CSC263 Fall 2015

Data Structures and Analysis

Toniann Pitassi

Email: toni@cs.toronto.edu

BIG THANKS TO: Larry Zhang
(for his amazing slides)

The teaching team

Professor:

➔ Toniann Pitassi

Tas:

Daniel Hidru
Lalla Mouatadid
Majid Komeili
Ladislav Rampasek
Dhyey Sejpal
Noah Fleming
Pan Zhang
Robert Robere

Outline for today

Why take CSC263?

What is in CSC263?

How to do well in CSC263?

Why take CSC263?

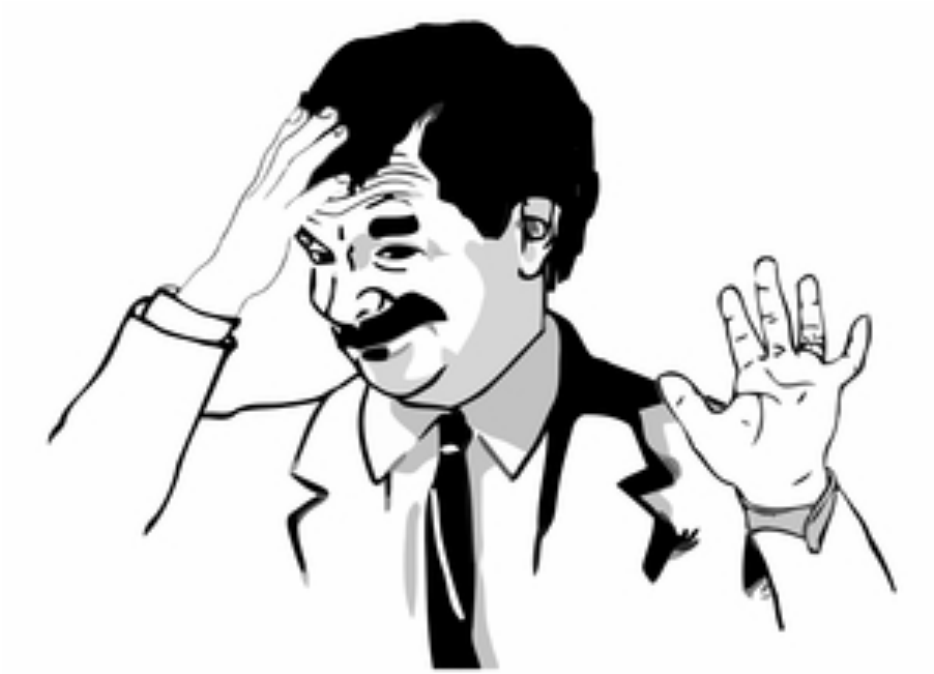
To land a job!

Scenario: The interview

Interviewer: You are given a set of courses, like *CSC165*, *STA247, CSC263, CSC373,* where each course stores a list of prerequisites. Devise an algorithm that returns a valid ordering of taking these courses.

You: (think for 1 minute…) Here is my algorithm:

- ➔ For a valid ordering to exist, there must be a course **X** that has no prerequisite.
- ➔ I choose **X** first, remove **X** from the set of courses, then remove **X** from all other courses' prerequisite list.
- ➔ Find the next course in the set that has no prerequisite.
- ➔ Repeat this until all courses are removed from the set.

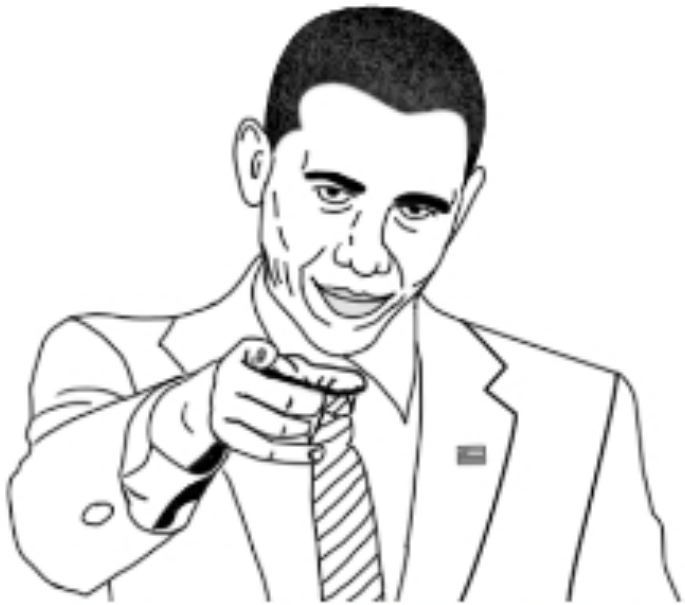


SHUT UP AND GO HOME!

