

EECS 281

Data Structures and Algorithms

Prof. Marcus Darden

Dr. Héctor Garcia-Ramirez

Dr. David Paoletti

Winter 2024

eeecs281admin@umich.edu

Teaching Assistants

Ammar Ahmed (GSI)
Daniel Chechelnitzsky (GSI)
Finn Roblin (GSI)
Hrithik Ravi (GSI)
Jifeng Wang (GSI)
Max Tennant (GSI)
Patrick Li (GSI)
Sarah Stec (GSI)
Abbas Fattah
Alexei Chen
Alexia Moreno
Andrew Mahler

Angelina Ilijevski
Aray Trujillo
Caroline O'Sullivan
Christine Zeng
Denaly Min
Efe Akinci
Emily McNichol
Eray Sabuncu
Gary Huang
James Maddock
Jason Chen
Julianne Park

Junjie Shen
Kazu Sakamoto
Keqian Wang
Kwan Ting Lau
Lais Najjar
Mallika Miglani
Megan Parada
Olivia Rui
Rishith Seelam
Ryan Chua
Swaraj Yadav
Yolanda Zhou

Course Weekly Schedule

Lectures

- Tuesday / Thursday
- 9am-3pm, 4:30-6pm, 5 in-person lectures
- Lecture 006 is remote only
- Important announcements in lectures

Labs

- See the Schedule of Topics on Canvas
- No labs during the first week

All lectures and labs cover the same material
Each instructor will have one lecture recorded
Lecture sections 1-5 will be streamed via Zoom

Important Dates

- **NOW: 281 Crash Course:** posted on YouTube, see next slide
 - Submit Project 0
- **Midterm:** Tuesday 3/5, 7-9pm
- **Final:** Thursday 4/25, 8-10am
- **EXAMS ARE IN PERSON ONLY**

281 Crash Course

- We used to do [one long session](#) (in person) and record it, but we made separate videos during COVID
- Watch one of the first 3, and the last 2:
 - [Visual Studio and WSL Tutorial](#)
 - [Xcode Tutorial](#)
 - [Visual Studio Code Debugging and WSL](#)
 - [Project 0 Tutorial](#)
 - [Makefiles Tutorial](#)

Syllabus

- Please read the syllabus on Canvas
- This lecture summarizes:
 - Course policies (prereqs, collaboration, honor code, office hours, etc.)
 - Assignments and grades
 - Tips for success
 - Organization of Topics

Prerequisites

- We enforce prerequisites: 203 and 280
 - If you enrolled in EECS 281 before receiving grades in EECS 203 or 280 that do not allow you to enroll in EECS 281, you must drop 281
- For EECS 203, we count Math 465 and 565 (graph theory, combinatorics, etc)
- Per Departmental Policy, grad students cannot register for, or audit, EECS courses below 400-level (including EECS 281)

Graduate Students

- Are enrolled in EECS 403
- Do same projects and labs
 - Have extra submits per day
- Take same exams
- Grades determined completely separately from 281 students

Topic Preparedness Self-survey

- There is a short survey on Canvas (Quizzes, under Practice Quizzes)
 - Multiple choice
 - Assessment of prerequisite material
 - Will not affect your course grade
 - Gives you practice on the Canvas “Quiz” tool
- Will help you decide whether you are adequately prepared for EECS 281

Policy on Collaboration

- All work submitted for Projects and Exams must be your own
- You may use source code provided by EECS 281 instructors
- You may reuse YOUR OWN CODE if you are retaking EECS 281
- If you use other code and try to obscure it, we have automated ways to detect that

Policy on Collaboration

- Do not show your project code to others
 - Do not post code on Piazza
 - Do not use open online repositories (github, etc.)
- Do not share project test files; only submit your own test files
- When in doubt, **ask us** (using Piazza or come to office hours)

Honor Code

- Read Honor Code (link is on Canvas)
- Please know that we take this very seriously
- We automatically check electronic submissions for violations of Honor Code
- We (teaching staff) are the 'traffic cops'. Honor Council is the 'court of law'

OK to use Wikipedia, Google, etc.?

- Yes, it is – to understand algorithms and data structures covered in lecture
 - External sources must be mentioned in labs & projects for credit assignment reasons
 - We do not accept external references *to justify answers* on labs, exams, etc.
 - Don't copy+paste from GitHub!

