CSC 148

Week 12

Sadia ‘Rain’ Sharmin

Classes begin 10 minutes after the hour

Administrative Stuff

Announcements

Marking stuff

We will announce when A2 grades are available.

Remark requests for the midterm

- We will announce when these have been completed

Other requests to the course account

- We are quite caught up on these

Announcements

Course grades

Will likely not be available until early May

We are super keen to get these done asap, so will be going as fast as we can

Announcements

Office hours

The regular schedule of instructor and TA office hours ends on Friday.

We will be holding a series of exam prep sessions instead.

See Quercus for the schedule.

Announcements

Exam Prep sessions

YOU will decide the content of these. We encourage you to make requests in advance for:

- Old test questions you'd like solved. Tell us which test (what term, midterm or final) and which question.
- Topics you'd like to have reviewed.
- Any other questions you have about the course.

You are also more than welcome to attend and just listen.

Announcements

Course Evaluations

Please take time to fill yours out.

We especially appreciate written comments.

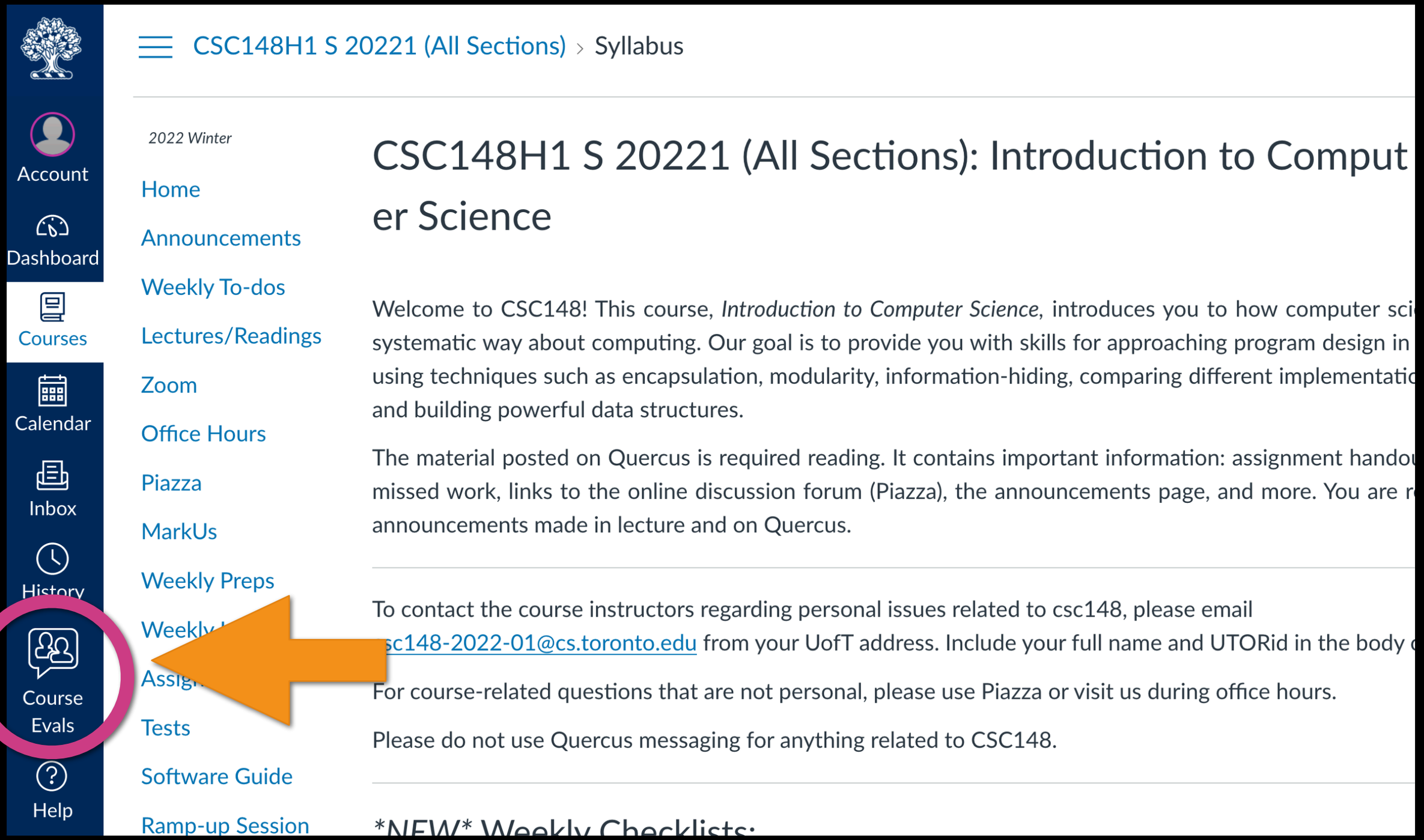
Announcements

Course evaluations matter!

