403: Algorithms and Data Structures

Prof. Petko Bogdanov

Introduction

Fall 2016

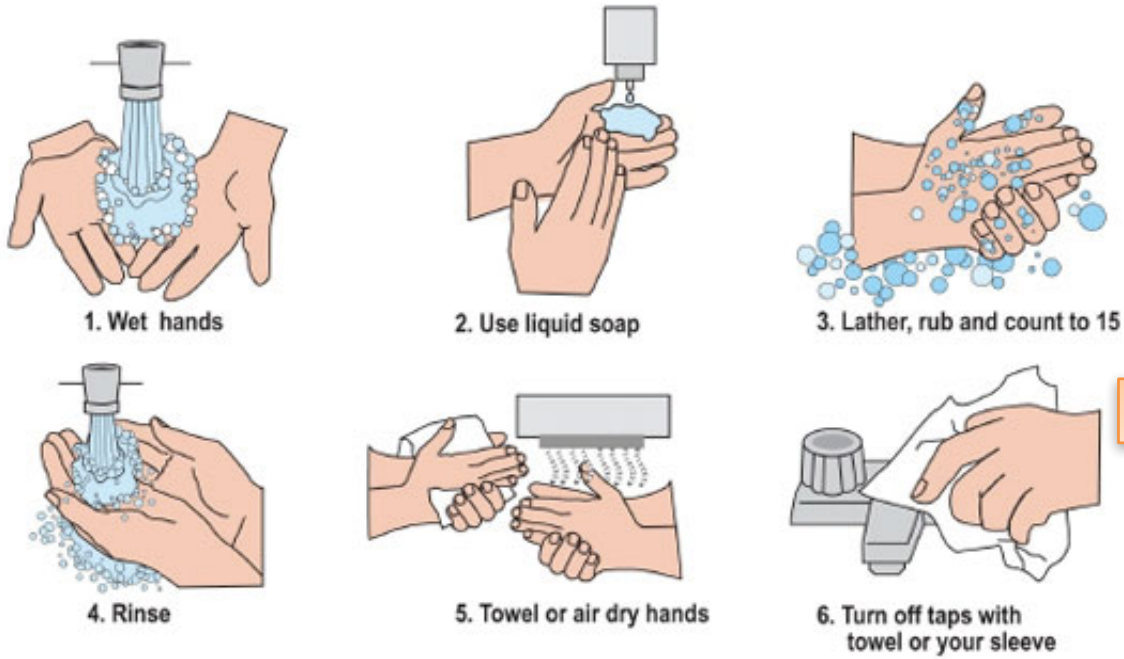
UAlbany

Computer Science

What is this course about?

- Design and analysis of algorithms.
- What is an algorithm?
- Algorithm: a well-defined (computational) procedure to solve a problem

Input: Dirty Hands



Output: Clean Hands

Before we delve into algorithms…

- **Instructor**: Petko Bogdanov,
- For this week only: Prof. Mariya Zheleva
  - mzheleva@albany.edu
- Office Hours: Tue, Thu: 1pm to 3pm, **or by appointment**
- Office: UAB 416 -> Administrative building not on the podium (1215 Western Ave)
- Email: pbogdanov@albany.edu


Washington Ave
East Center Drive
Uncommon Grounds
Coffee & Bagels
1-71 University Place
UAB
Google