Getting Help & Contacting Us

- For *urgent & personal* issues
 - eeecs281admin@umich.edu readable by faculty and a few staff
- For *really personal* issues –
 - Email there and ask to make an appointment
- <http://cppreference.com>
- <http://www.cplusplus.com/>
- <http://piazza.com>
 - Do not post code from lab and project solutions
 - Do not ask if your solution is correct
 - You can post anonymously to other students
(but we will know your name)
 - Students can answer questions of other students
 - Instructors endorse good answers
- **Please “close” your questions once answered**



eeecs281admin Email

- Seen by faculty and a few select staff
- You can feel safe contacting us with medical issues, kept entirely confidential
 - Everyone goes through FERPA training
- If you don't get a sufficient response or sufficiently quickly, come to Proffice Hours
 - Offered almost every weekday
 - Ask to speak to us individually

Office Hours

- Come prepared with specific questions
 - Conceptual is fine
 - If code-specific, please have input that your program does work with, and input that it does not
- Attend soon after the project is assigned and get conceptual questions answered before you start coding
- Sign up at <https://eecsoh.eecs.umich.edu/queues/>
- Please respect other students
 - Ask one question, then move to back of line
 - Can listen to other student's questions, as long as not personal in nature
 - If you hear someone that has the same issue that you already solved, feel free to tell them, **in general**, about the problem and solution!

Office Hours

- **Staff Office Hours Etiquette**
 - Will be posted on the Google Calendar by the end of the first week
 - Please respect course staff availability, as TA's are students too
- **Professor Office Hours Etiquette**
 - Always available during scheduled office hours
 - Sometimes available for quick questions (1-2 min) when office door is open
 - Can schedule time outside of posted OH for personal matters
 - Not available when office door is closed
 - Not available during undergrad advising hours

Office Hours Queue

- Come to Proffice Hours!
 - Join the Google Meet for any question that doesn't need you to show us code
 - This discussion helps everyone in the room
 - Listen to other students' questions
- Don't join an 80-person queue instead of Proffice hours meet!
- Join the help queue when you need 1-on-1 help
 - Code won't compile
 - Need to talk about your course grade

Grading

- Grading Policy
 - 20% Labs (10)
 - 40% Projects (4)
 - 20% Midterm Exam
 - 20% Final Exam

What Guarantees that I Pass?

- Achieve minimal competency
- If you earn **ALL OF:**
 - ($\geq 50\%$ on Exams)
 - AND** ($\geq 55\%$ on Projects)
 - AND** ($\geq 75\%$ on Labs)
- You **WILL** pass this course
- A total of 68, with 30% projects, 100% labs and 90% exams is **NOT PASSING**

Labs (20%)

- 10 lab assignments
- **Can work with other students**
- Submit on paper (in lab), electronically via Canvas, and/or autograder machine
- Late submissions for **Quizzes & AG** assignments are accepted at 50% credit
 - Up to midnight of the day before each exam
 - Do not ask for extensions
 - Cannot use late days

Lab Times

- Labs meet Tuesday – Monday every week
- You do not have to attend the lab that you're enrolled in
- We would like you to attend the same lab each week
 - Make contacts and consistent partnerships

Lab Written Portion

- Every lab has a “written” problem
 - Done on paper, during lab
 - This is practice for the exams
 - It is graded by effort
 - You can miss up to two at full credit
- These problems will prepare you for writing code by hand on the exams, in interviews, etc.

Lab Partners

- No need to “register” partnerships
- All students must submit all parts individually to receive points!
 - The in-class written problem is done in person!
 - Talk to your partner or even other students
 - Discuss the problem and approach to solving it
 - You MAY NOT submit for your absent partner

Projects (40%)

- 4 projects
- **Individual work**
- Submitted electronically to autograder
 - Details to follow
- Approximately 3 weeks per project
 - Less in Spring
- Late submissions: USE LATE DAYS WISELY (see “Policy on Deadlines”)

Policy on Deadlines

- Autograder: 2 Late Days per semester
- Use them as you want
- Project 0 late days are “free”, use them for practice!
 - Before any “real” assignment is due, everyone will have their late days restored
- Example: if a Project was due Tuesday, today is Thursday; you didn't submit yesterday = 2 late days to submit today (submitting 2 days late)

Projects (40%)

- C++ (International C++11 Standard)
 - <https://en.wikipedia.org/wiki/C++11>
- CAEN Linux Computing Environment
 - g++ (GCC) 11.3.0
- Beware if you are doing development in any other environment
 - May compile/run perfectly for you, then not even compile on the autograder