These are read by instructors, chairs, deans, the Provost and the President! They are used to:

- Help instructors improve their courses
- Help the department identify problem areas
- Provide students with info about courses
- Assess professors for annual performance reviews
- Assess professors for tenure and promotion
- In nominations for teaching awards

You can see evals on Quercus



The Final Exam

The Final

Coverage

Comprehensive (covers the whole term), including:

- Class design
- Inheritance
- ADTs in general; stack and queue in particular
- Trees, BSTs, linked lists
- Recursion
- Recursive sorting
- Time-efficiency and big-O analysis
- Memory models
- Use of preconditions, assertions, representation invariants

General tips: How to prep

The Final

I read all the slides and examples, I'm ready! ✗

I did the labs at the time, I must be ready now! ✗

I did all the examples myself from scratch, without looking at the solutions and I got it right!

plus

I did extra examples and looked into the documentation when I was stuck, then solved the extra work! ✓

The Final

How to prepare

Re-solve parts of the assignments that your partner did.

Re-do / complete lab exercises (without looking at solutions!).

Run examples from class. Modify them and explore the impact. ("Sandbox" approach! Play with your code.)

Test out your “what if” questions / theories.

Make up your own problems.

Focus on topics you aren’t fully confident in.

Solve old tests and finals.

Note: Some use content or corners of Python we didn’t touch

The Final

How to prepare

1. **Concept mastery**: definitions, examples, connections
2. **Code mastery**: make a plan, break down tasks into small chunks, use code templates
3. **Practice in a test-like environment**: pencil and paper, closed book, timed environment

The Final

Aim for mastery

"Mastery" means that **you** are the expert.

Go beyond "re-reading and re-doing":

create multiple solutions

identify common mistakes or errors, and

explain ideas to others

make up new questions

The Final

Informal study groups can help a lot!

Study groups can challenge each other, critique solutions

Here is a message you can use to gather an informal study group:

> *Hey folks,*
>
> *Anybody want to study for the final together? I'm going to be in [location on campus] on April _____ from ____ onwards, feel free to join! I'm looking to do really well on the final, so some serious studying will be happening. I'll be the one wearing a purple shirt with an Angry Birds tie.*
>
> *P.S. This post is so ironic because I copied exactly what the instructor showed us in class, lol.*

Recursion

How to prepare

Did I mention practice?

The Final

During the exam

Do not panic! Take a deep breath, you've got this!

- This is your chance to show us what you've learned
- We WANT to give you the credit that you've earned

Read carefully!

- What is the question asking?
- Don't confuse things
- If there's anything unclear, please ask!

Keep track of your time

- Some questions take more time than others
- Do not spend too much time on a question if you are stuck – might want to revisit it later

Final

The

During the exam

As you may have seen on past exams, we'll provide an aid sheet. It is posted on Quercus.

There is very little to memorize.

It's about understanding, not regurgitation.

Helpers are always welcome, unless we specifically say otherwise.

Comments are not necessary, unless we specifically ask for them, but could help us understand your reasoning for complex code.

This Week's Cool Programmer Feature

This Week's Cool Programmer Feature: *YOU!*

in review

You are no longer a novice programmer

You have written significant Python code and covered a broad range of practical experience, including:

- Working with code across multiple files and classes
- Using code-writing tools like PyCharm
- Testing and profiling tools (pytest, hypothesis, etc.)
- Useful Python libraries (e.g., pygame)

Semester

in review

Your software tools

You've started to gain proficiency with

- A professional IDE
- A unit testing framework
- A property-based testing framework
- The debugger

This is part of what makes you a professional.

*More on this in **csc207**, Software Design.*

Semester

in review

Semester

Your thinking tools

You've also added new tools for thinking about:

- Data
- Algorithms
- Code design
  - Analysis of correctness
  - Analysis of algorithms

Knowing when and how to use them is part of what makes you a computer scientist.

The journey is just beginning!

What have we learned?

in review

Semester

1. Abstraction

in review

Semester

Abstraction

If a function provides a consistent interface

- Client code can ignore implementation and think abstractly.
- Implementation details change with no impact on client code.

The same holds for classes and methods.

Inheritance can capture what classes share.

- Specifies what any future descendant class must do.
- Specifies what every descendant class has.
- Allows client code to call methods on an object without knowing which kind it is

Abstraction

Allows plug-out plug-in compatibility.

Allows polymorphism.

Think hard about the interfaces to your classes.

If you have to change the interface, all client code must change.

Much more on this in **csc207**, Software Design.

in review

Semester

2. Recursion

in review

Semester

Tracing recursive functions

An easy way to verify if your implementation exhibits the intended behaviour

As with any debugging, find bugs or corner cases that are not addressed

It can often be helpful to draw diagrams!

in review

Semester

Communicate via parameters & return

Each time we call a recursive method:

- Everything it needs should be sent through parameters
- Everything it must report back should come through what is returned

Don’t attempt to work around this protocol by using local variables.