Scenario: The interview, Take 2

Interviewer: You are given a set of courses, like *CSC165*, *STA247, CSC263, CSC373,* where each course stores a list of prerequisites. Devise an algorithm that produces a valid ordering of taking these courses.

You: This is a **topological sort** problem which can be solved using **DFS**.


YOU GOT IT

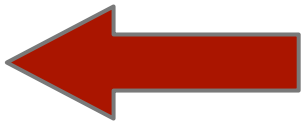
Data structures are smart ways of organizing data, based on which we can develop efficient algorithms easily, in ways that people who don't take CSC263 can't even imagine.

Design algorithms **like a pro**.

“Bad programmers worry about the code. Good programmers worry about the data structures and their relationships.”

-- Linus Torvalds

What's in CSC263?

(1) **Data structures** 

and

(2) **Analysis**

What data structures

- ➔Heaps
- ➔Binary search trees
- ➔Balanced search trees
- ➔Hash tables
- ➔Disjoint set forest
- ➔Graphs (matrix, lists)
- ➔...

What data structures

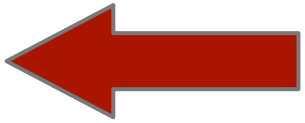
Ways of storing and organizing data to facilitate access and modifications.

We learn the strength and limitations of each data structure, so that we know which one to use.

(1) **Data structures**

and

(2) **Analysis**



What analyses

➔Worst-case analysis

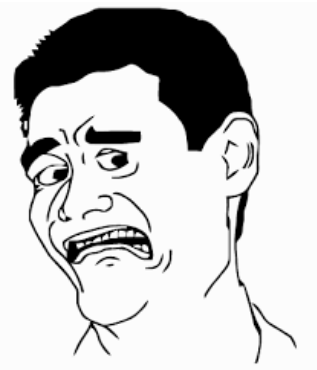
➔Average-case analysis

➔Amortized analysis

➔Expected worst-case analysis for randomized algorithms

➔...

Math and proofs



Secret Truth

Data structures are fun to learn,

but **analyses** are more important.

Cooking

A data structure is like a dish.

The analysis is to know the effect of each ingredient.

Analyses enable you to invent your own dish.

Background (Required)

➔Theory of computation

- Inductions
- Recursive functions, Master Theorem
- Asymptotic notations, “big-Oh”

➔Probability theory

- Probabilities and counting
- Random variables
- Distribution
- Expectations

How to do well in CSC263?

First of all...