Autograder

- We will grade projects with an autograder
 - Correctness, timing and memory usage
- Immediate feedback on most test cases
- ~3 submissions per day
 - Some projects have more, some have two parts
 - +1 submit per day for finding enough bugs!
 - More in Spring (due to double speed)

Autograder

- **DO NOT WAIT** until the last minutes to submit right before midnight
 - There may be so many people trying to submit that you can't
 - If you're unable to submit because of this, that's a reason late days are provided
- When there's a tie for best score, we use the most recent best score for final grading
 - If you want us to use your final submit (even if it's not the best), there's a form linked in the project spec and in the AG FAQ

Before Debugging Help

- Before getting help debugging, you should have:
 - Submitted to the autograder
 - Included test files of your own
 - The autograder will tell you if your own test file reveals your solution as buggy!
 - Tested all provided examples using valgrind
 - Found as small a test as possible that reveals your bug




P F15P2 Project 2 - Office Hours of the Dead

- **Due date:** October 23, 11:59:59 PM
- **Today's used submits:** 0 / 3
- **Late days remaining:** 2 (Not usable)
- [View scoreboard](#)

Upload submission:

No file selected.

Upload submission

	Timestamp	Score	Passed	Bugs caught	L1m	L1s	L2b	L2p	L3m	L3s	L4b	L4p	L5s	L6b
	15.12.22.115737*	67.5	26	18	0.003	0.004	WA	WA	0.004	0.005	0.003	0.003	1.555	0.224
	15.12.22.114935*	44.5	26	18	0.003	0.004	WA	WA	0.004	0.005	0.003	0.003	1.557	0.226
	15.12.22.113848*	47.0	28	18	0.003	0.004	WA	WA	0.004	0.005	0.003	0.003	1.520	0.338

Timestamp* Submission didn't count toward your daily limit.

N/A Not available: the test didn't run.

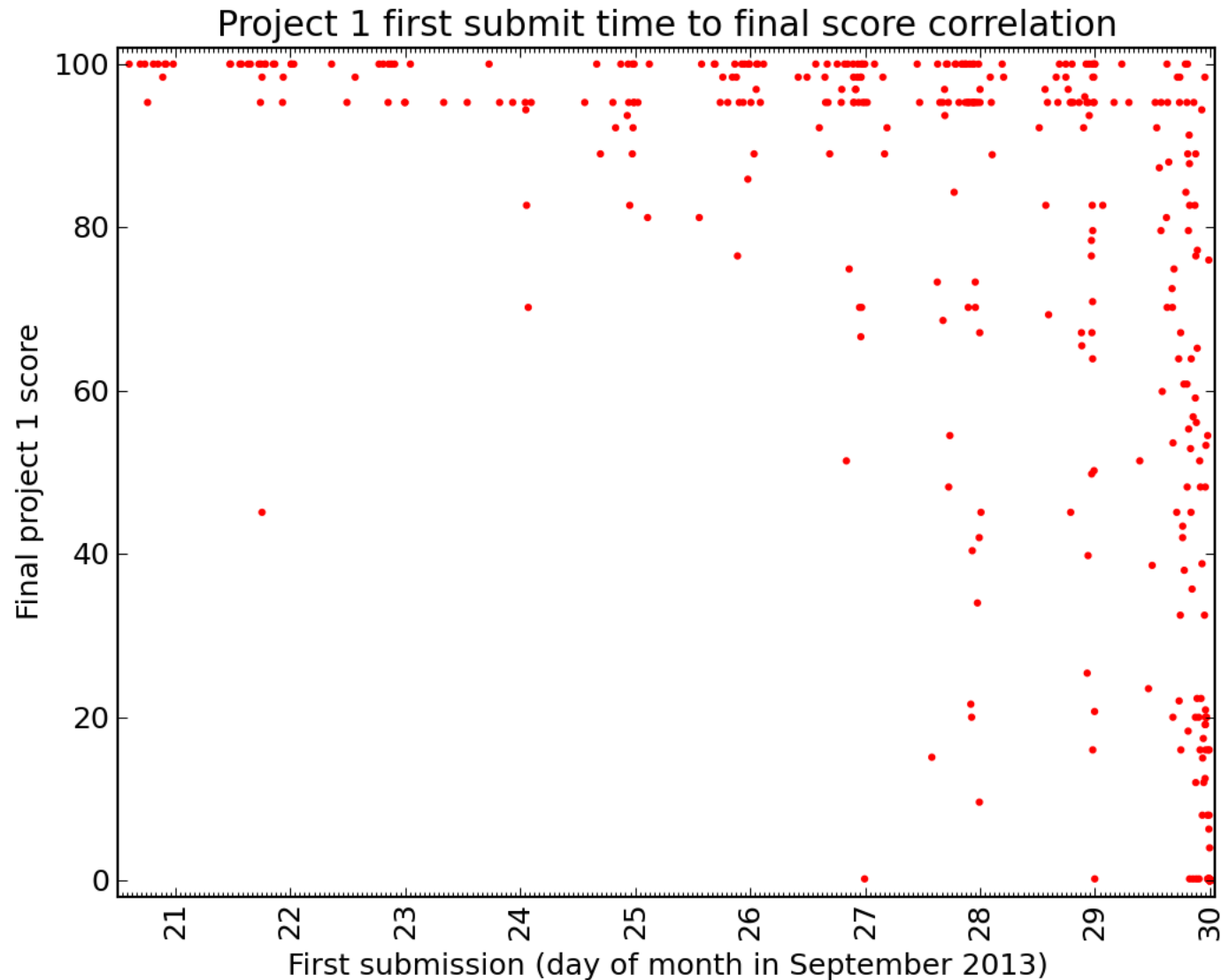
WA Wrong answer: your answer was incorrect or incomplete.