TAs

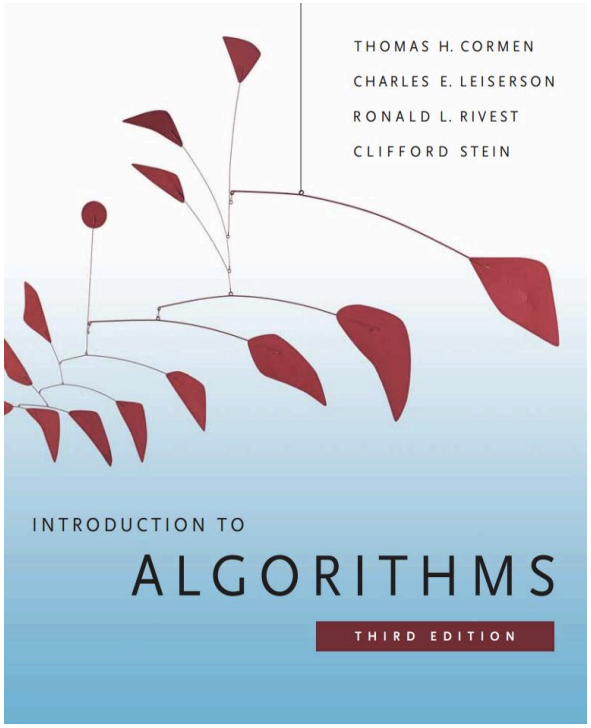
- Ashish Jadhav
  - ajadhav2@albany.edu
  - Office hours: Tue,Thu 10:30am-11:30am.
- Zumrut Akcam
  - zakcam@albany.edu
  - Office hours: Tue 3-4pm and Wed 11am-12pm
- TA office hours rooms TBD (look for an announcement from Blackboard)

Prerequisites

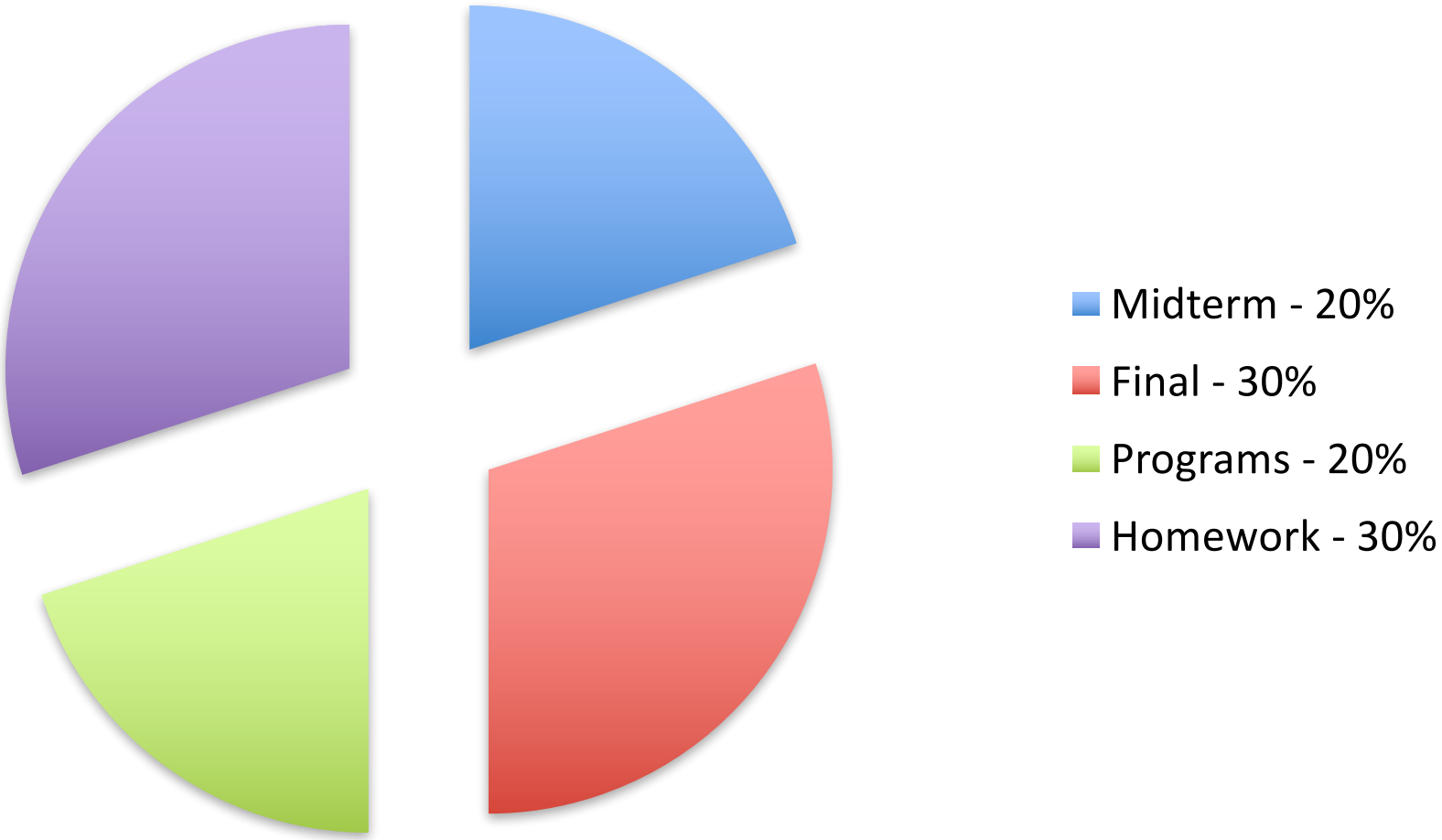
- Prerequisites:
  - CSI 210 (Discrete Structures) and
  - CSI 213 (Data Structures)
- Is there someone who has not taken those classes?
  - Do HW 1 and …
  - If you get more than 80 points you can stay in the class
  - Else, please, take 210 and 213 first