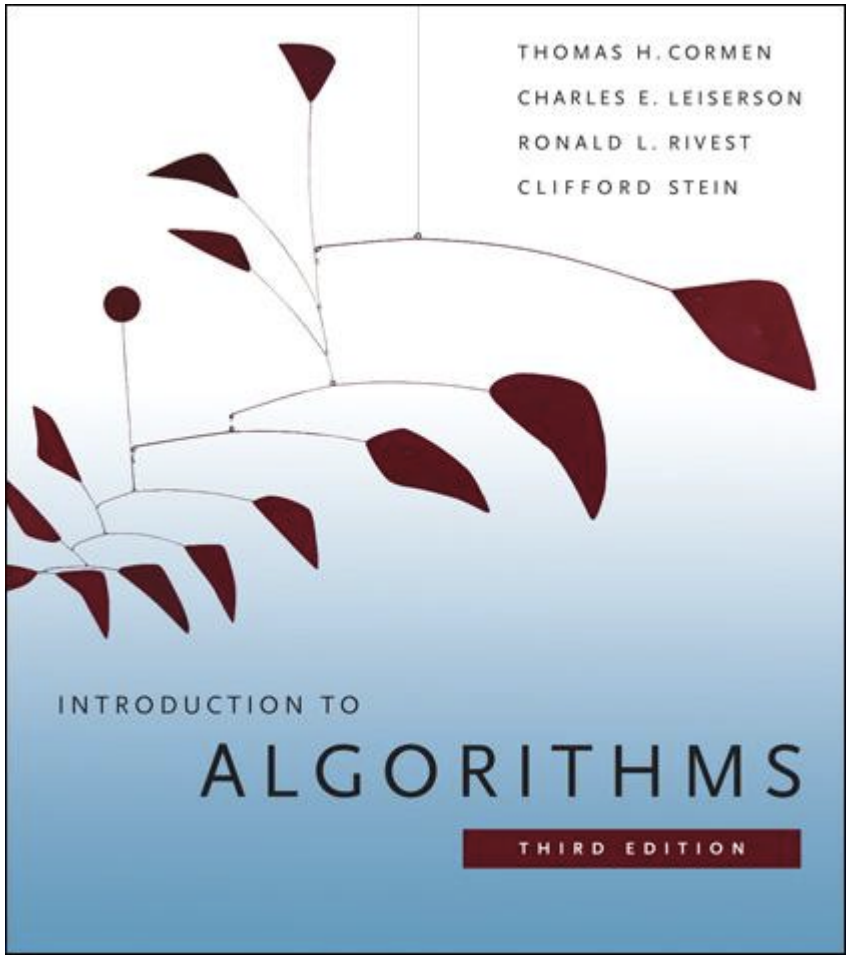
Be interested.

Course Web Page

www.cs.toronto.edu/~toni/Courses/263-2015

Lecture notes / slides (and everything else) will be posted at the course web page.

Textbook: “CLRS”



Second and third editions are both fine.

Available online at UofT library

Reading for each week is on course info sheet.

Lectures

Exception: No tutorial on Oct 23, Dec 2

➔Wed 10-12 in RW 117

➔Wed 2-4 in WI 1016

➔Learn stuff

Tutorials

Starting this week!

Practices for homeworks and exams.

Tutorials are as important as lectures.

A tip for lectures and tutorials...