TLE Time limit exceeded: your program exceeded the time limit for this test case.

SIG Signal: your program encountered an error (segfault, exception, assertion failure, ran out of memory, etc.) and exited.

- Red tests failed (wrong output or exit status)
- Blue tests are over time and/or memory

Submission time vs. score



Exams (40%)

- Midterm Exam (20%)
- Final Exam (20%)
- Will test your **understanding** of material and **problem-solving skills**
- Both a multiple choice section and a long answer section
- Must notify instructor 2 weeks ahead if conflict
- Cannot miss exam without documented serious medical or personal emergency

Curving

- If necessary, we will change the points required to pass the exams
 - Will only make it easier to pass, never harder
- May also adjust overall grades
 - May adjust grades upward if needed, never down
- You can calculate what you need on the final to pass
 - Have to pass both exams and projects
- **Let us know of any concerns early**

Will Solutions Be Posted?

- Yes - for labs (see Canvas after the due date)
- No
 - For in-class exercises (some yes, some no)
 - For projects
 - For exams
- Midterm solutions may be outlined in class
- Clarifications on Piazza and office hours

Lectures

- You can print out (or use digital version) lecture notes, and must go over them
 - Before the lecture – to prepare questions
 - After the lecture – to make sure everything was clear and review for exam
- Take notes during lecture!
 - Studies demonstrate that students taking notes longhand remember more and have a deeper understanding of the material
 - <https://www.scientificamerican.com/article/a-learning-secret-don-t-take-notes-with-a-laptop/>

Lectures

- Not all material presented in lecture will appear in the lecture slides
 - Explanations on a tablet
 - Additional practice questions
- If you are not following lecture material, don't wait until just before the exam
 - Ask questions, attend office hours

What Can I Do To Fail?

- Fall behind
- Use Copilot
 - The written exams require writing code, by hand, on paper
 - Only resource is your brain
- How do you think companies will run coding interviews?
 - a) Can you write comments to generate code?
 - b) Can you actually code?

What Should I Do To Succeed?

- Be serious and organized, stay sharp
- Allocate sufficient time for this course
- Be proactive: do Project 0!
- Prioritize tasks -- don't waste your time
- Don't get stuck, do ask for help
- Practice writing code by hand! To prepare for the exams, treat Labs, projects, etc. as exam questions

Computing CARES

- View their website here:

http://www.eecs.umich.edu/eecs/about/articles/2015/Computing_CARES.html

- 281 Video:

<https://youtu.be/5MkRjP9qpKY>

How Many Hours Per Week?

- **It varies widely**, based on
 - How well you remember EECS 280 material
 - Makefiles, library functions, debugging, using headers properly, etc.
 - Same with EECS 203 material
 - Counting, induction, complexity, summations, graphs
 - Following our directions
 - How well you plan?
 - Do you need to redo things?

Study Groups

- Generally, a great idea
 - You will not overlook important material
 - Someone can fill you in on a lecture you missed



- Downside: your project code may look similar and will attract extra scrutiny

Useful tools

- Automated compilations
 - make
- Editors for “power users”
 - Vim, Emacs
- Version control system
 - Git (use private repositories only!)
 - <https://github.com/>
 - <https://gitlab.eecs.umich.edu>

Making Copies of your Code

- Suppose you DON'T do any of the following things (the first 3 which we suggested):
 - Upload to the autograder
 - Upload to CAEN to test building with g++
 - Upload your code to a git server
 - Copy your files to a flash drive
- Then your computer dies...

