Competitive Programming

CS 104C

Introductions

I am **Prof. Etienne Vouga**

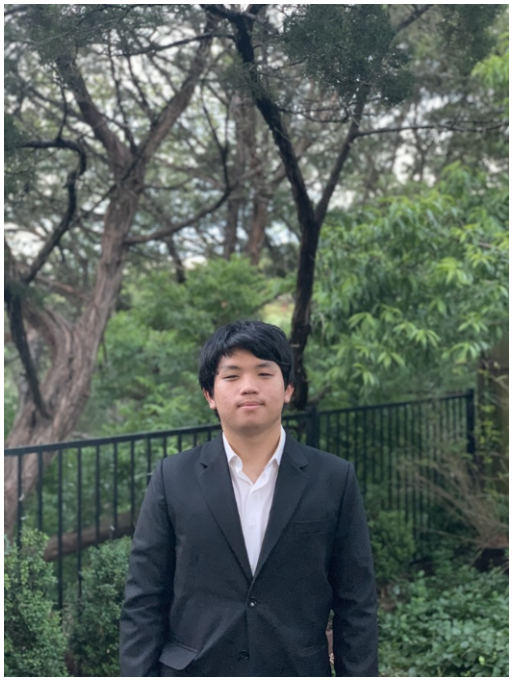
- evouga@cs.utexas.edu
- GDC 5.508

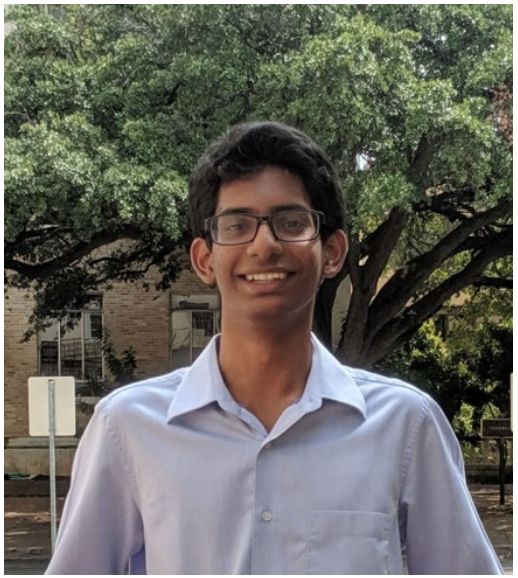
Coach of the ICPC teams

Office hours by appointment

Fellow Lecturers

Aditya Arjun, **Kevin Li**, and **Viraj Maddur**





Other Affiliated Faculty

Glenn Downing

Shyamal Mitra

What is Competitive Programming?

Example Problem

Input: An integer $\mathbf{N}$ $(0 \leq \mathbf{N} \leq 10^6)$

Output: The number of zeroes at the end of $\mathbf{N}!$

Time limit: 1 sec

Memory limit: 10 MB

What is Competitive Programming?

Given a concrete problem statement:

1. Analyzing the problem and determining what algorithm can solve it;
2. Identifying the possible pitfalls and corner cases;
3. Quickly producing an implementation that "Just Works the first time"

In other words, *efficiently analyzing and solving low-level programming problems*

	High-Level Design	Low-Level Design
Unit of Concern	Application; Library	Function
Main Goals	User Experience Maintainability	Correctness Performance
Key Questions	What libraries can I use? Will these features be useful? Will this design scale to more users/data?	What algorithm can I use? What are the corner cases? Will this run in reasonable time and memory?
Skills Needed	Planning Communication	Problem-solving Algorithms knowledge
Scope of Effort	Teams working for months	One programmer working for hours

Benefits of Competitive Programming

Improved problem solving skills
Improved knowledge of algorithms

Write more optimized code
Write less buggy code

Great practice for technical interviews
Have fun, win prizes and glory

Mechanics of a Competition

You receive several (3-10) problems of varying difficulty

Write solution, submit it to an **automated judge** that runs test suite

Mechanics of a Competition

You receive several (3-10) problems of varying difficulty

Write solution, submit it to an **automated judge** that runs test suite

Get back ~2 bits of information:

Accepted, Wrong Answer, (Compile Error), (Time Exceeded), (Memory Exceeded)

You are **not** told the failure cases!