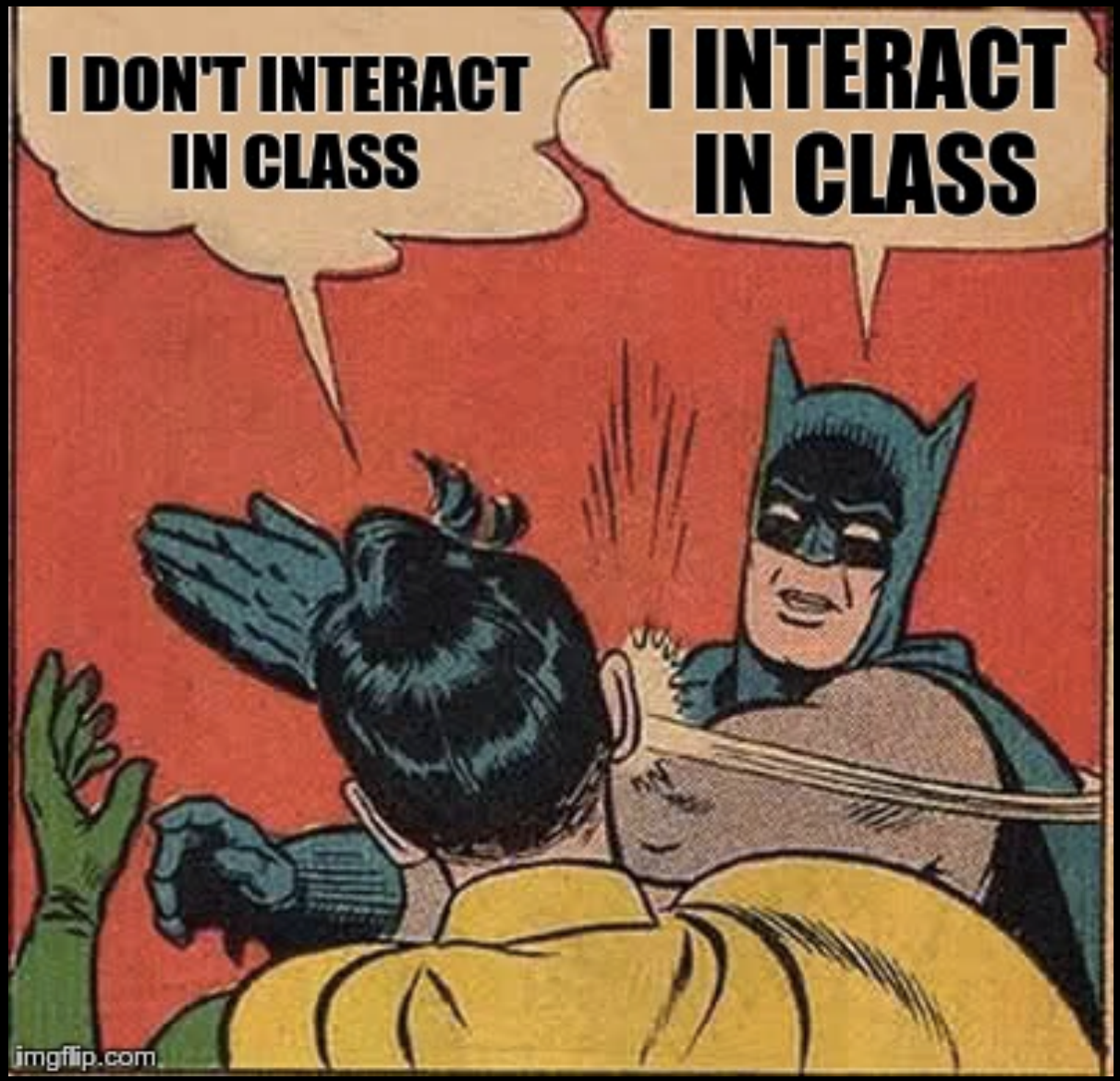
Get involved in classroom **interactions**

➔answering a question

➔making a guess / bet / vote

➔back-of-the envelope calculations

Emotional involvement makes the brain remember better!


I DON'T INTERACT IN CLASS
I INTERACT IN CLASS
imgflip.com

Course Forum

piazza.com/utoronto.ca/fall2015/csc263h1

Use UToronto email to sign-up.

For discussions among students, instructors will be there answering questions, too. Very helpful.

Communicate intelligently!

Don’t discuss homework solutions before due dates.

Office Hours

Wednesdays 12:15pm – 1:45pm
Location: SF2305A

Fridays 2-3pm, Mondays 6-7pm
Location Pratt 266

- They are very helpful and NOT scary at all!
- Statistically, students who go to office hours get higher grades.
- Additional office hours Friday and Mondays before assignments are due (CHECK WEBPAGE!)

Marking Scheme

➔5 assignments:	40% = 8% x 5
➔1 midterm:	20%
➔1 final exam:	40%
➔**TOTAL**	**100%**

Must get at least 40% on the final to pass.

Assignments (5 of them)

- ➔ You may work **individually** **or in** **groups of up to 4 students**
- ➔ Submissions need to be **typed** into a **PDF** file and submitted to **MarkUS** (link at course web page).
- ➔ Due dates are Tuesdays 5:59pm
- ➔ Late assignments: You will receive 4 tokens, each worth 6 hours of lateness. If your group submits late, everyone in the group will use up tokens.
- ➔ **Collaborate intelligently!** (so that everyone can pass exam)

about typing

LaTeX is beautiful and strongly recommended

➔ We will post our TeX source files, which you can use as templates.

➔ Many tutorials online. e.g.,

http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/

➔ Handy tools that do everything in the browser

- www.sharelatex.com
- www.writelatex.com

Problem Set 1 is out today!

Due Tuesday (Sept 29)

Exams

Midterm:
Thursday, October 22, 8-9pm.