- Each call has its own stack from with its own instance of the local variables.
- So nothing can accumulate in them across calls.

in review

Semester

Recursive helper methods

Sometimes, a method's API doesn't "have enough" to support the recursion.

A helper can have an additional parameter.

Example: The helper for Tree's __str__ method

Adds a parameter for indent

A helper can have an additional return value.

Example: The helper for Tree's average method

Returns a tuple with total and number

in review

Semester

3. Data Structures

in review

Semester

Common structures

Some new data structures are in your repertoire:

Linked list, tree, binary search tree

There are many more you will learn about later:

AVL tree, red-black tree, trie, heap, hash table, graph…

*More on this in **csc263**, Data Structures and Analysis.*

There are also algorithm structures:

Eg, greedy algorithms (a whole *category* of algorithms)

*More on this in **csc373**, Algorithm Design, Analysis & Complexity*

in review

Semester

4. Debugging

in review

Semester

Debugging

Main idea

Follow the logic of your code, step by step

Analyze if the behaviour of your code is as expected

Important skill

Especially for complex code

Gets you out of tricky spots

Better alternative to using print statements

in review

Semester

5. Testing

in review

Semester

Testing

Doctest examples are very important.

But we need to design thorough test cases.

For complex code, this is not easy.

Tools like pytest help us implement testing.

Property-based testing lets us easily check more cases, but only check for properties.

Assertions and proofs are valuable for very tricky algorithms.

in review

Semester

Testing

Important: test-as-you-go!

Write meaningful tests

This is a very important skill!

Consider corner cases

Your code should generally handle exceptional cases gracefully instead of crashing

Remember preconditions and representation invariants!

Re-test your code on every revision

Whenever you make substantial changes, re-test that everything that used to work still does

in review

Semester

6. Efficiency

in review

Semester

Run-time analysis

Some algorithms are much faster than others.

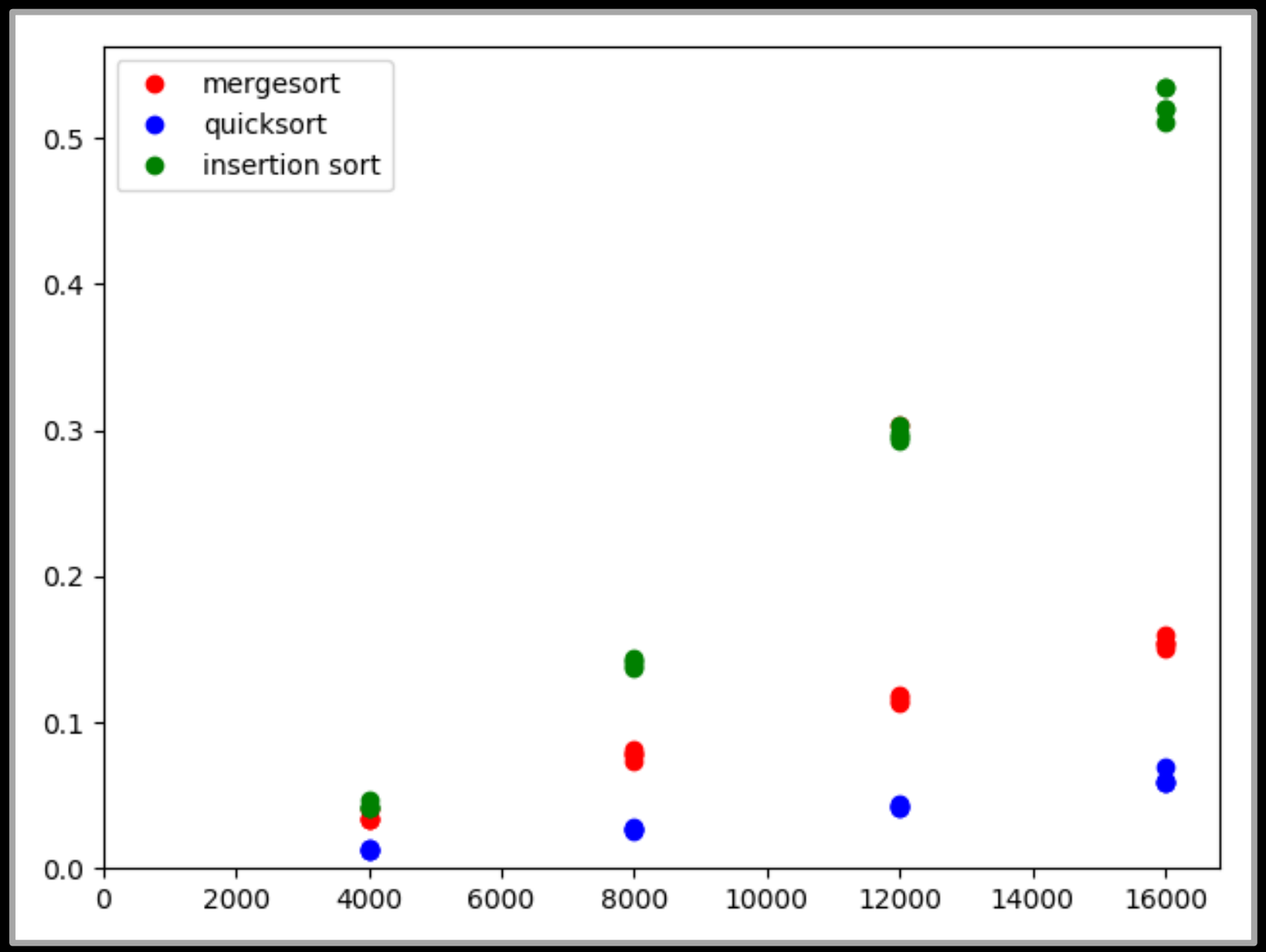
Big-oh helps us express such things.