Book

- Required Text:
  - **Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliord Stein, *"Introduction to Algorithms: Ed. 3"*, MIT Press**
- We will mostly follow the book
- Some assignment from the book



Grading



Policies: exams

- Two **exams** – open book
  - Midterm – in class on 10/17/2016
  - Final – early December 2016 (date TBD)

- **Make-up** exams
  - Only for major emergencies (valid and verifiable)
  - Contact the instructor as soon as you know you will be missing an exam to re-schedule

Policies: programming

- Two **programming assignments**
  - Implementation and comparison of algorithms
  - In Java on the department Linux cluster
    - Be sure to check they compile on the cluster before submitting! 0 points for programs that don't compile.
  - Test input/output provided
    - Will not be the same for grading

Policies: homework

- Six **homework assignments**
  - About a week to complete each
  - Equally-graded
  - Best-5-of-6 policy
    - No make-up homeworks
- Assignment submission through Blackboard
  - Programming as a single archive file
  - Homework as a scanned PDF of your solution
    - Only if you are not able to scan before class: you can bring a hard copy

Policies: cheating

- Cheating on exams
  - You will receive an E-grade and will be reported
- Cheating on assignments
  - **All assignments are to be completed individually**
  - First attempt: all students involved will get 0 points for the assignment
  - Following attempts will result in E-grade; students will be reported for disciplinary action

Policies: *I* grades (incomplete)

- Only given for circumstances beyond your control **IF**:
  - Your work is in *good standing* as of the midterm point (10/17/2016)
    - Homework score >= 50%
    - Programming completed at least at 50%
    - Midterm grade equivalent of C
  - You are able to provide written documentation to prove you are unable to complete the course

Policies: attendance

- Not mandatory but will determine your class performance
  - It is your responsibility to find out the material and notes covered in classes you missed
- This is a hybrid slide/board course
  - Always come prepared to take notes

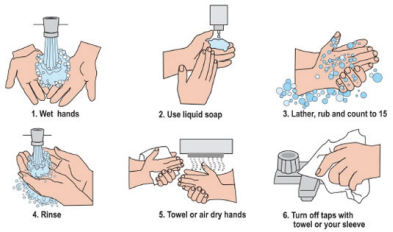
Reading ahead

- Read material ahead in the book
  - make more out of the lectures
  - let ideas “sink” longer
  - do not worry if some details unclear
- For this week: Read Chapters 1 and 2

Homework 1

- Due 09/07/2016 before class
- 4 problems to brush up your Discrete Math
- Need to pass with at least 80% to stay in class
- Start early, so you can consult with the TAs and Prof. Bogdanov.