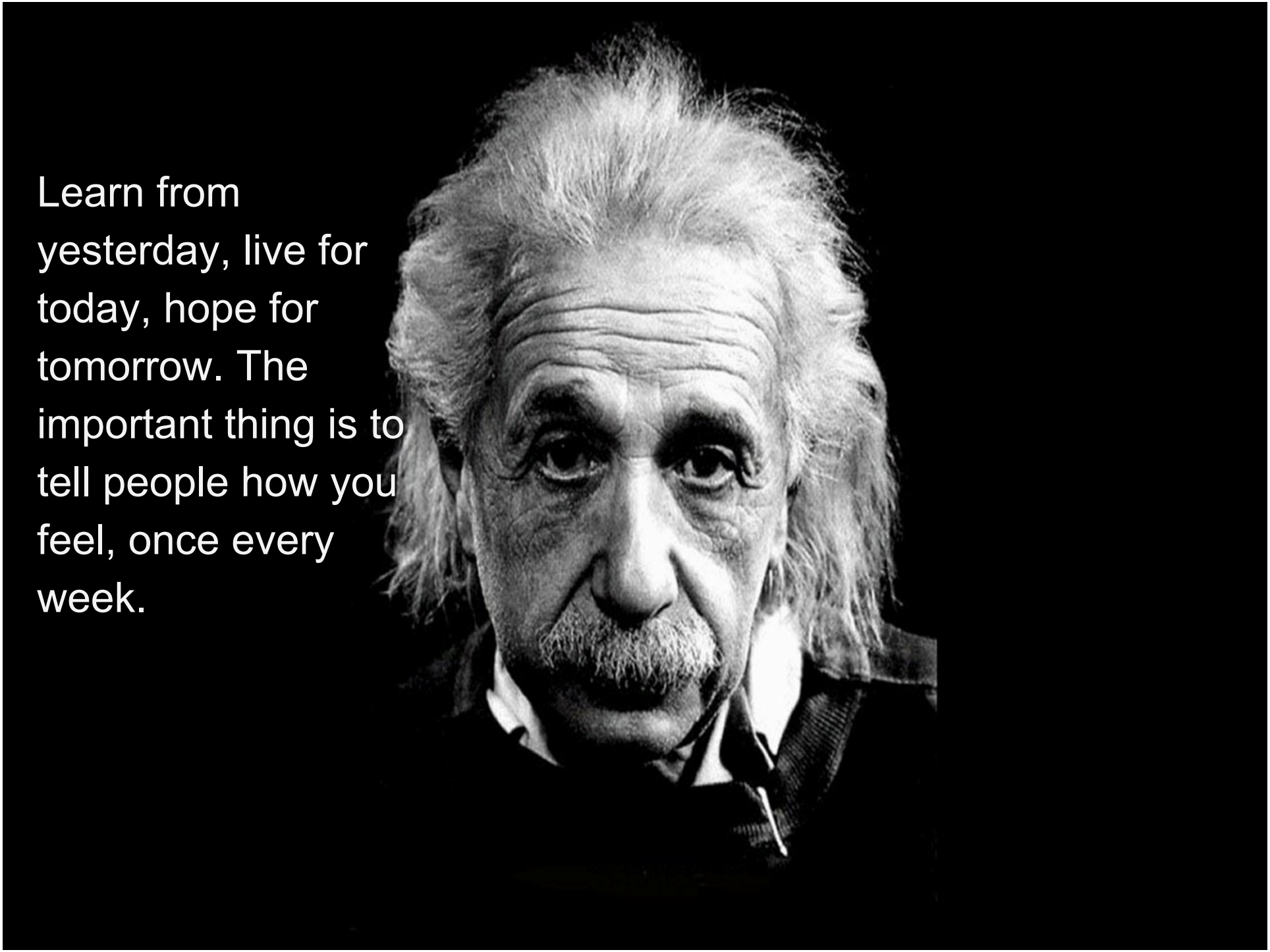
Fill out the form (see course webpage, under Tests) if you have a time conflict, by Sept 25.

Final exam:
Date to be announced.