*More on this in **csc236**, Introduction to the Theory of Computation.*

Semester in review

Lessons in efficiency

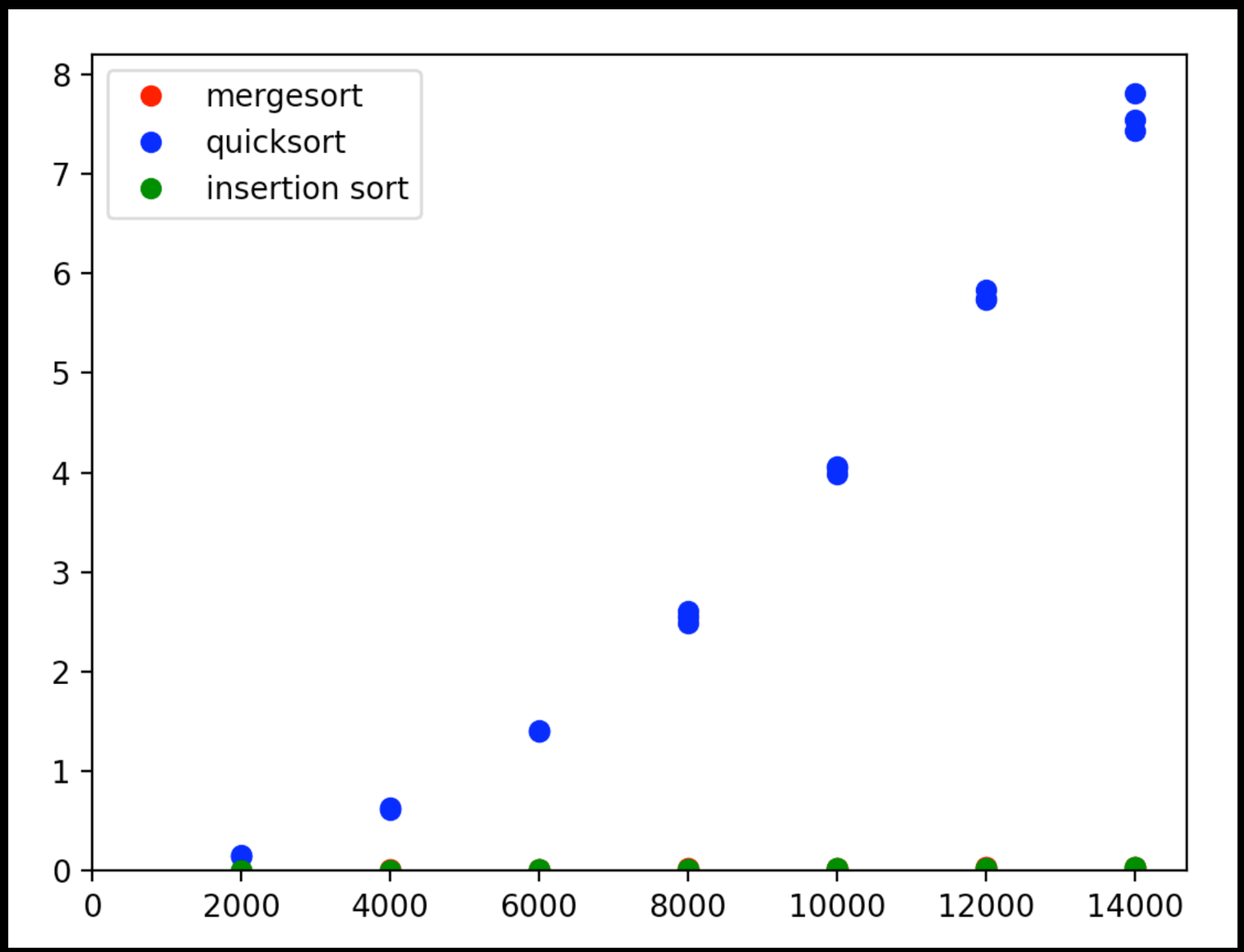
1. Big-Oh describes behaviour as input size grows