Back to our definition



- Algorithm: a well defined (computational) procedure to solve a problem
- To specify the problem we need:
  - Input: What is given?
  - Output: What is needed?

Example problem specifications

- Finding the maximum
  - Input: A sequence of numbers $<x_1,x_2 \ldots x_n>$
  - Output: The largest number among $x_1,x_2 \ldots x_n$
- Sorting in non-decreasing order
  - Input: A sequence of numbers $<x_1,x_2 \ldots x_n>$
  - Output: A permutation (reordering) of $<x'_1,x'_2 \ldots x'_n>$ such that $x'_1 \leq x'_2 \leq \ldots \leq x'_n$

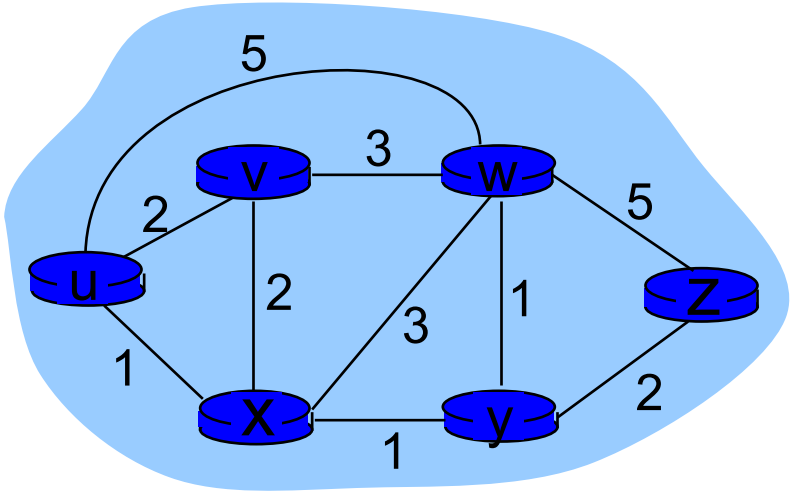
Instance and correctness

- A sequence such as <-7,4,9,2,17,8> is an **instance** of the problem (i.e. values for all required inputs, satisfying stated constraints)
- A sorting algorithm transforms the sequence <-7,4,9,2,17,8> into <-7,2,4,8,9,17>
- An algorithm is **correct** if for every instance it *halts* with the *correct output*.

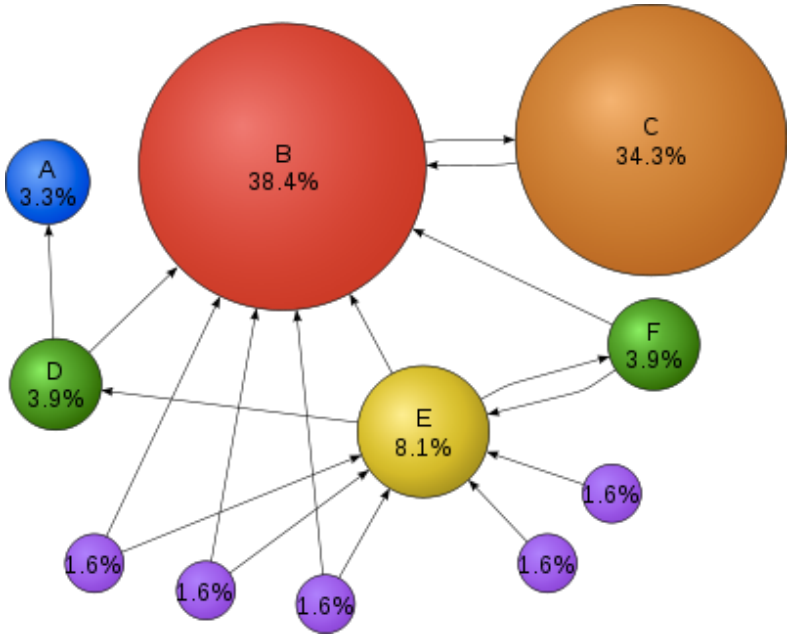
What kind of problems are solved by algorithms?