Feedback

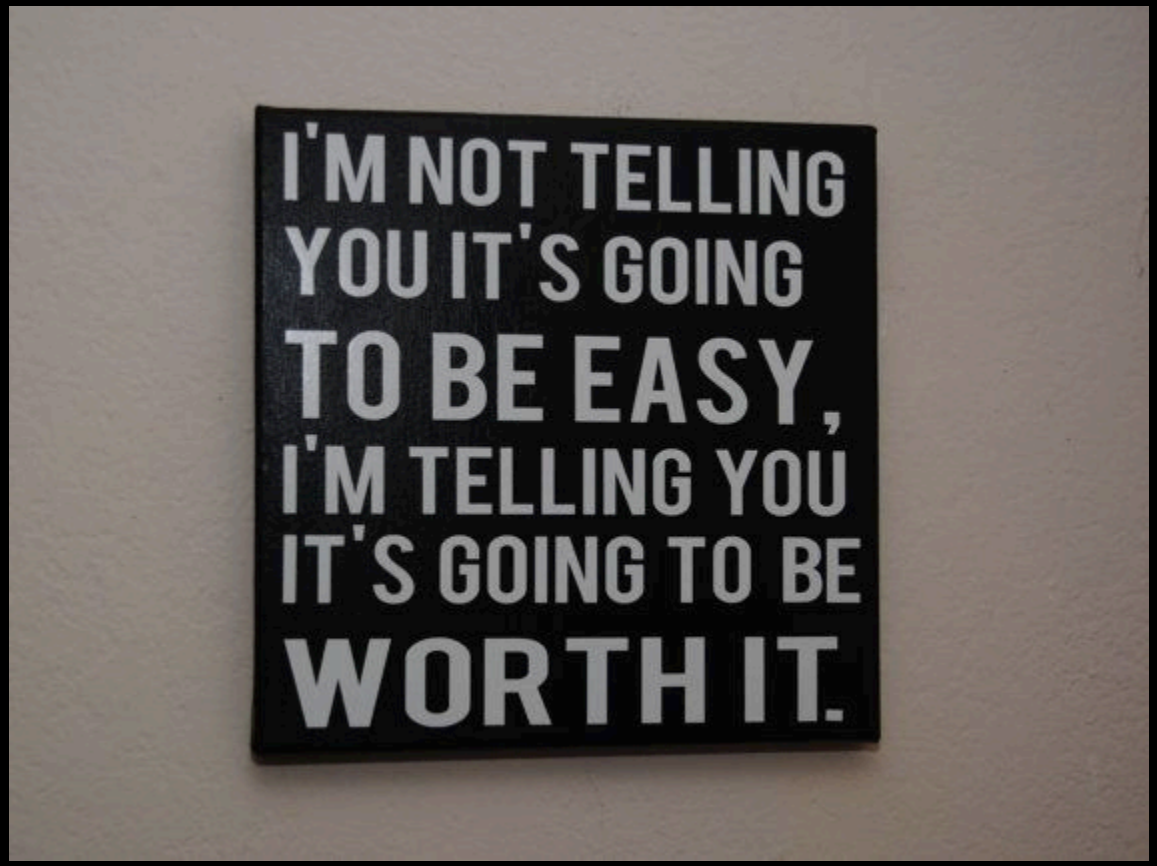
Feedback at any time is encouraged and appreciated.

- ➔ Things you like to have more of.
- ➔ Things you want to have less of.
- ➔ A topic that you feel difficult to understand.
- ➔ Anything else related to the course.


Learn from yesterday, live for today, hope for tomorrow. The important thing is to tell people how you feel, once every week.

Checklist: How to do well

- ➔ Be interested.
- ➔ Check course web page regularly.
- ➔ Go to lectures.
- ➔ Go to tutorials.
- ➔ Read textbook and notes.
- ➔ Discuss on Piazza.
- ➔ Feel free to go to office hours.
- ➔ Work on homeworks, and submit on time.
- ➔ Do well in exams.


I'M NOT TELLING
YOU IT'S GOING
TO BE EASY,
I'M TELLING YOU
IT'S GOING TO BE
WORTH IT.

Abstract Data Types (ADT) and Data Structures

Two related but different concepts

In short,

➔ADT describes **what** (the **interface**)

- what data is stored
- what operation is supported

➔Data structure describes **how** (the **implementation**)

- how the data is store
- how to perform the operations

Real-life example

ADT

➔ It stores ice cream.