Semester in review

Lessons in efficiency

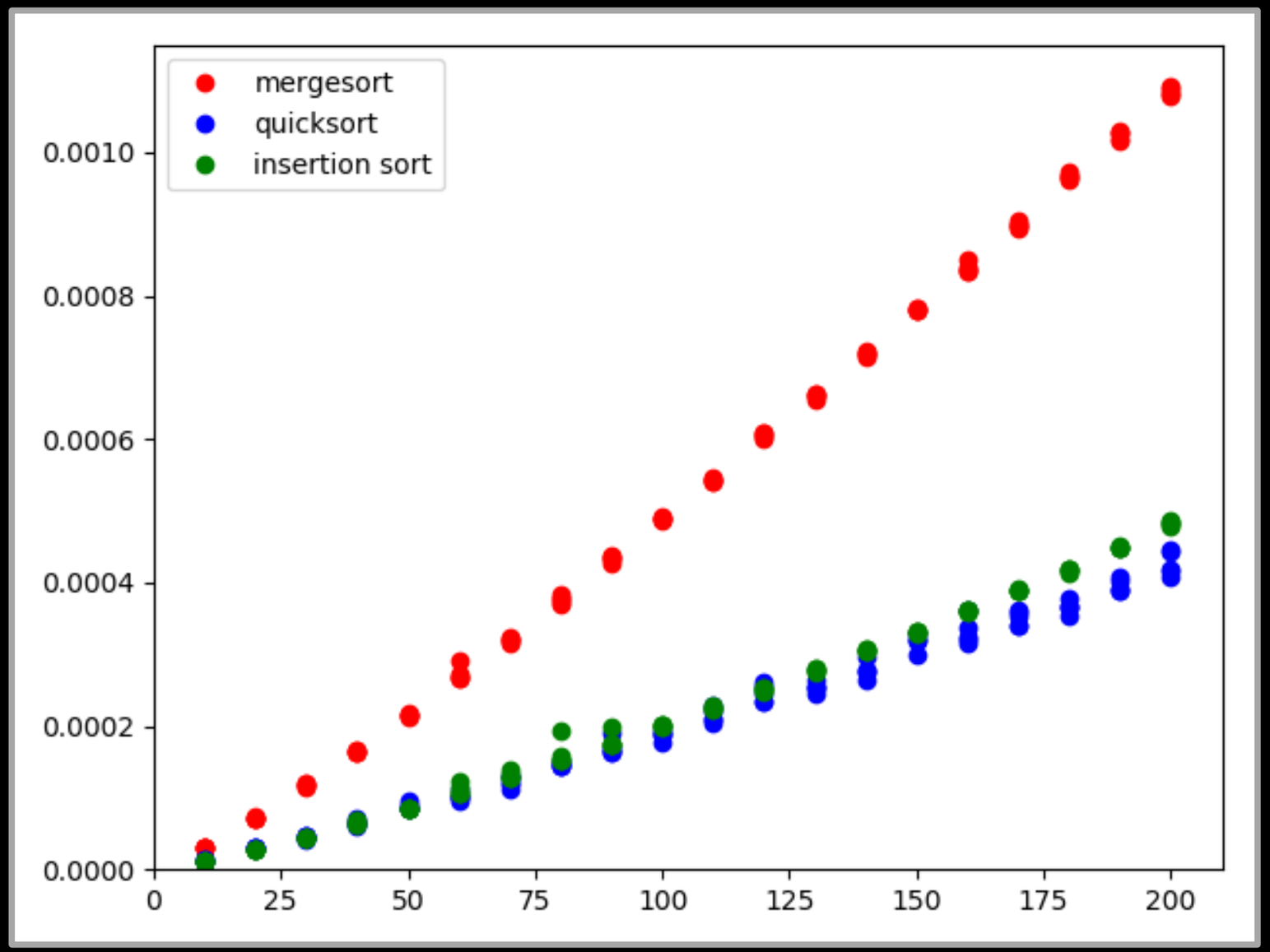
2. An algorithm can be “good on average” and “bad in the worst case”



