Introduction to Algorithms

COMP 3270

Yilmaz

What this course is all about...

Understanding

Analyzing

Designing

Implementing

Algorithms to solve fundamental problems

What is an algorithm?

- A well defined procedure; Like a recipe
  - One or more inputs, finite steps, one or more outputs
  - Criteria for assessing quality:
    - correctness
    - efficiency

Algorithms

- What is the difference between algorithm and software?
  - Software (or a program) is Algorithms translated + data structures

Why study algorithms

- Algorithms affect **ALL FACETS** of your life!
  - your commercial transactions
  - your privacy (or lack of it)
  - how you entertain yourself
  - you could be well-off from algorithms
  - or be one of the richest man or woman on earth!
- Algorithms form the backbone of **ALL COMPUTATION**!

Algorithms

- **Algorithm:** Any well-defined (i.e., a finite number of steps specified at the required level of detail) computation procedure that generally (but not always) takes some value, or set of values, as input and generally (but not always) produces some value, or set of values, as output in order to solve a problem.

- **Example:** Sorting problem
- Input: A sequence of *n* numbers $< a_1, a_2, \ldots, a_n >$
- Output: A permutation $< a_1^{'}, a_2^{'}, \ldots, a_n^{'} >$

of the input sequence such that $a_1^{'} \leq a_2^{'} \leq \ldots \leq a_n^{'}$

Problem Instance & Correctness

- An ***instance of a problem*** consists of one set of valid inputs from which a solution to the problem can be computed.
- An algorithm is said to be ***correct*** if for every input instance, it ***halts*** with the ***correct output***.
- A correct algorithm ***solves*** the given computational problem. An incorrect algorithm might not halt at all on some input instance, or it might halt with other than the correct answer.

Efficiency

- Time: How many steps are executed by the algorithm to solve a problem when it is given an input of a certain size

- Space: How many memory locations are needed by the algorithm to solve a problem when it is given an input of a certain size

- Why are efficient algorithms so important?
  - Inefficient algorithms take too long to compute a solution to **large** instances of a problem

What exactly will we study?

- This course is all about **Computational Problem Solving**, which is:
  - Developing correct and efficient computational solutions to problems.
  - In particular, we will study:
    - Existing algorithms:
      - how they operate, how to analyze their correctness and efficiency
      - how to modify and reuse algorithms to solve problems
    - Techniques for designing new ones; practice algorithm design
    - Implementing algorithms

What exactly will we study?

- The course is divided into two parts:
- Tools for designing and analyzing algorithms.

  First half: The tools you need to understand, analyze, debug, improve, modify, and design algorithms. This part is mathematical, abstract and harder than the second part.

- Algorithms that solve specific computational problems, such as sorting, searching, optimizing…

  Application of the tools to existing algorithms that solve fundamental problems of computing.

So, what should I expect?

Theoretical, somewhat abstract and requires to think about problems and how to solve them in new ways...

Mathematical…but no rocket science!!!

NOT a programming course (but programming homeworks)

Not easy (so plan to spend a lot of time and keep up with the materials; if not, you won't be able to catch up later)

What will I learn?

By the end of the course, you would learn the following skills:

Understand how a given algorithm works
Check if an algorithm is correct
If not, fix it
Calculate the efficiency of an algorithm
If inefficient, modify it to make it more efficient
Given a problem, see if you can reuse an existing algorithm
If not, design a new one
Implement algorithms

Appreciate the “beauty” of algorithms

Algorithms

- “Appreciate the beauty of algorithms” – what in the world do you mean?
  - Beauty is generally in the eyes of the beholder...
  - But NOT in this case!!
  - Algorithmic beauty can be stated in terms of “whether an algorithm correctly and efficiently solves a problem”

How do I make an A?

- Review the lecture slides BEFORE class
- Attend all classes
- Review the slides, your notes and the text AFTER class everyday
- Ask any questions you might have in the NEXT class
- Study with friends
- Ask questions in class
- Practice solving problems – make up and solve problems similar to those in homeworks, or problems at the end of chapters, problems from earlier classes if you can get them from friends
- Do all homeworks by yourself
- Study any supplemental materials

What I will be strict about...

Make-up exams/homeworks: Except in emergencies, I must be informed **beforehand** if you are going to miss something
post-hoc excuses not accepted!

Homeworks must be submitted online via Canvas, unless stated otherwise in the homework…**no late homeworks** or homeworks that do not follow stated requirements will be accepted or graded

If you want a homework regraded, you must ask me **within** a week of when it is returned after grading

Items not picked up **within** a week will be discarded

Cheating of any kind is unacceptable

In Class

No food or drinks of any kind…

Cell phones in off or vibrate mode…answer calls outside class

If you doze off, don' t snore…if your sleep becomes a distraction in the class I will have to act!

Use laptops for taking notes or reviewing class materials; tweeting, facebooking, etc. will be injurious to your grade!

Syllabus

- Two midterm exams

- Homework assignments and a programming assignment

- Final exam: comprehensive
  - Why? Because the analytical tools are the most important take away from this course; you will use these in other courses and in your professional life…the final gives you a second chance to learn these tools.

- Syllabus: On Canvas

Problems of the Day

- Time limited: depending on the questions, you will have 1-5 minutes to solve the problems.
- Late, absent no makeup (unless…)
- Based on recent past lectures

Computational Thinking

- Computational thinking involves
  - solving problems,
  - designing systems, and
  - understanding human behavior,

  by drawing on the concepts fundamental to computer science.

Computational Thinking

- Computational thinking is reformulating a seemingly difficult problem into one we know how to solve, perhaps by
  - reduction,
  - embedding,
  - transformation, or
  - simulation.

Computational Thinking

- Computational thinking is using abstraction and decomposition when attacking a large complex task or designing a large complex system.
- It is separation of concerns.
- It is choosing an appropriate representation for a problem or modeling the relevant aspects of a problem to make it tractable.

Computational Thinking

- It is using invariants to describe a system’s behavior succinctly and declaratively.
- It is having the confidence we can safely use, modify, and influence a large complex system without understanding its every detail.

Computational Thinking

- Computer science is the study of computation—what can be computed and how to compute it.
- Computational thinking thus has the following characteristics:

Computational Thinking

- *Conceptualizing*, not programming. Computer science is not computer programming. Thinking like a computer scientist means more than being able to program a computer. It requires thinking at multiple levels of abstraction

Computational Thinking

- A way that humans, not computers, think. Computational thinking is a way humans solve problems; it is not trying to get humans to think like computers. Computers are dull and boring; humans are clever and imaginative.

Computational Thinking

- *Complements and combines mathematical and engineering thinking*.
- Computer science inherently draws on mathematical thinking, given that, like all sciences, its formal foundations rest on mathematics.

Computational Thinking

- Computer science inherently draws on engineering thinking, given that we build systems that interact with the real world.
- The constraints of the underlying computing device force computer scientists to think computationally, not just mathematically.
- *Being free to build virtual worlds enables us to engineer systems beyond the physical world;*

Computational Thinking

- *Ideas*, *not artifacts*. It’s not just the software and hardware artifacts we produce that will be physically present everywhere and touch our lives all the time, it will be the computational concepts we use to approach and solve problems, manage our daily lives, and communicate and interact with other people

Computational Thinking

- Intellectually challenging and engaging scientific problems remain to be understood and solved.
- The problem domain and solution domain are limited only by our own curiosity and creativity.