CPSC 335 – Lecture 1

ADITA Intro & Algorithm Fundamentals

(ADITA Ch 1-2)

Book Structure (Ch 1)

- Part 1: Common Algorithm Patterns
- Part 2: Algorithm Limitations
- Part 3: Limitation Workarounds

Background Knowledge (Ch 1)

- Basic programming knowledge: variables, data types, loops, if statements
- Fundamental data structures: array, string, linked list, binary search tree, and priority queue
- Discrete math: set theory, combinatorics, probability, functions (the math kind), graphs, formal logic & proofs

Terminology (Ch 2.2)

- **Data: *finite mathematical objects*** that can be represented by a string of binary 0 and 1 digits
- Primitive data types: int, float, char, bool, pointer
- Data structure data types: list, array, queue
- Can be described using mathematical notation, e.g. S is a subset of list X, $S \subseteq X$

Terminology (Ch 2.2)

- **Problem:** an ***input*** and ***output*** specification, each specifying a ***type of data*** and possibly some ***constraints*** on that data
- **(Problem) Instance:** a ***concrete input object*** for a specific problem
- If the **problem** is:
  - ***Input*:** a list of X integers
  - ***Output*:** the sum of all elements of X
- A corresponding **instance** of the problem is:

  <1, 3, 5, -7>

Terminology (Ch 2.2)

- **Solution:** a valid ***concrete output*** corresponding to a specific ***instance*** of the problem
- If the **problem** is:
  - ***Input*:** a list of X integers
  - ***Output*:** the sum of all elements of X
- A corresponding **instance** of the *problem* is:

  <1, 3, 5, -7>
- A corresponding **solution** to this *instance* is:

  2

Terminology (Ch 2.2)

- Define a problem and give an instance of it that has more than one solution:

  Problem:

  Input:

  Output:

  Instance:

  Solution:

Terminology (Ch 2.2)

- **Algorithm:** a ***process*** which produces a ***solution*** for an ***instance*** of a specific problem
- *Process:* a defined *series of actions* directed to some end
- Algorithms must:
  - Describe the process clearly enough for implementation (**clarity**)
  - Always produce a correct solution (**correctness**)
  - Take a finite amount of time (**termination**)

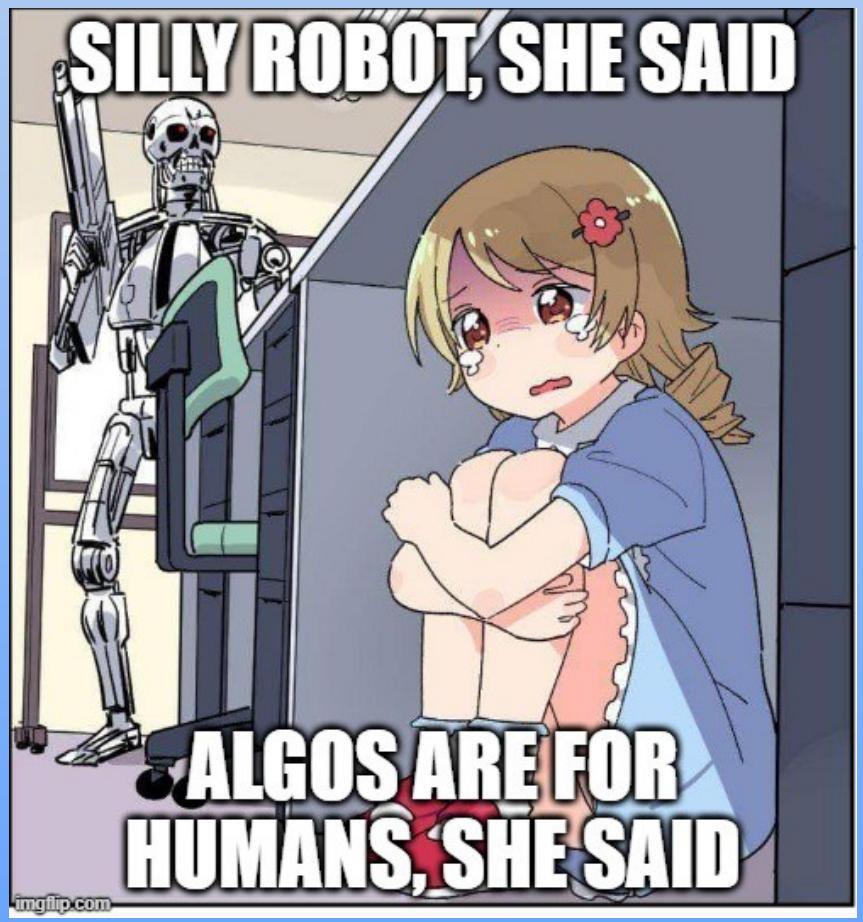
Pseudocode (Ch 2.3)

- **Pseudocode: *human-readable*** format for ***communicating algorithms*** that may include *code-like syntax*, *math notation*, *and prose*
- **Implementation: *executable computer code*** that follows the process defined by the (usually written in pseudocode) algorithm
- Pseudocode on homework, implementation on projects

Pseudocode (Ch 2.3)

- Mathematical objects (for humans): problems, algorithms, pseudocode
- Digital artifacts (for machines):

  instances, solutions, implementations (code)



Pseudocode (Ch 2.3)

- Minimum requirements (*clarity*, **correctness**, termination):
  - *Algo input and output clear (input as params/arguments, output as return value)*
  - *Variables clearly defined and initialized before use*
  - **Every possible execution of the algorithm must return a value that matches the algo's output type**
  - **Must return the right output for every possible input, even if output is None**
  - The algorithm must terminate: no infinite loops, no recursion without base cases

Problem: common subset

input: a list X of integers and a list Y of integers

output: the subset S of integers common to X and Y

[feel free to try out your own pseudocode here, reemeber: clarity, correctness, termination]

Patterns (Ch 2.4)

- Think of algorithm design in terms of **patterns**
- **Patterns** are like algorithm **templates**: fill in the blanks to define a *specific problem*
- Algorithm design steps:
  1. Clearly define the problem (input, output).
  2. Pick a pattern template and copy it.
  3. Fill in the blanks of the template to match the problem (producing pseudocode).
  4. Revise the filled template until pseudocode meets criteria for clarity, correctness, and termination.