

Algorithmic Thinking



What steps does a computer go through to solve a problem? To be effective computer users, we need to learn the steps involved. Thinking algorithmically is the first one.

© Copyright 2000-2001, University of Washington



Unambiguous Instructions

- ❖ An *algorithm* is a systematic method for deterministically producing a specified result. In other words, step-by-step instructions that, if followed, get you the same result every time.
- ❖ There are two main parts to an algorithm – giving the instructions and following the instructions
 - ❑ Giving the instructions (specifying the algorithm) – in this class, from now on, that will be the programmer
 - ❑ Following the instructions (executing the algorithm) without the help or intervention of the programmer – once again, from this point on we will consider the computer to be doing this
- ❖ We must always assume that the agent giving the instructions will not be the agent who follows them

© Copyright 2000-2001, University of Washington



You're Already Familiar With Algorithms

- ❖ Recipes
 - ❑ Recipes are examples of algorithms written by chefs (programmers) and followed by cooks (computers) to produce a specified food (deterministic result)
- S'mores: Place a toasted marshmallow on a Graham cracker and then place a square of chocolate on top
- ❖ Driving Directions
 - ❑ Written by agents (people or applications –i.e. MapQuest) and followed by drivers to get to a specific destination

© Copyright 2000-2001, University of Washington



The 5 Properties of Algorithms

- ❖ All algorithms must have certain properties if a computer is to execute them successfully without intervention by the programmer
 - ❑ Input Specification
 - ❑ Output Specification
 - ❑ Definiteness
 - ❑ Effectiveness
 - ❑ Finiteness

© Copyright 2000-2001, University of Washington

FIT 100 Input Specification

- ❖ The *input* is the data that will be transformed by the algorithm to create the output
- ❖ The input must exist in a format the computer can access and manipulate
- ❖ When giving an algorithm, one needs to state:
 - ❑ The types of data expected: whole numbers, letter strings
 - ❑ The number of data items expected (the amount so that the computer will know when it has reached the end of the data)
 - ❑ The structure, if any, of the data expected – a list, a table, etc.

© Copyright 2000-2001, University of Washington

FIT 100 Output Specification

- ❖ The *output* is the result of the computation – the description of the result often forms the name of the algorithm
- ❖ The output must be specified in a format that the computer can express (such as on screen, or with audio)
- ❖ The features specified are the same as for the input:
 - ❑ The types of data forming the result
 - ❑ The number of data items forming the result
 - ❑ The structure of the result

© Copyright 2000-2001, University of Washington

FIT 100 Definiteness

- ❖ An algorithm must be explicit about how to work the computation
- ❖ Definiteness comes by giving commands that state unambiguously what to do, in sequence
- ❖ The commands may be ...
 - ❑ Conditional, which requires that a decision to be made. This requires explicit directions on how to respond to all different outcomes
 - ❑ Repeated (Loops), which requires explicitness about when to stop the repetition

The definiteness property assures that the agent executing the instructions will ALWAYS know what command to perform next

© Copyright 2000-2001, University of Washington

FIT 100 Effectiveness

- ❖ Effectiveness assures that the agent following the instructions (the computer) is able to do so without intervention
 - ❑ No additional inputs, special talent, creativity, clairvoyance or help from Superman or other beings
- ❖ Effectiveness is achieved by reducing the task to the primitive operations known to the computer
- ❖ Definiteness assures that the computer ALWAYS know what command to perform next; Effectiveness assures the computer CAN accomplish the command

© Copyright 2000-2001, University of Washington

FIT 100 Finiteness

- ❖ An algorithm must eventually end / terminate with either
 - The "right" output
 - An indication that no solution is possible
- ❖ An algorithm that never terminates is useless since it is impossible to know the difference between continued progress and "stuck"
- ❖ Finiteness is relevant to computer algorithms since they typically repeat instructions

$$\begin{array}{r} 3.3 \\ 3 \overline{)10.000000000 \dots} \\ \underline{9} \\ 10 \\ \underline{9} \\ 1 \end{array}$$

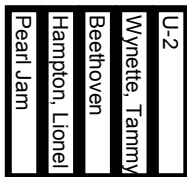
© Copyright 2000-2001, University of Washington

FIT 100 How Definite, Effective and Finite Can You Be? Try Writing an Algorithm

- ❖ Write an algorithm to sort 5 numbers from largest to smallest
- ❖ Take out a scratch piece of paper. Tear it into 5 small pieces. Write the following numbers, one on each piece of paper: 5, 19, 38, 7, 9
- ❖ Shuffle them around
- ❖ Take out another piece of paper. Write your name on it.
- ❖ Write down the steps (an algorithm) to sort these 5 numbers. Note: You can only view and compare 2 numbers at any single point in time
- ❖ Turn these in at the end of class (not graded)

© Copyright 2000-2001, University of Washington

FIT 100 Alphabetized CDs



- ❖ Input: Unordered CDs filling a slotted rack
- ❖ Output: CDs in slotted rack, alphabetized

© Copyright 2000-2001, University of Washington

FIT 100 Algorithm for Alphabetizing

1. "Artist_Of" means the name of the group
2. Pick one end of the rack to be the beginning of the alphabetic sequence. Call that end's slot the "Alpha" slot
3. Call the slot adjacent to the Alpha slot the "Beta" slot
4. If the Artist_Of the CD in the Alpha slot is later in the alphabet than the Artist_Of the CD in the Beta slot, then interchange the CDs
5. If there is a slot following the Beta slot, begin calling it the "Beta" slot and go to step 4; otherwise, continue on
6. If there are two or more slots following the Alpha slot, then begin calling the slot following the Alpha slot, "Alpha" and the slot following it the "Beta" slot and go to step 4; otherwise, stop

© Copyright 2000-2001, University of Washington

Different Ideas for Sorting Algorithms

- ❖ Insertion Sort
 - ❑ Make the first number a list by itself – it is already sorted
 - ❑ "Insert" each number, one at a time, into the correct place in the list; shift the other numbers if you need to
- ❖ Bubble Sort
 - ❑ Compare each pair of numbers, one pair at a time; if the pairs are out of order, swap them.
 - ❑ Keep doing this step until you go through the complete list without having to swap a single pair
- ❖ Exchange Sort
 - ❑ Go through the list, at each step swapping the smallest number into the first slot in the list
 - ❑ Repeat this step with each successive position in the list

Is It An Algorithm, A Program, Or Both?

- ❖ A program is simply an algorithm specialized to a particular situation
 - ❑ Alphabetize CDs, if it were a program, would be called an instance of the Exchange Sort algorithm
- ❖ Exchange Sort can be specialized to other cases
 - ❑ Sort CD's by other criteria – title, genre, etc.
 - ❑ Sort books by title, author or ISBN number
 - ❑ Sort homework papers turned in by student ID, or Name
- ❖ The algorithm, being a process with only a limited number of specifics, is more abstract than a program
- ❖ Therefore, all programs are algorithms, but not all algorithms are programs