CSC165 Week 12

Larry Yueli Zhang, November 25, 2014

Announcements

- Assignment 2 marks up
  - class average: **76%**
  - double-check on MarkUs
- Assignment 3 due, December 1, 10:00pm
- Extended office hours this week for A3
  - Thursday 4-6pm, in BA4262
  - Friday 4-6pm, in BA5287
  - Monday 4-6pm, in BA5287
- More extended office hours before final, TBA
- (Official) course evaluation !

Today's outline

- ➔ tips for Assignment 3
- ➔ countability
- ➔ Induction
- ➔ review for final exam

Tips for Assignment 3

Question 1

$$|x - y| = |y - x|$$

$$|x - y| < |x + y| \quad ?$$

definitions of limit (for Q3 and Q4)

$$\lim_{n \to \infty} f(n) = \infty$$

$$\forall c \in \mathbb{R}, \exists n' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n' \Rightarrow f(n) > c$$

no matter how large c is, f(n) can be larger than c

$$\lim_{n \to \infty} f(n) = 0$$

$$\forall c \in \mathbb{R}, \exists n' \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n' \Rightarrow f(n) < c$$

no matter how small c is, f(n) can be smaller than c

Question 3 & 4

using limit technique of calculus

does not imply

you must use L’Hopital’s rule

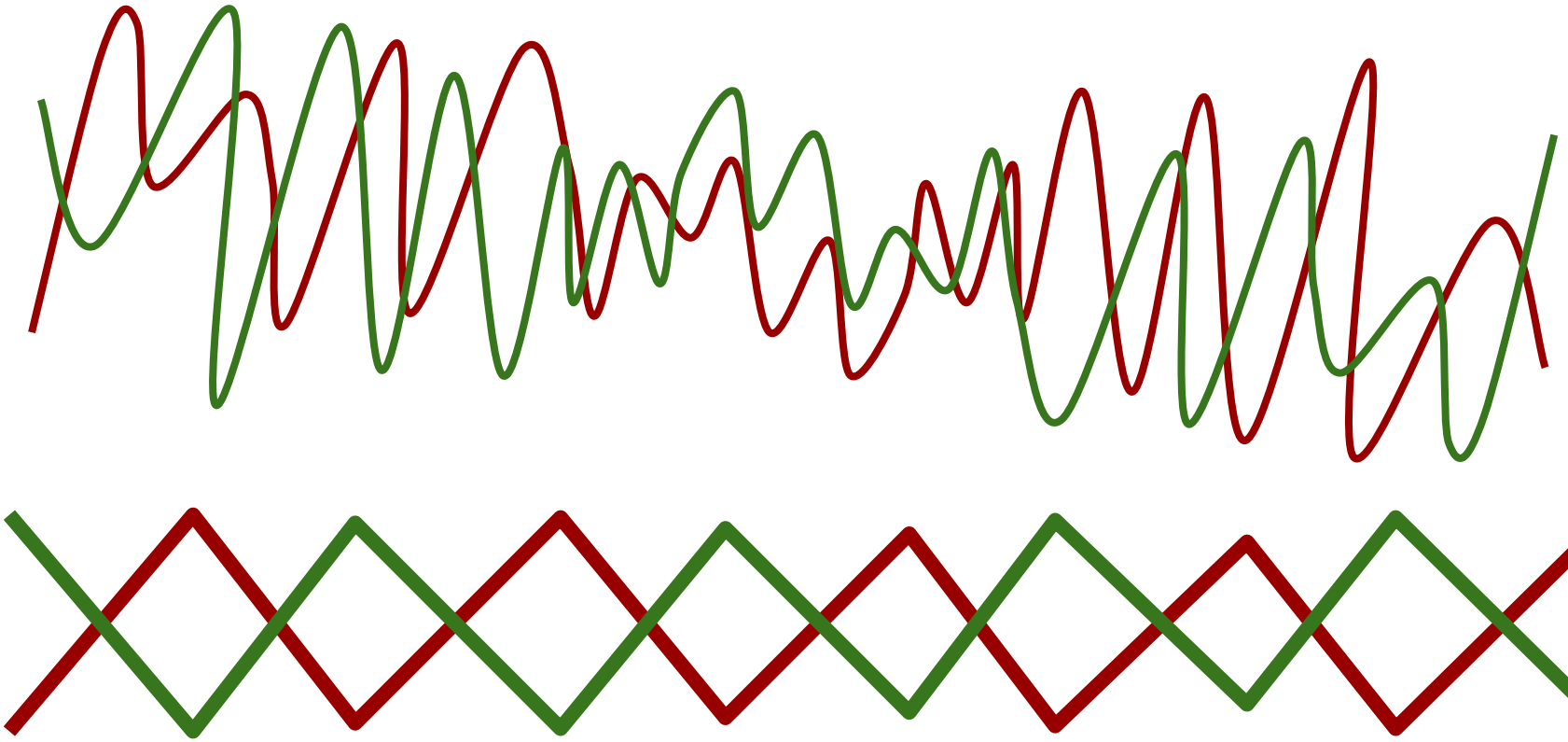
Question 5

True or False?

Given any two functions **f(n)** and **g(n)**, one must be upper-bounding the other.

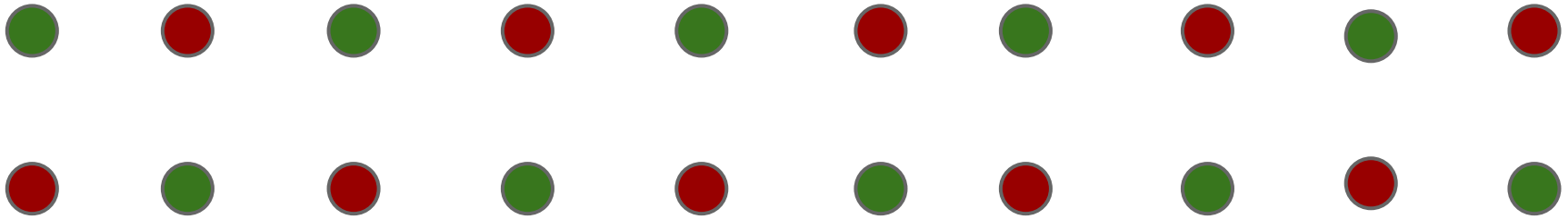
If you believe it's true, prove it, and good luck!

If you believe it's false, then disprove it by picking a **f(n)** and a **g(n)** such that nobody is bounding nobody.



In fact, we only care about natural number points ...

$$f : \mathbb{N} \mapsto \mathbb{R}^{\geq 0}$$



Question 6

The proof structure should emulate the proof in Course Notes Chapter 5.3 (Page 71), Claim 5.1, using reductions.

Reduce **halt** to **meaning_of_life**, i.e., implement **halt** using **meaning_of_life** as a helper function.

countability

compare the size of two sets

X = {1, 2, 3}

Y = {“one”, “two”, “three”}

how do their sizes compare?

$$|X| = |Y| = 3$$

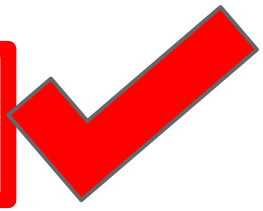
How about these two?

X = {natural numbers} # 0, 1, 2, 3, 4, 5, ...

Y = {even natural numbers} # 0, 2, 4, 6, ...

$|X| < |Y|$?

$|X| > |Y|$?