Don't let This be You



IDE

- One platform allows the use of multiple tools through a single interface
 - Text editor
 - Many have tooltip popups for method parameters
 - Some detect errors while typing
 - Advanced code browsing (look up method definitions, jump directly to them from a call)
 - Project management/make
 - Compiler, debugger, profiler
 - Some include version control

Partial List of IDEs

Proprietary

- Visual Studio 20XX, Enterprise or Community
 - [Enterprise edition](#)
 - [Community edition](#)
 - C++, C#
 - PC only
- Xcode (free)
 - [apple.com](#)
 - C++, Swift, Objective-C
 - Mac only

Multiple Platforms*

- NetBeans (free)
 - [netbeans.org](#)
 - C++, Java, etc.
- Eclipse (free)
 - [eclipse.org](#)
 - C++, Java, etc.
- VS Code (free)

*Need a separate g++ compiler such as Cygwin or Min-GW

Plotting Tools

- Useful for plotting algorithm statistics
 - Runtimes
 - Memory Usage
 - Other parameters
- Gnuplot (installed on CAEN Unix)
 - <http://www.gnuplot.info/>
- Google Sheets
- Excel (installed on CAEN Windows)
 - <http://www.usd.edu/trio/tut/excel/>
- Matlab (installed on CAEN Windows)
 - <http://www.math.ufl.edu/help/matlab-tutorial/>

Pre-Midterm: Foundational Skills & Techniques

- Complexity analysis of algorithms
- Building blocks – elementary algorithms & data structures
 - Sorting, searching, stacks and queues, priority queues (+ possibly more)
- Implementation in C++11 using STL
 - How to be efficient, what to avoid
- Time measurement and optimization
- Algorithmic problem-solving
- Examples for how to select the best algorithm for a problem

After the Midterm: Sophisticated Algorithms

- Binary search trees (dictionaries)
- Hashing and hash tables
- Graph algorithms
- Algorithm types
 - Divide-and-conquer
 - Greedy
 - Dynamic programming
 - Backtracking and branch-and-bound

Data Structures and ADTs

- Need a way to store and organize data in order to facilitate access and modifications
- An **abstract data type (ADT)** defines a collection of valid operations and their behaviors on stored data
 - e.g., insert, delete, access
 - ADTs define an interface
- A **data structure** provides a concrete implementation of an ADT

Algorithms

- An **algorithm** is a well-defined procedure that solves a computational problem
 - Transforms given input data into desired output or answer
- “Recipe” or “Set of Directions”
- Algorithms are tools for solving problems
 - Sort a list of data, find the shortest path between classes, pack as many boxes as possible in a delivery truck

Analyzing Data Structures and Algorithms

- When designing algorithms and DSs, we care about:
 - How long does an operation take (# of steps)?
 - How much space is used?
- Predict answers before running the code
 - Avoid wasting time on bad designs
- **Complexity analysis** answers these questions relative to the size/quantity of input data

Algorithm Engineering

- For a given application, is it better to use:
 - Algorithm A or algorithm B?
 - Data structure X or data structure Y?
 - Often you can tell before writing any code
 - Sometimes you need an empirical comparison
 - Sometimes the answer is surprising
- For a given piece of code:
 - Where is the bottleneck?
 - How do you speed it up?

Algorithm Exercise

1. Write this function
2. How many multiplications will it take if size = 1 million?

```
//REQUIRES: in and out are arrays with size elements
//MODIFIES: out
//EFFECTS:  out[i] = in[0] *...* in[i-1] *
//          * in[i+1] *...* in[size-1]
void f(int *out, const int *in, int size);
```

Developing Your Skills

- Problem-solving
- Algorithm analysis
- Software development
- Practice, repetition, and rewriting
 - Building skills
- Memorization
 - Not necessarily rote!
 - Required for speed in programming
 - Building blocks and processes

Be a Good Software Engineer!

- When is a given technique appropriate?
 - Pointers (or references), classes, STL
- Good code versus bad code
 - Modular, concise, readable, debuggable
- Functional robustness
 - Input checking, assertions, etc.
- Code reuse: less work, less debugging
- Write code to avoid and minimize bugs