The Internet

- Internet routing algorithms
  - Decide how to handle packets through Internet paths
  - You will study Dijkstra's algorithm for finding the shortest path



- Internet applications. E.g. Google Search uses PageRank
  - A rough estimate of page importance



Other examples include

- Electronic commerce
  - Privacy of user information
  - Persistency across multiple server instances
- Manufacturing/commercial/political enterprises
  - Political candidates – how to spend campaign money to maximize the chance of winning
  - Airline companies – assign crews to flights to minimize expenses
  - Internet service provider – where to build infrastructure to minimize delays and maximize Internet bandwidth

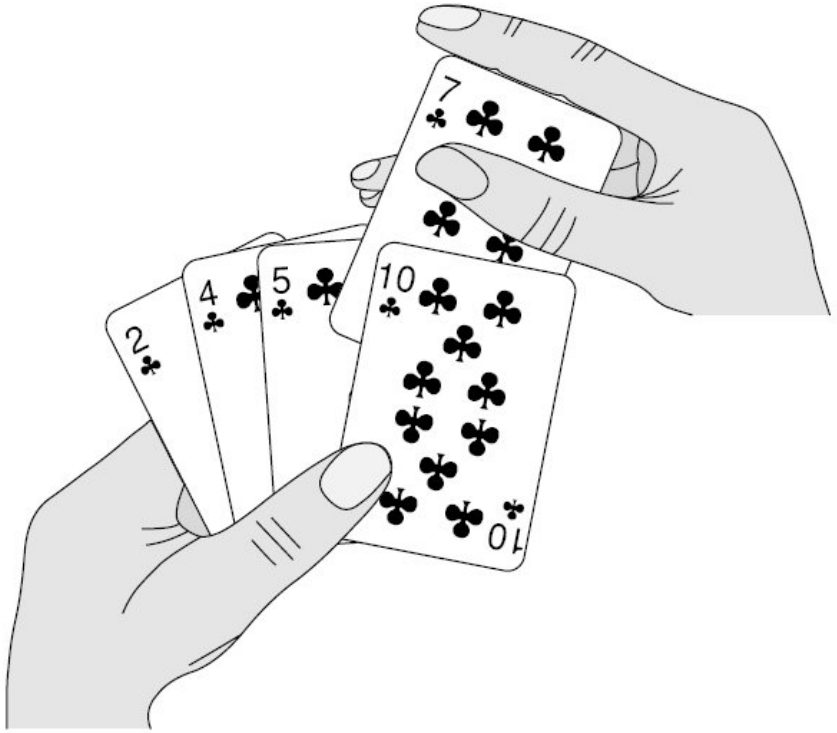
Two common characteristics

- Many candidate solutions
  - Finding one that is ”the best” can present quite a challenge
- Practical applications
  - Presents opportunities to narrow the scope → easier to find “the best” solution