$|X| = |Y|$ 

an real-life example

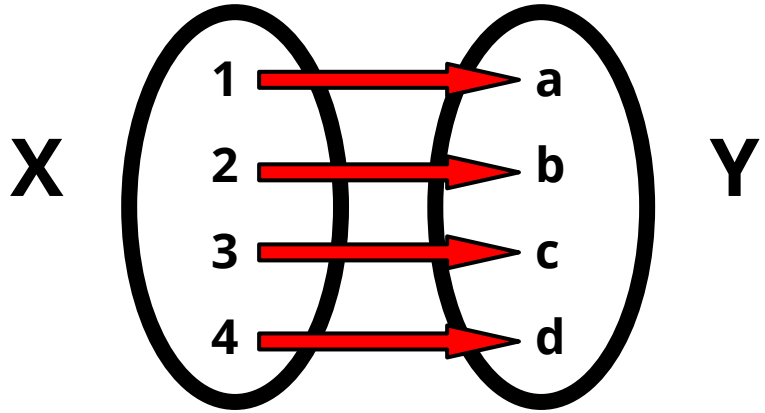
X: set of **coins**

Y: set of **coin tails**

they're of the **same size**, because

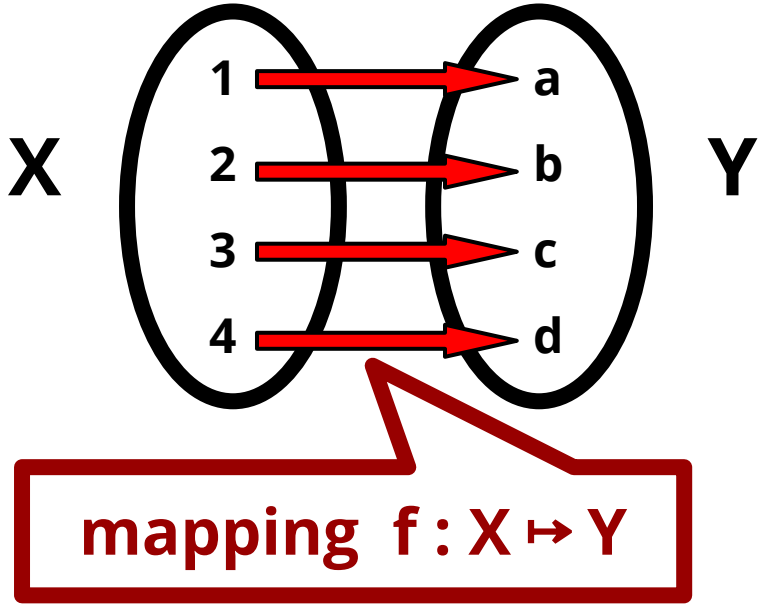
each coin has **one and only one** tail,

and each tail belongs to **one and only one** coin



If there is a mapping from X to Y like this, then we have |X| = |Y|

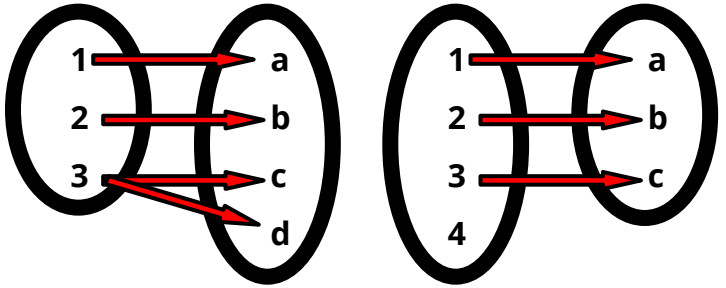
"like this", more precisely



What does $f : X \mapsto Y$ need to satisfy, mathematically?

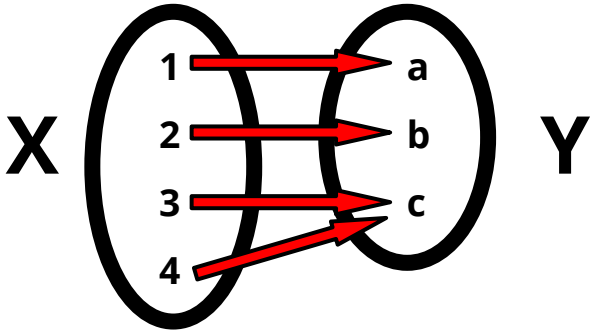
f is a **well defined function**, i.e., each **x** maps to a unique **y**.

So these don't happen:



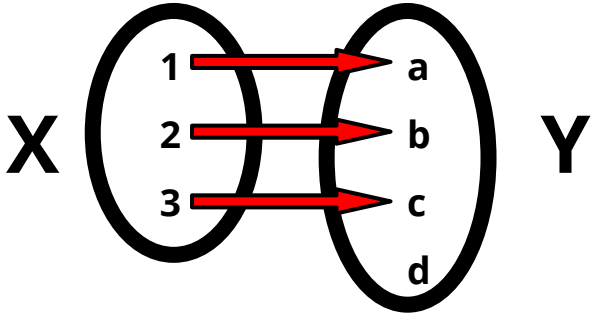
f is **"1-1"**, i.e., each **y** mapped to by at most one **x**.

So this doesn't happen



f is **"onto"**, i.e., every **y** has some **x** that maps to it.

