ALGORITHMIC THINKING

Overview

- Finish HTML concepts from last class
- Introduce algorithmic thinking
- Algorithm examples

- Reading: Reed, Chapter 8.

What is an algorithm

- "A step-by-step sequence of instructions for carrying out some task"
- Examples of algorithms outside of computing:
  - Cooking recipes
  - Dance steps
  - Proofs (mathematical or logical)
  - Solutions to mathematical problems
- In computing, algorithms are synonymous with *problem solving*.

- *How to Solve It*, by George Polya
  1. understand the problem
  2. devise a plan
  3. carry out your plan
  4. examine the solution
- Example: find the oldest person in the class (besides me)

Analysis of algorithms

- Often, there is more than one way to solve a problem, i.e., there exists more than one algorithm for addressing any task
- Some algorithms are better than others
- Which *features* of the algorithm are important?
  - Speed (number of steps)
  - Memory (size of work space; how much scrap paper do you need?)
  - Complexity (can others understand it?)
  - Parallelism (can you do more than one step at once?)

- In computer science analysing the speed of an algorithm is important.
- **Big-Oh** notation
  - $O(N)$ means solution time is proportional to the size of the problem (N)
    - For example, N might be the size of the class in our age-finding example.
  - $O(log_2 N)$ means solution time is proportional to $log_2 N$
  - $O(N^2)$ means solution time is proportional to $log_2 N$
- Which is better?
- See examples in Reed page 142

Classic algorithm example: search

- *Sequential* search
- *Binary* search
- Example: search the Manhattan phone book for "Al Pacino".
  - How many *comparisons* do you have to make in order to find the entry you are looking for?
  - Can you take advantage of the fact that the phone book is in *sorted* order? (i.e., an "ordered list")
  - What would happen to your algorithm if the phone book were in random order?
- Test for *equality* versus *relativity*—which will tell you more? which will help you solve the problem more efficiently?

Algorithms and programming

- Programming languages provide a level of abstraction that is more understandable to humans than binary machine language ($0's$ and $1's$).
- *Assembly languages* (in the early 1950's). provided abbreviations for machine language instructions (like *MOV*, *ADD*, *STO*).
- *High-level languages* (introduced in the late 1950's) provided more "programmer-friendly" ways for humans to write computer code (e.g., FORTRAN, LISP).
- The languages we will use in this course are all high-level.

Program translation

- Translates assembly or high-level languages into binary machine language
- Translates *source code* into *object code*, also called *machine code*.
- Two methods:
  - Interpretation
  - Compilation

Interpretation

- Reads and translates statements one at a time;
- Doesn't optimize across an entire program;
  - No file of object code.
- Doesn't store executable statements—just runs them;
- Error checking only happens at "run time".
- Run-time can be slow, but there's no "compile time"

Compilation

- Reads and translates entire program, and stores result as an executable file;
- Can optimize;
- Can perform "compile time" error checking.
- Run-time is fast, but there is "compile time".

Concepts

- *Compile-time (noun)*: (same as "compilation")

  The process of compiling a program from an assembly or high-level language into binary machine language and storing it on the computer's hard disk.

  vs.
- *Compile time (adj noun)*:

  The amount of time it takes a *compiler* to translate (or "compile") a program.

- *Run-time (noun)*: (same as "execution")
  The process of executing a compiled, stored program.
  vs
- *Run time (adj noun)*:
  The amount of time it takes a program to run. This is where *Big-Oh* comes in.

- *Errors*:
  Can be found at compile-time and at run-time.
  vs
- *Error checking*:
  Is done at compile-time.

Summary

- This lecture introduced the idea of *algorithms*
  – Sequences of instructions for sloving problems.
- We looked st some different algroithms for solving simple problems.
- We considered how one compares one algorithm against another.
  – Particularly with respect to speed.
- We also looked at some of the concepts involved in translating a program from source code to object code.