CS 2200: Algorithms
Introduction

Bowdoin College
Fall, 2015

Administrative Information

- http://courses.bowdoin.edu/computer-science-2200
- Textbook: *Introduction to Algorithms*, 3rd Edition; Cormen, Leiserson, Rivest, and Stein; McGraw Hill, 1990
- My Office Hours:
  - Mon, 1:00-2:00 pm, Searles 222
  - Wed, 6:00-8:00 pm, Searles 128
- TAs and QR Mentors (Office Hours TBA):
  - Clara Belitz
  - Clara Hunnewell
  - Dan Navarro
  - Mingo Sanchez

What you can expect from me

- Strategies for designing algorithms
  - Divide and Conquer
  - Greedy
  - Dynamic Programming
- When to use those strategies
- Mathematical analysis of algorithm efficiency
- Techniques for arguing algorithm correctness
- Specific algorithms

What I will expect from you

- Every spare minute of your waking life
- And your sleeping life
- And any other life not covered by those two…

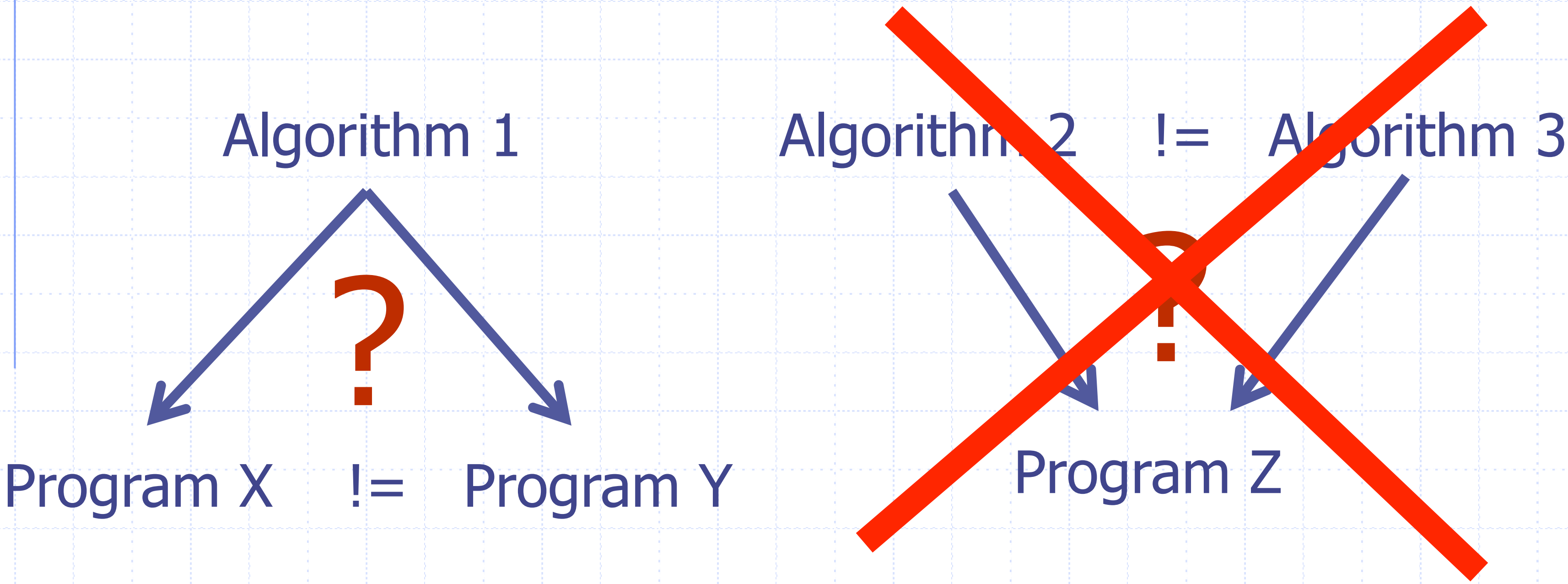
What I will expect from you

- Problem Sets (40%):
  - Collaboration encouraged (but <= 3 total)
  - List collaborator names
  - Write up solutions individually
- Two Non-Cumulative Exams (30% each):
  - No collaboration
  - Exam 1 possibly take-home
  - Exam 2 possibly in-class
  - Open book, open notes

Algorithms and Programs

- An algorithm is a computational recipe designed to solve a particular problem
- Must be implemented as a program in a particular programming language
- Data structures are critical
  - Aren't you glad you've had Data Structures?
  - There is a correct answer to that question.

Algorithms and Programs



Making a telephone call to Jen

pick up the phone

dial Jen’s number

wait for person to answer

talk

Correctness

Waiting at a traffic light

if (light is red)

    wait a while

    accelerate

Definiteness

Looking for an integer >= 0 with property P.

```
i = 0;
test i for property P
while (i with property P not found)
        test i for P
        increment i
```

Finite number of steps

Packing for vacation

flip coin

if (heads)

        pack paraglider

else

        pack knitting

Predictability

Desirable Characteristics

- THEORY suggests (requires):
  - Correctness
  - Definiteness
  - Finiteness
  - Predictability
- Practice suggests:
  - Efficiency (time, but there are other possibilities)
  - Clarity
  - Brevity

An algorithm is:

…a list of **precisely** defined steps that can be done by a computer in a **finite** (and, hopefully, relatively **short**) amount of **time** to **correctly** solve a particular type of **problem**.

Types of Problems

- STRUCTURING: transform input to satisfy Y (SORT)
- CONSTRUCTION: build X to satisfy property Y (MST)
- OPTIMIZATION: find best X satisfying property Y (TSP)
- DECISION: does the input satisfy property Y (SAT)
- APPROXIMATION: find X that almost satisfies property P and has bounded error (TSP)
- RANDOMIZED: make random choices (QuickSort)
- PARALLEL ALGORITHMS (Factoring)
- ON-LINE ALGORITHMS (Job Scheduling)

Sorting

- Pervasive problem:
  - Data processing
  - Efficient search
  - Operations research (e.g. shortest jobs first)
  - Event-driven simulation (e.g. what happens first?)
  - Sub-routine for other algorithms (e.g. Kruskal's MST)
- Informally:
  - Bunch of items
  - Each has a "key" that allows "<=" comparison
  - Put items in ascending (or descending) order according to key comparisons

Sorting

- More formally:
  - Input: A list of n keys A[0], A[1], …, A[n-1] and a total order relation "<=" on the keys
    - Total order is a relation that has reflexivity, antisymmetry, transitivity, and comparability
  - Output: A permutation B of A such that:
    - B[0] <= B[1] <= …. <= B[n-1]
    - A 1-to-1 mapping B such that every object in A is in B and every object in B is in A
- We will assume that:
  - keys are integers,
  - "<=" is the normal integer comparison operator

Sorting

- Bubble Sort
- Selection Sort
- Insertion Sort

Algorithm Design Principle

Sometimes we can devise a new
(possibly better) algorithm
by reallocating our computational efforts.

Reallocate Computational Effort: Example 1

- Sorting a list
- Selection Sort
  - Picking next element to place is harder (always)
  - Placing it is easier
- Insertion Sort
  - Picking next element to place is easier
  - Placing it is harder (only sometimes)

Reallocate Computational Effort: Example 2

- Searching in a list
- Unsorted list
  - Easy to add items
  - Harder to find an item
- Sorted list
  - Extra effort to add items
  - Easier to find an item

Better Sorting Through Recursion

- Selection Sort → Quick Sort
- Insertion Sort → Merge Sort