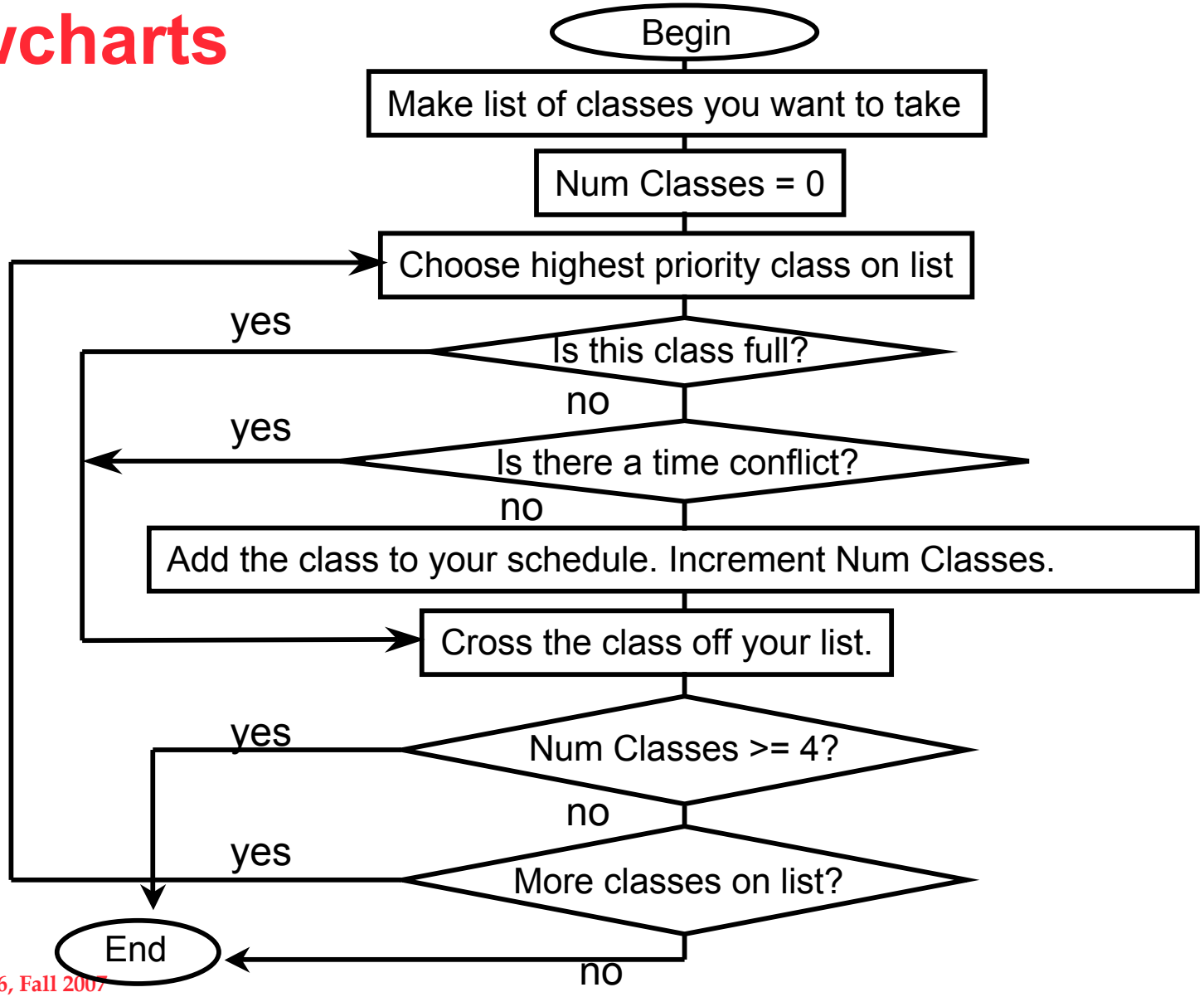
```
Assign r the reminder of x divided by y
  If r equals 0 STOP, final value is y
  Else
    Assign x the value of y
    Assign y the value of r
    Repeat from beginning
```

- What does this algorithm do?

Picking courses

1. Make a list of courses you want to register for, in order of priority
2. Start with empty schedule. Number of courses = 0.
3. Choose highest priority class on list.
4. If the chosen class is not full and its class time does not conflict with classes already scheduled, then register for the class (2 steps):
    1. Add the class to the schedule
    2. Increment the number of classes scheduled
5. Cross that class off of your list.
6. Repeat steps 3 through 5 until the number of classes scheduled is >= 4, or until all classes have been crossed out.
7. Stop.

Flowcharts


Begin
Make list of classes you want to take
Num Classes = 0
Choose highest priority class on list
yes
Is this class full?
no
yes
Is there a time conflict?
no
Add the class to your schedule. Increment Num Classes.
Cross the class off your list.
yes
Num Classes >= 4?
no
yes
More classes on list?
End
no

Sequential search

```
procedure Search (List, TargetValue)
if (List empty)
    then
      (Declare search a failure)
    else
      (Select the first entry in List to be TestEntry;
       while (TargetValue > TestEntry and
                 there remain entries to be considered)
            do (Select the next entry in List as TestEntry.);
       if (TargetValue = TestEntry)
            then (Declare search a success.)
            else (Declare search a failure.)
      ) end if
```