More basics

- Pseudo code and Algorithm Analysis by example

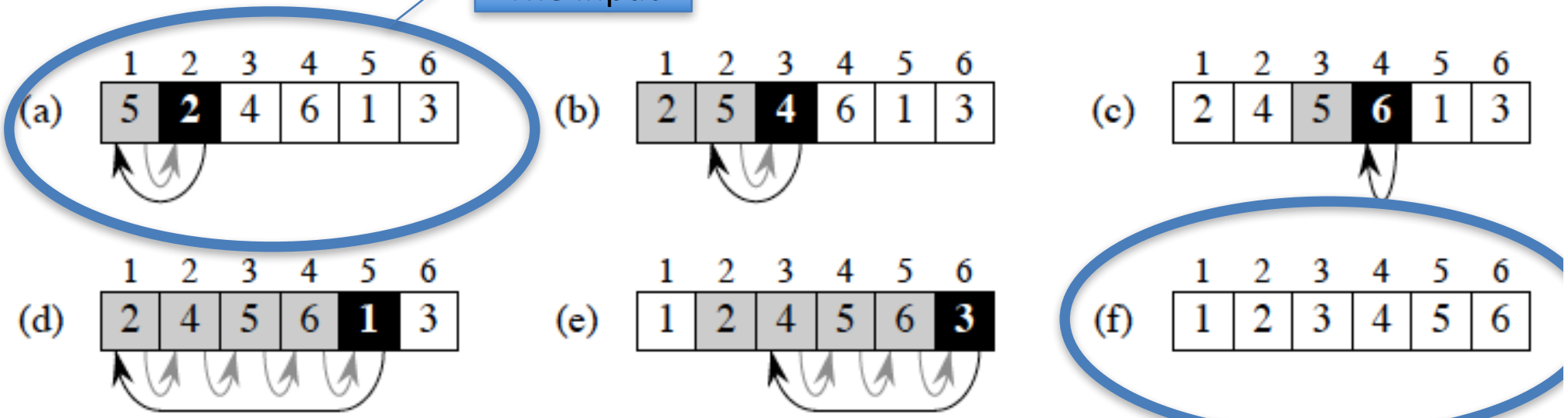
- Insertion sort



Example of ***insertion sort*** on an instance

The algorithm

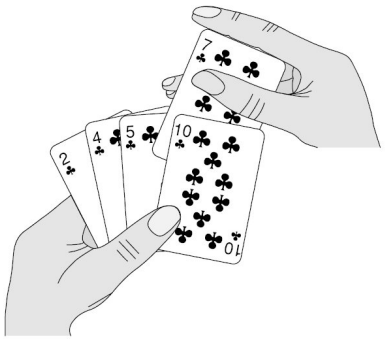
The input



The output

- Which is the algorithm?
- Which is the input?
- Which is the output?
- What is the instance?

<5,2,4,6,1,3>



Insertion sort (pseudo code)

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

This step can be reached when i=0 or if A[i]≤key. In both cases key is placed s.t. A[1…i] is sorted

What is algorithm analysis?

- Estimating (predicting) the resources needed by an algorithm
  - ***Time:*** how long would it take
  - ***Memory:*** how much memory it would require for the used data structures
- Our goal is to find the best algorithm among alternatives in terms of minimum resource requirements.
- How to quantify important algorithm characteristics and omit tedious details?

Analysis

- Resources are expressed as a function of the *size of the input*
- The size of the input depends on the problem:
  - *Sorting:* number of elements being sorted n
  - *Integer multiplication:* size of the integers (i.e. number of bits to be multiplied)
- Execution (running) time of an algorithm: the number of primitive operations executed
  - e.g. assignment, comparisons, arithmetic ops. …
  - typically assumed that operations take const time
  - allows machine independence for the analysis

Worst case

- When using *running time* we will mean **worst-case** time – the largest running time for any input of a ***fixed size*** n
  - Note it is also possible to define average (or expected) time assuming we have picked an instance at random from all possible instances, but we will most likely not discuss this here
- In a sense we will be pessimists about our algorithms, always expecting the worst possible input.

Advantages of being pessimist

- Worst-case time analysis
  - Provides an upper bound on any input, i.e. the algorithm will never take more than this.
  - For some algorithms, worst-case occurs often
  - Helps us provision with redundancy (often a good idea in building software systems)

Sanity check

- Suppose computers were infinitely-fast and computer memory was free.
  - Would you have any reason to study algorithms?
  - Yes! You still need to demonstrate that your solution **terminates (halts)** and gains the **correct** answer.

- But... memory is not free and CPUs take time to complete a primitive operation.
  - *Efficient* use of resources to gain *correct* answers.

Announcements

- Next time: Insertion Sort
  - Read through Chapters 1 and 2 from the book
- Homework 1 posted, Due on Sep. 7
- Go over the syllabus on your own and ask questions as necessary