CPSC 327
Data Structures and Algorithms

Data Structures and Algorithms

- a **data structure** deals with the organization of data for efficient access and modification
- an **algorithm** is a procedure to accomplish a specific task

Ideally an algorithm is
correct,
efficient, and
easy to implement.

(But you can't always get all three.)

Challenge

An array contains each of the numbers 1 – 1,000,000 plus one duplicate value. Which value is duplicated?

- fairly easy to come up with algorithms
- correctness is fairly straightforward
- main question is efficiency

Challenge

An actor has received *n* offers of movies to star in. The first and last days of filming for each movie is known. Which offers should the actor accept? She wants to be in as many movies as possible but cannot work on two movies whose filming times overlap.

- not too hard to come up with possible algorithms
- correctness in terms of non-overlapping movies is easy to show
- optimality (“as many movies as possible”) is not necessarily straightforward and harder to show – reasonable-looking algorithms can easily be incorrect

Challenge

A robot needs to solder *n* points on a circuit board. The time it takes for the arm to move from one point to another is proportional to the distance between the points. Find the order in which the points should be soldered so as to get the job done as fast as possible. (The arm must return to its initial position when the job is done.)

- trivial to find an easily-proven-to-be-optimal algorithm but impossible* to find an efficient optimal algorithm
- typically must settle for heuristics or approximation algorithms

* under current models of computation, as far as we know

Key Questions

How do we come up with a data structure or an algorithm?

Is the algorithm correct?

How much time/space does it take? Can we do better?

Are there some problems that fundamentally require more time/space to solve than others? Are any unsolvable?

Course Content

- essential tools
  - analysis of algorithms
  - establishing correctness
- toolbox
  - ADTs
    - containers, ordered containers, lookup
  - data structures
    - building blocks: arrays, linked structures
    - key structures: ordered search trees, heaps, hashtables
    - graphs
  - algorithms and techniques
    - searching
    - sorting
    - hashing
    - graph algorithms
- solving problems via modeling
  - it's just a data structure!
  - it's just a graph problem!
  - it's just <fill in algorithm here>!
- design
  - of data structures
  - of algorithms
    - control-flow paradigms: iterative, recursive
    - solution construction paradigms: decomposition, series of choices
    - design paradigms: divide-and-conquer, greedy algorithms, recursive backtracking, dynamic programming
- real world stuff
  - identifying and dealing with hard problems
  - implementation

Goals

- developing the skill of analyzing a problem and creating an efficient and provably correct solution to that problem
  - includes both algorithm development and choice/design of data structures
- fostering an appreciation for the practical value of studying algorithms and data structures
- developing other skills useful in computer science
  - abstract thinking
  - comfort with the idea of tradeoffs
  - a habit of critical reflection and revision

An Observation

"This material is difficult. There is no getting around that."
[Jeff Edmonds, *How to Think About Algorithms*]

- there are lots of open problems!
- a (seemingly) very small change can turn an easy problem into a hard problem
- there are lots of techniques and clever tricks – experience helps a lot
  - (ask if there's something unfamiliar that we don't spend time on)

But we also accept this as a premise – a major focus is on tactics for trying to make progress.

- the goal is to understand various techniques, how to apply them, when they apply, and what you are aiming for even if you still get stuck on some individual problems

Another Observation

"Ask questions. Why is it done this way and not that way? Invent other algorithms for solving a problem. Then look for input instances for which your algorithm gives the wrong answer. Mathematics is not all linear thinking."
[Jeff Edmonds, *How to Think About Algorithms*]

Keeping this in mind will make you stronger at designing complete, correct, and efficient data structures and algorithms.

Prerequisites

- **C- or better in CPSC 225**

This means you are comfortable with Java programming, including –

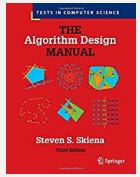
- fundamental programming constructs and concepts (variables, assignment statements, conditionals, loops, static methods)
- input and output, including reading from files
- fundamental OO programming constructs and concepts (classes, objects, inheritance, abstract classes, interfaces, polymorphism)
- constructing a program from specifications, without a lot of specific direction about how to achieve those specifications

This also means you are familiar with –

- basic abstract data types (lists, stacks, queues)
- basic data structures (arrays, linked lists, binary trees)
- recursion

Course Materials

- textbook is *The Algorithm Design Manual* by Steven Skiena, 3rd ed
  - focus is on practical algorithm design
  - also has an extensive reference of data structures and algorithms that are of practical use



- all of the necessary software is available in the campus Linux environment (recommended)
  - accessible in Demarest 002 and via the Linux VDI
  - it is also possible to set up your own computer (optional)

Course Website

http://math.hws.edu/bridgeman/courses/327/s26/

CPSC 327: Data Structures and Algorithms
Spring 2026

Instructor	Stina Bridgeman bridgeman@hws.edu Gulick 203, x3614	**Office Hours**	drop-in office hours: TBD office hours are also available by appointment if you cannot make the scheduled times
Class Hours and Meeting Place	lecture: MWF 8:30-9:30am — Napier 202		

Course Links

- Schedule
  (the course schedule, including links to assignments, readings, slides and examples from class, handouts, etc — pretty much everything you want on a daily basis is here)
- Course Policies
  (expectations and evaluation, attendance, late/makeup work, extensions, academic integrity and collaboration, use of AI, getting help, accommodations, etc — things you should read at the beginning of the semester, then refer back to as needed)
- Course Information
  (course description, textbook information, required materials and software, etc — things you should look over at the beginning of the semester, but probably don't need too often after that)

Documentation and Reference Material

- Accessing Linux

Key Things to Pay Attention To

- in addition to class, expected to attend four lab sessions and four 15-minute interviews (associated with each programming assignment)
- attendance policy
- learning is an iterative process, and practice, feedback, and revision are important elements
  - revise and resubmit policy
- grading
  - in-class exams carry significant weight, and an average of 70 or higher on the exams is required for a C- or better in the course

Key Things to Pay Attention To

- collaboration, outside resources, and use of AI
  - assignments are about learning the process of figuring out the answer, not simply getting the answer
  - your primary resources should be the course materials (textbook, slides and examples from class, things posted on the course webpage) and office hours
    - if you work with others or use other resources, including AI, it should be to explain and understand the course materials, not to produce solutions for assignments
  - unless otherwise specified for a particular assignment, AI is not allowed