Welcome to CSE 431/531

- Class Name: Algorithm Analysis and Design.
- Place: 322 Clemen
- Time: T & Th 2:00-3:20pm
- Semester: Fall 2004

Upper Undergrad. - Beginning Grad. course about:

- Elements of the design and analysis of algorithms
- Discrete Math. problem solving skills essential for computer engineers and scientists
Today’s Agenda

- Administrative Matters
- Brief Overview of the Course
Who should take this course?

Anyone who is either

- a computer science/engineering student

- interested in getting to know the most fundamental area of Computer Science

or

- forced to take it because it’s required and/or all other courses were filled up

The catches are

- Data Structures (CSE250 or equivalent),

- Calculus II (or some formal calculus/analysis course),

- and a course which requires formal proofs (discrete Math.)

... not crucial if you’re motivated enough, though.
Who should teach this course?

- Hmm ... as if you have a choice
Teaching Staff

Instructor:

- **Dr. Hung Q. Ngô**

- Assistant Professor, CSE-SUNY Buffalo

Teaching Assistants: Ph.D. candidates

- **Mr. Guang Xu**: recitation section **A2**, Wednesday 3:00pm-3:50pm, 220 Clemen

- **Mr. Zhiyong Lin**: recitation section **A1**, Friday 11:00am-11:50am, 213 Obrian

- Recitation **A3** (Tuesday 4:00pm-4:50pm, 101 Baldy) will be held alternatively between the two TAs.
When/Where to talk to me?

Algorithm 1 (Your First Algorithm). To ask the Prof. a question, try

1. send questions to class newsgroup sunyab.cse.531
2. email him (hungngo@cse.buffalo.edu)
3. come to office hours - 238 Bell Hall, 10-12am Thursdays
4. sneak in whenever the door is opened
5. goto 1
Course objectives

- Have fun learning!

- Grasp a few essential ideas of algorithm analysis and design
  - asymptotic notations and analysis
  - basic parallel sorting networks
  - typical algorithm design methods: divide and conquer, greedy, dynamic programming, network flows analysis
  - basic graph algorithms: BFS, DFS, MST, ...
  - the notions of NP-Completeness and approximation algorithms

- Gain substantial problem solving skills in designing algorithms and in solving discrete mathematics problems
Course Materials

Required textbook:


Online Materials: class website (see syllabus for URL)

Recommended references

- Knuth’s Classic three volume *The Art of Computer Programming*.

Work Load

- Heavy! So, start early!
- Approx. 30 pages of dense reading per week
- 5 written homework assignments (to be done individually)
- 1 midterm exam
- 1 final exam
Grading Policy

- 5 assignments: **8%** each
- 1 midterm exam: **25%**
- 1 final exam: **35%**

Note:

- Assignments are due at the lecture’s end on due date
  - 1 day late (24 hours): 10% (of max score) reduction
  - each extra date: 30% more
- Incomplete grade and make-up exams: **not given**, except in **provably extraordinary** circumstances
**Academic Honesty**

Absolutely no tolerance on plagiarism

- 0 on the particular assignment/exam for first attempt
- Fail the course on the second **plus** report to department and school
- Consult the University Code of Conduct for details
- In summary, I will take plagiarism very seriously

Note:

- You are encouraged to discuss class materials and homework problems with classmates
- The final writeup **must** be on your own, in your own words
Absolutely No Lame Excuses

- I have to go home early, please let me take the final on Dec 1.
  ... NO, NO, NO, NO, not even one day before the actual exam.
  (Do you know how long it takes me to come up with a good exam?)

- I had a fight with my girlfriend
  ... yeah, right

- I’ve studied hard, I understood the material very well,
  ... but I got a C. Please consider giving me A-
  ... you’re funny

- I think I deserve a better score, please give me some work to do next semester to improve the score
  ... sorry, I have no time.
How to make it more interesting?

How to do well in this course?

- Ask questions in class
  The only stupid question is the question you don’t ask
- Suggestions are always welcome
- Attend lectures
- Do homework/reading assignments early
- At least, skim through reading assignments before lectures
- Print out lecture notes before lectures

We, the TAs and I, are here to help you. Don’t hesitate to ask.
A few motivating examples

Example 1 (Fibonacci numbers). Write an algorithm to calculate the $n$th Fibonacci number, given $n$

\[
F_0 = 0 \\
F_1 = 1 \\
F_n = F_{n-1} + F_{n-2}, \ n \geq 2
\]

Example 2 (Primality testing). Given a natural number $n$, return YES if it is a prime number, NO otherwise.


http://www.cse.iitk.ac.in/primality.pdf

Example 3 (Shortest Path). Devise an algorithm to find a shortest path from a source (e.g. your computer) to a destination (e.g. www.nfl.com) in the Internet

Example 4 (Steiner Tree). Given a set of cities, find an algorithm to assist in building a highway system connecting all these cities, so that that total length of highways is minimized.
Aha - Algorithms!

Algorithm 2 (FibA). *Input:* non-negative integer $n$.

1: if $n \leq 1$ then
2: return $n$
3: else
4: return $(\text{FibA}(n - 1) + \text{FibA}(n - 2))$
5: end if

Algorithm 3 (FibB). *Input:* non-negative integer $n$.

1: if $n \leq 1$ then
2: return $n$;
3: else
4: $a \leftarrow 0; b \leftarrow 1$
5: for $i$ from 1 to $n - 1$ do
6: temp $\leftarrow a; a \leftarrow b$
7: $b \leftarrow \text{temp} + a$
8: end for
9: return $b$
10: end if

Question: What are the pros and cons?
Analyzing Algorithms

- mean of “roughly predicting” the resources required

- Resources:
  - How fast?: time complexity
  - Memory requirement?: space complexity
  - Others: communication bandwidth, hardware costs, ...

Need a specific machine model: RAM, parallel computers, quantum computers, DNA computers, ...

- We’re mostly concerned with time complexity: a rough estimate of running time wrt the input size

- We will be very informal until NP-completeness is discussed
Designing Algorithms

I assume you know what it means.

Approaches

- Ask someone
- Hack around ’til it works
- Brute force
- Incremental
- Divide and conquer
- Greedy
- Dynamic programming
- Formulate the problem as a network flow, linear/non-linear programming problem
- A stroke of genius
- Give up

Note: “programming” is not programming
Lastly

- Hope to learn as much from you as you’d learn from me
- Enjoy the ride!

Next time:

- Growth of functions
- Solving recurrences