So this doesn't happen:



put these together…

Given two sets **X** and **Y**, if there exists a **well defined** function $\mathbf{f : X \mapsto Y}$ that is **1-1** and **onto**, then $\mathbf{|X| = |Y|}$.

We can use this to compare the sizes of sets with **infinitely** many elements.

Key is: find an $\mathbf{f : X \mapsto Y}$

Whaaaaat?!

X = {natural numbers} # 0, 1, 2, 3, 4, 5, ...

Y = {even natural numbers} # 0, 2, 4, 6, ...

Prove: |X| = |Y|

$n \in \mathbb{N}$	$\longrightarrow$	$f(n)$
0	$\longrightarrow$	0
1	$\longrightarrow$	2
2	$\longrightarrow$	4
3	$\longrightarrow$	6
4	$\longrightarrow$	8
$\vdots$	$\vdots$	$\vdots$

Proof: ("find an **f : X ↦ Y**")

Let **f : X ↦ Y** be f(n) = 2n

then f is **well defined** # 2n is well defined

then f is **1-1** # if f(m) = f(n) = y, then m = n = y/2

then f is **onto** # all y in Y has x=y/2 in X mapping to it

then ∃ a well defined **f : X ↦ Y** that is 1-1 and onto

then |X| = |Y|

“countable”

when we **count**, we do: 0, 1, 2, 3, 4, …
that’s basically **enumerating natural numbers**

so, we say the set $\mathbb{N}$ is countable

and, any set A that satisfies $|A| \leq |\mathbb{N}|$ is countable

$\mathbb{Z}$ (integers) is countable

$\mathbb{Q}$ (rational numbers) is countable

$\mathbb{R}$ (real numbers) is uncountable **it’s “dense”**

countability and computability

How many Python functions could be written?

Answer: **infinitely many**

Better Answer: **countably infinitely many**

Why?

- ➔ the code of each Python function is basically a string of characters
- ➔ each (UTF-8) character corresponds a number between 0~255
- ➔ if we just concatenate all the numbers corresponding to all characters in the Python function, we get a unique natural number
- ➔ that is a 1-1 mapping: **Python functions ↦ N**

```
def hello(s):
    print(s)
    return
```

```
23762436234234
62347623462342
74262342342344
```

so, we can enumerate all Python functions like:

$f_0, f_1, f_2, f_3, \ldots$

so, we can create a **table of function halting behaviours**, i.e., the table of return values of $Halt(f_i, f_j)$

f_i	$H(f_i, f_0)$	$H(f_i, f_1)$	$H(f_i, f_2)$	$H(f_i, f_3)$	...
f_0	T	F	T	F	
f_1	T	T	F	F	
f_2	T	F	T	T	
f_3	F	T	F	F	
⋮					

this row describes the "**behaviour**" of f_3

all Python functions' behaviours are **enumerated** in this table

Now, construct a function f_x with "weird" behaviour

f_i	$H(f_i, f_0)$	$H(f_i, f_1)$	$H(f_i, f_2)$	$H(f_i, f_3)$
f_0	T	F	T	F
f_1	T	T	F	F
f_2	T	F	T	T
f_3	F	T	F	F

Take the diagonal and flip T/F $H(f_x, f_i) = \neg H(f_i, f_i) \quad i = 0, 1, 2, \ldots$

f_x	F	F	F	T

This row should be the same as some row in the table, since f_x is just one of the Python functions being enumerated.

Which row? **NO ROW!**

$$\forall i \in \mathbb{N}, \mathrm{Entry}(x, i) = \neg \mathrm{Entry}(i, i)$$

at least one mismatch in every row *i*

f_x does NOT exist in the table!!

f_x does NOT exist in the table!!

$$f_x : H(f_x, f_i) = \neg H(f_i, f_i) \quad i = 0, 1, 2, \ldots$$

i.e., $f_x(f_i)$ halts if $H(f_i, f_i)$ is False

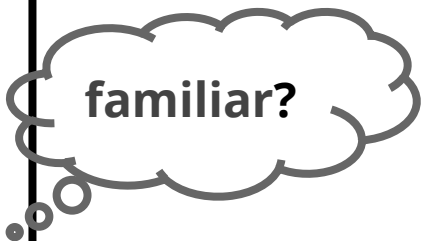
$f_x(f_i)$ does not halt if $H(f_i, f_i)$ is True

f_x cannot be programmed (the table is for codable funcs)

if it could be programmed, the code would look like...

```
def fx(fi):

    while halt(fi, fi):
        pass
    return 42
```




```
def clever(f):

    while halt(f, f):
        pass
    return 42
```

fx() cannot be programmed because halt() cannot be programmed!!