Semester in review

Lessons in efficiency

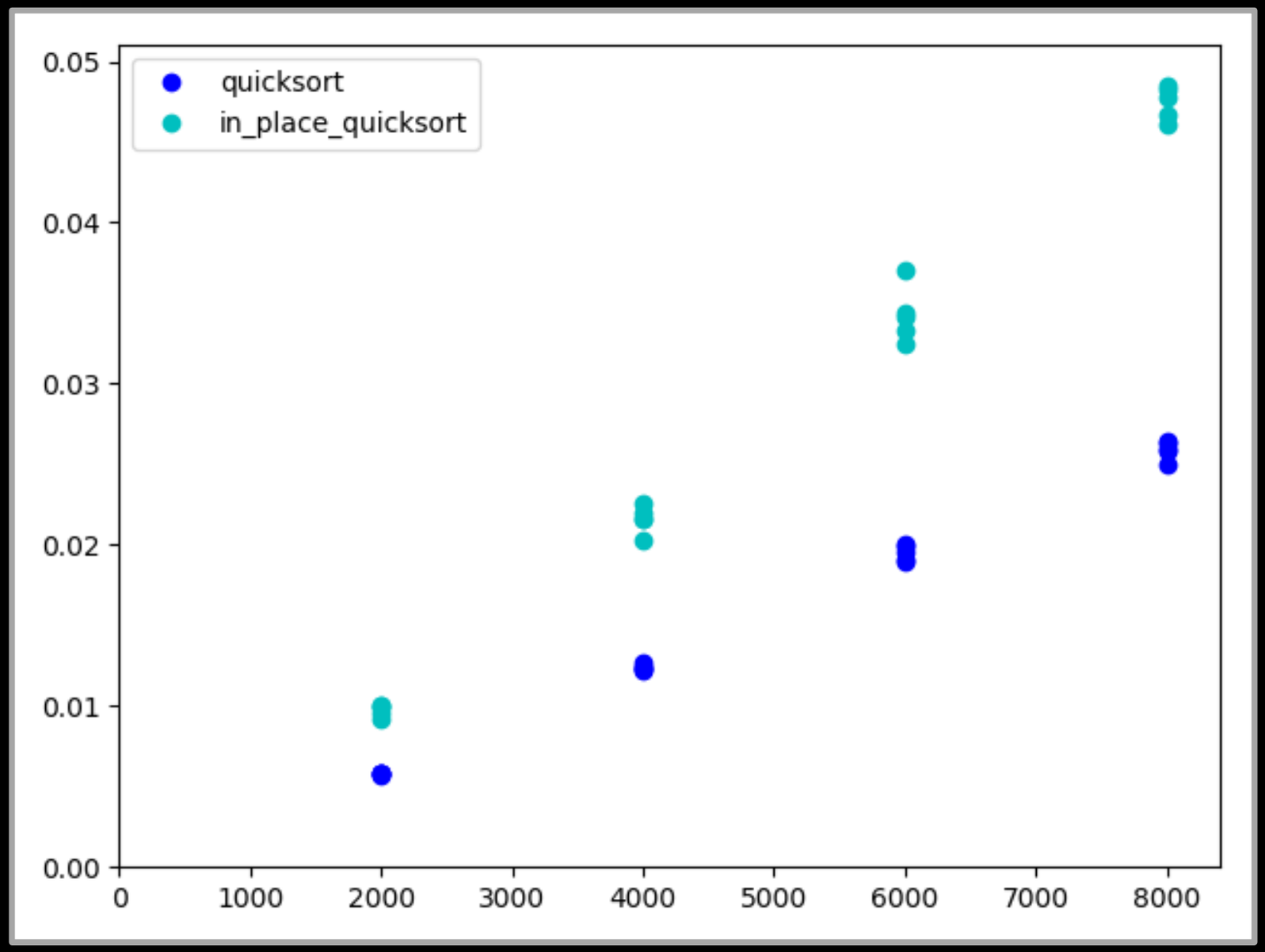
3. Big-Oh is *not* good at predicting behaviour on small inputs.