Two Types: Online and Offline

Online: (HackerRank, TopCoder, CodeForces, Google CodeJam)

- Individual
- Can use Internet, your old code, …

Offline: (ICPC)

- Team (shares one computer)
- No Internet

Competitions vs Industry

In competitions:

- Severe time pressure
  - no time for unit tests
  - no time to write documentation
- Style and code quality doesn’t matter*
- Maintainability doesn’t matter
- Problems self-contained

Competitive Programming Considered Harmful?

```
#include <bits/stdc++.h>
#define rep(i, a, b) for (int i = (a); i < (b); ++i)
#define trav(a, x) for (auto& a : x)
#define all(x) begin(x), end(x)
#define sz(x) int(x.size())

using namespace std;
using vi = vector<int>;
using vvi = vector<vi>;
using ll = long long;
using ull = unsigned long long;
using vll = vector<ll>;
template <typename T> void mini(T& x, T y) { x = min(x, y); }
template <typename T> void maxi(T& x, T y) { x = max(x, y); }

ll V, X;
ull M;
```

If you cannot do high-level design, you are **not a programmer**

If you cannot do low-level design, you are an **incompetent programmer**

Example Problem II

Input: A **sorted** list of $\mathbf{N}$ integers and an integer $\mathbf{x}_i$

- $\mathbf{y}$
- $1 \leq \mathbf{N} \leq 10^6$

  $-2^{30} \leq \mathbf{y}, \mathbf{x}_i \leq 2^{30}$

Output: The number of elements in the list strictly less than $\mathbf{y}$

Time limit: 1 sec

Memory limit: 10 MB

Efficiency Rule of Thumb

complexity	maximum N
(N)	100 000 000
(N log N)	40 000 000
(N^2)	10 000
(N^3)	500
(N^4)	90
(2^N)	20
(N!)	11

Lessons?

Read the problem statement

- Look at the limits
- Check for corner cases

Almost always a performance vs complexity tradeoff – choose carefully!

Big-O critical (but don’t sweat log N)

Other Tips for Getting Started

Become proficient in **one** language, and know its libraries and I/O functions cold

- C++, Java, Python

Get out there and code

- try out online contests: codeforces, codechef, projecteuler, hackerrank
- you greatly improve by practicing

Assignment and Grading

75% homework exercises (3/week):

- **Vanilla Problem**: tests key concepts
- **Codeforces Exercises**
  - online judge
  - we pick three problems, you choose one
- **Find the Bug**
  - our solution gets “wrong answer”;
    find the problem

Assignment and Grading

75% homework exercises (3/week)

25% programming contest participation

- every two weeks on Friday night
- first contest: **Sept. 11** 5:30 pm

One contest (your choice) is required

Can do more for extra credit

Assignment and Grading

Questions about the class logistics?

Check the web site first:

https://www.cs.utexas.edu/users/downing/cs104c/

Contains:

- syllabus and grading breakdown
- academic integrity policy
- harassment-free conduct policy

Assignment and Grading

Questions about the class logistics?

Check the web site first:

https://www.cs.utexas.edu/users/downing/cs104c/

Ask on Piazza / Email TAs

Prerequisites

Data Structures (or equivalent)

Working knowledge of Java (for **Find the Bug** problems)

Strong working knowledge of one of:

- C/C++, Java, Python

Tentative List of Topics

- state space search / graph algorithms
- recursion / backtracking
- binary search
- greedy algorithms
- dynamic programming
- advanced graph algorithms
- number theory
- probability/combinatorics

Coming Soon on Canvas/Piazza

Links to online competitions

Instructions for setting up Codeforce account (needed for assigments)

Codeforces problems for the semester