summary

- ➔ The argument we made has a name, it's called "**diagonalization**".
- ➔ Given any list of countably many of functions, we can always construct an f_x that is **outside** that list.
- ➔ that means the set of all possible functions is **uncountable**.
- ➔ Since we can only program **countably many** Python functions
- ➔ there are **uncountably many** functions that we **cannot program** in Python, or in any computer language

We barely scratched the surface of computability theory.

Learn more in CSC463 and CSC438.

one more thing …

an awesome proof technique that we didn't learn

Induction !

Mathematical induction is a proof technique typically used to establish a given statement for all natural numbers.

$$\forall n \in \mathbb{N}, P(n)$$

example

Name the dominoes $d_0, d_1, d_2, d_3, \ldots$

Prove: d_0 falls $\Rightarrow$ $\forall i \in \mathbb{N}, d_i$ falls

Proof: $d_0 \text{ falls} \Rightarrow \forall i \in \mathbb{N}, d_i \text{ falls}$

Assume d_0 falls

then d_1 falls **# d0 will hit d1**

then d_2 falls **# d1 will hit d2**

then d_3 falls **# d2 will hit d3**

then d_4 falls **# d3 will hit d4**

...

keep doing this until all natural numbers have been enumerated (which means never)

There must be a better way!

Principle of Simple Induction (PSI)

$$P(0) \wedge (\forall n \in \mathbb{N}, P(n) \Rightarrow P(n+1))$$

$$\forall n \in \mathbb{N}, P(n)$$

Base case

the first domino falls

Inductive step

every domino hits the next one

all dominoes fall !

$$d_0 \text{ falls } \Rightarrow \forall i \in \mathbb{N}, d_i \text{ falls}$$

Proof by induction:

Assume d_0 falls

Base case: d_0 falls **# by assumption**

Inductive step:

assume $n \in \mathbb{N}, d_n$ falls

then d_{n+1} falls **# dn will hit dn+1**

then $\forall n \in \mathbb{N}, d_n \text{ falls} \Rightarrow d_{n+1}$ falls

then $\forall n \in \mathbb{N}, d_n$ falls **# PSI**

then $d_0 \text{ falls} \Rightarrow \forall n \in \mathbb{N}, d_n$ falls

exercise for home

$$\text{Prove: } \forall n \in \mathbb{N}, \sum_{i=0}^{n} i = \frac{n(n+1)}{2}$$

use induction

summary

➔ Again, we barely scratched the surface of induction. You will learn more in CSC236 or CSC240.

➔ Countability and induction will **NOT** be tested in the final exam

Final Exam Review

PLEASE HAND IN

PLEASE HAND IN

UNIVERSITY OF TORONTO
Faculty of Arts and Science

December 2014 Examinations

CSC165H1F

Duration — 3 hours

Aids allowed: 8.5"x11" aid sheet

Student Number: |_|_|_|_|_|_|_|_|_|_|

Last Name: _____

First Name: _____

Do **not** *turn this page until you have received the signal to start.*
(In the meantime, please fill out the identification section above, and read the instructions below.)

This exam consists of 9 questions on 18 pages (including this one). *When you receive the signal to start, please make sure that your copy of the exam is complete.*

Please answer questions in the space provided. You will earn 20% for any question you leave blank or write "I cannot answer this question," on. You will earn substantial part marks for writing down the outline of a solution and indicating which steps are missing. You must achieve at least 40% of the marks on this exam in order to pass the course.

Write your student number at the bottom of pages 2-18 of this exam.

# 1:	_____/ 8
# 2:	_____/ 8
# 3:	_____/10
# 4:	_____/10
# 5:	_____/10
# 6:	_____/10
# 7:	_____/ 8
# 8:	_____/ 6
# 9:	_____/10

TOTAL: _____/80

Good Luck!