Semester in review

Lessons in efficiency

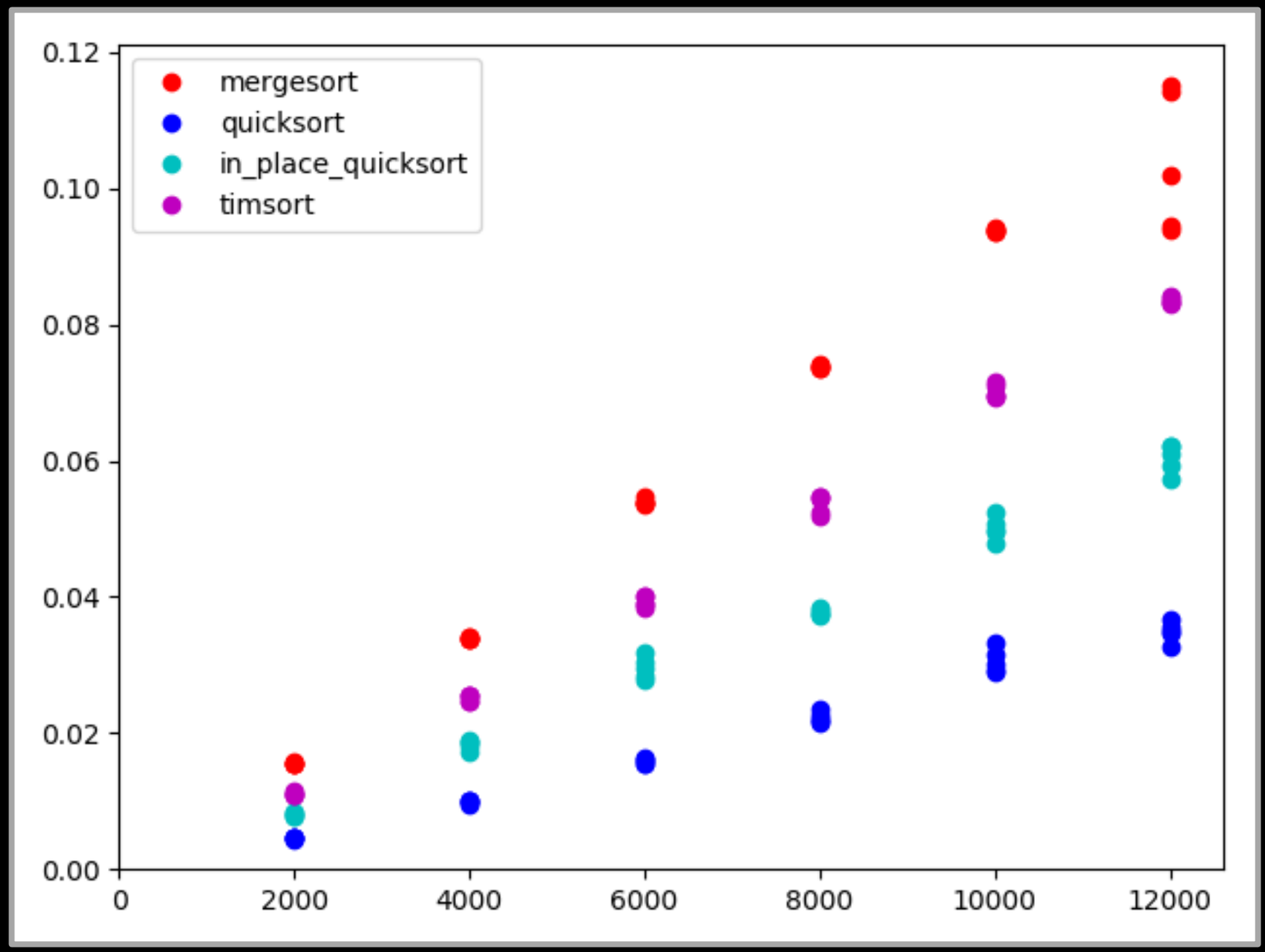
4. Saving space doesn't always mean saving time!



Semester in review

Lessons in efficiency

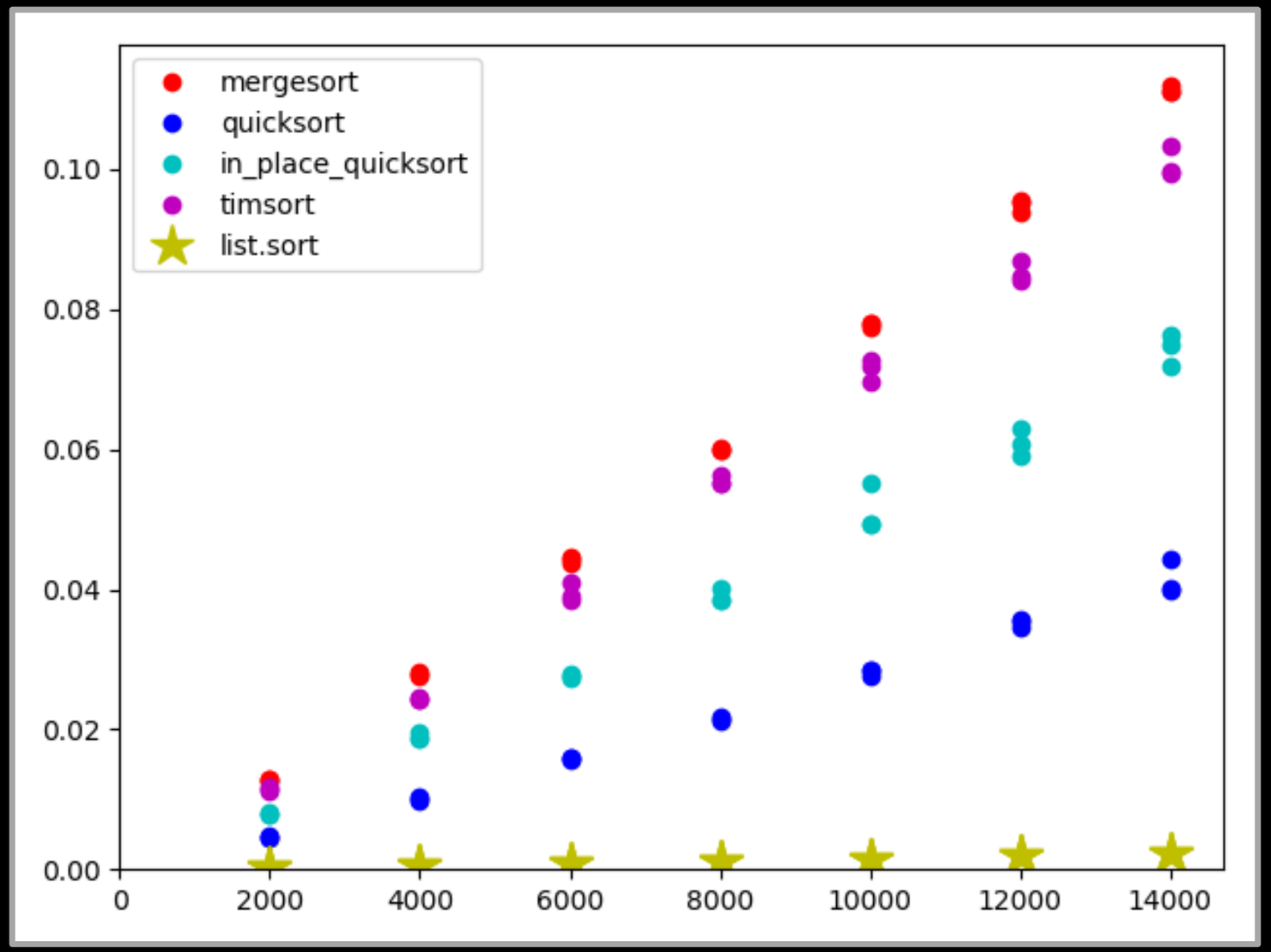
5. Hard work doesn't always mean saving time, either!