(rough) composition of the 9 questions

- ➔ 4~5 questions: write proofs
  - Chapter-3-style proofs
  - big-Oh/Omega proofs
- ➔ 4~5 questions: other things
  - logics stuff
  - algorithm analysis
  - halting
  - …

topics necessary to be on top of

logics stuff

- quantifiers: ($\forall$, $\exists$). Deal with them when they are mixed.
- conjunction and disjunction ($\wedge$ and $\vee$)
- implications (=>) and bi-implications (<=>), know their relation with $\wedge$ and $\vee$
- negations (¬), know how to negate a statement of an expression
- manipulation rules
  - Course Note: Chapter 2.17 (all of them!)
  - be able to handle very complex expressions and statements, really know how they work, instead of relying on aid sheet
- Venn diagrams
  - know how to draw, and put marks (X, O) on them
  - be able to handle Venn diagram with more than 2 circles
- Truth table: Can read one, can draw one.

Proofs (Chapter 3 style)

- ➔ Proof techniques
  - ◆ direct proofs, contrapositive, contradiction
  - ◆ disproofs (negate and prove the negation)
- ➔ Proof structures
  - ◆ Keywords: “assume”, “pick”, “let be”, “then” -- should match the quantifier (ordered) in the statement to prove.
  - ◆ indentations
  - ◆ comment (!): critical for important steps, prefer writing more to writing less
- ➔ Types of questions
  - ◆ related to what we have practiced in the course

Proofs (big-Oh and big-Omega)

- for polynomials
  - use the standard procedure with a chain of overestimate and underestimate (and maybe the magical breakpoint B = 1)
- for non-polynomials
  - use calculus (possibly L'Hopital) to prove limit, translate to definition of limit, then relate it to the definition of big-Oh
  - review calculus: know basic rules of taking derivatives
  - don't use magical B = 1 here
- for general statements
  - use the c and B from the given big-Oh statement to construct the c' and B' that you need
  - don't use magical B = 1 here

Proofs (big-Oh and big-Omega) cont.

- Prove and disprove
  - in O, not in O, in Ω, not in Ω, know how to do them all
- Definitions of big-Oh, big-Omega
  - really understand what these definitions mean
  - also understand the negation of these definitions
- Definition of limit in calculus
  - definition of limit going to infinity
  - definition of limit going to zero
  - really understand the meaning of them
  - then put them on aid sheet

algorithm analysis

- ➔ given a piece of Python code, analyse running time
- ➔ we typically ask for an **upper-bound** on the **worst-case** complexity
- ➔ we typically want the upper-bound to be in **exact form**, not in big-Oh form
- ➔ you may be asked to count the number of executions of all lines, or some particular line, read the question carefully
- ➔ your upper-bound should be as **tight** as you can, tighter bounds will be given higher marks.
  - ◆ e.g., $3n$ is better than $2n^2$

halting

- ➔ halting problem and reduction
  - ◆ understand what you did in Assignment 3 Question 6
- ➔ countability, diagonalization, induction
  - ◆ not required in final

come to office hours!

materials to review

- ➔ understand assignment questions and (correct) solutions, inside-out (also midterm questions)
- ➔ go over tutorial exercises, understand them
- ➔ go over lecture notes, understand them
- ➔ read Course Notes for more detailed explanation
- ➔ work on practice questions
  - ◆ http://www.cdf.toronto.edu/~heap/165/F14/A65_Practice_Questions_Solns.pdf
- ➔ old exam repository
  - ◆ https://exams-library-utoronto-ca.myaccess.library.utoronto.ca/simple-search?query=csc165*&submit=%EF%BF%BD%EF%8F%A5%E9%8A%B5%EF%BF%BD

exam tactics

- ➔ be cool
- ➔ write your names and student number
- ➔ expect to spend ~20 minutes per question
- ➔ read questions carefully, when in doubt, ask invigilator
- ➔ get an overall sense of the difficulties of the questions, hard ones don't necessarily come last
- ➔ if stuck in one question for too long, move on
- ➔ you get 20% for leaving it blank
- ➔ you get ~50% for writing a **valid** proof structure
  - ◆ indicate which steps you cannot fill in, and what you are assuming without proof.
- ➔ spaces on the exam paper are more than enough, so if you are writing too much, something might be wrong.

➔ Date & location:

- December 10, 7:00-10:00pm
- EX100 (A-SE), EX200 (SH-Z)
- 8.5”x11” aid sheet, double sided
- bring your T-Card

Good luck!