Semester in review

Lessons in efficiency

6. But sometimes hard work pays off.*



in review

Semester

7. Design by Contract

in review

Design by Contract

We saw a new way of thinking about code.

A function or method assumes certain pre-conditions

And guarantees certain post-conditions

A representation invariant is *always* true

This is called “design by contract.”

Design by contract is about pieces of code hooking together properly.

Semester

Code aesthetics is communication

"Programs must be written for people to read, and only incidentally for machines to execute."

Harold Abelson

Documentation is communication

"The most important single aspect of software development is to be clear about what you are trying to build."

Bjarne Stroustrup (Inventor of C++)

Software design is communication

> “There are two ways of constructing a software design: One way is to make it so simple that there are obviously no deficiencies, and the other way is to make it so complicated that there are no obvious deficiencies. The first method is far more difficult.”
>
> *Sir Charles Antony Richard Hoare (Inventor of Quicksort!)*

Maximizing your experience

Things to learn

Foundations outside CS

Calculus is important, for example, for

Machine learning, computer graphics, computational finance, …

Linear algebra is important, for example, for

Machine learning, computer vision, medical imaging, …

Statistics is important, for example, for

Machine learning, data mining, computational finance, user-experience (UX) studies, CS education research, …

Things to learn

Foundations outside CS

Embrace opportunities to strengthen skills in

- communication
- teamwork

Employers value these greatly

Embrace opportunities to explore other interests

No matter how “far” from CS, there are ways to connect.

And regardless, your interests matter

Things to do

Research opportunities

In the Department of Computer Science (DCS)
https://web.cs.toronto.edu/undergraduate/mentorship-research-work - research *(Learn more about our research groups here – https://web.cs.toronto.edu/research/areas)*

For course credit: csc299Y, csc399&, csc490H, csc494/5H

For pay: Undergraduate Summer Research Program

For extra-curricular credit: PRISM program – **P**reparation for **R**esearch through **I**mmersion, **S**kills, and **M**entorship

Things to do

Research seminars

DCS Distinguished Lecture Series

https://web.cs.toronto.edu/dls

DCS events more broady

https://web.cs.toronto.edu/events

Schwartz Reisman Institute for Technology and Society

https://srinstitute.utoronto.ca/events

Things to do

Other opportunities

DCS Alumni-Student Mentorship program https://web.cs.toronto.edu/mentorship

DCS Student Ambassadors
https://web.cs.toronto.edu/undergraduate/ambassadors-program

CSSU
https://www.cssu.ca/

Arts and Science Internship Program (ASIP)
https://www.artsci.utoronto.ca/current/academics/asip

Read the CS Undergrad Update

Enrol in the Quercus shell for CS undergrads
https://q.utoronto.ca/enroll/EMFCPB

My #1 advice:

Get to know people!

Some advice from Chirly

Now, put away your laptops and phones!

WE ARE WRITING A "MOCK" EXAM

Important note about the "mock" exam activity:

This is definitely NOT the same length or difficulty as the actual exam, but will give you great practice for solving exam-style questions in an exam-style setting!

Quizizz winners!

Congratulations to:
Ashley de Witte & Ross Magill!
(a tie!)

Runner-ups were:
Sanzhar Shamshiyev & John Chen (a tie!)
Chenrui Feng & Alice Zhang (another tie!)

Congratulations to:
Tianyi Pan!

Runner-ups were:
Javier Mencia & Benson Li

Congratulations to:
Welna!

Runner-ups were:
Jason Wang & Serena Zhang