➔ Supported operations:

- ◆start getting ice cream
- ◆stop getting ice cream

Data structures

➔ How ice cream is stored.

➔ How are the start and stop operations implemented.

➔ It's the inside of the machine

A CS example

Stack is an **ADT**

➔ It stores a list of elements

➔ supports PUSH(S, v), POP(S), IS_EMPTY(S)

Data structures that can used to implement **Stack**

➔ Linked list

- PUSH: insert at head of the list
- POP: remove at head of the list (if not empty)
- IS_EMPTY: return “head == None”

➔ Array with a counter (for size of stack) also works

In CSC263, we will learn many ADTs and many data structures for implementing these ADTs.

Review:
Algorithm Complexity

Complexity

Amount of **resource required** by an algorithm, measured as a function of the **input size.**

Time Complexity

➔ Number of **steps** ("running time") executed by an algorithm

*In CSC263 we will be dealing with **time** complexity most of the time.*

Space Complexity

➔ Number of units of space required by an algorithm

- e.g., number of elements in a list
- number of nodes in a tree / graph

Example: search a linked list

```
SearchFortyTwo(L):
1.    z = L.head
2.    while z != None and z.key != 42:
3.       z = z.next
4.    return z
```

Let input **L = 41 -> 51 -> 12 -> 42 -> 20 -> 88**

How many times Line #2 will be executed?

4

Now let **L = 41 -> 51 -> 12 -> 24 -> 20 -> 88**

How many times Line #2 will be run?

7 (the last one is z == None)

Note

Running time can be measure by counting the number of times **all lines** are executed, or the number of times **some lines** (such as Line #2 in LinkedSearch) are executed.

It's up to the problem, or what the question asks.

best-case
worst-case
average-case

Worst-case running time

- ➔ $t_A(x)$: the running time of algorithm A with input **x**
- ➔ If it is clear what **A** is, we can simply write **t(x)**
- ➔ The **worst-case running time** **T(n)** is defined as

$$T(n) = \max\{\, t(x) : x \text{ is an input of size } n \,\}$$

"worst-case" is the case with the **longest** running time.

Slow is bad!

Best-case running time

Similarly to worst-case, **best-case** is the case with the **shortest** running time.

$$\min\{\, t(x) : x \text{ is an input of size } n \,\}$$

Best case is not very interesting, and is rarely studied.

Because we should prepare for the worst, not the best!

Example: Search a linked list, again

```
SearchFortyTwo(L):
1.    z = L.head
2.    while z != None and z.key != 42:
3.       z = z.next
4.    return z
```

What is the worst-case running time among all possible **L** with length **n** , i.e., **T(n)**?

T(n) = n + 1

the case where 42 is not in **L** (compare all **n** nodes plus a final *None*)

Average-case running time

In reality, the running time is NOT always the best case, and is NOT always the worst case.

The running time is "**distributed**" between the best and the worst.

For our SearchFortyTwo(L) algorithm the running time is distributed between …

1 and n+1, inclusive

Average-case running time

So, the average-case running time is the **expectation** of the running time which is distributed between 1 and n+1, i.e.,...
Let $\mathbf{t_n}$ be a random variable whose possible values are between 1 and n+1

$$E[t_n] = \sum_{t=1}^{n+1} t \cdot \Pr(t_n = t)$$

We need to know this!

$$E[t_n] = \sum_{t=1}^{n+1} t \cdot \Pr(t_n = t)$$

Average-case running time

To know **Pr(t_n = t)**, we need to be **given** the **probability distribution** of the inputs, i.e., how inputs are **generated** (following what **distribution**).

Now I give you one:
For each key in the linked list, we pick an integer between **1** and **100** (inclusive), **uniformly at random.**

Figure out Pr(t_n = t)

For each key in the linked list, we pick an integer between **1** and **100** (inclusive), **uniformly at random.**

What is ….

$$\Pr(t_n = 1) = 0.01$$

when head is 42

$$\Pr(t_n = 2) = 0.99 \times 0.01$$

head is not 42 and the second one is

$$\Pr(t_n = 3) = (0.99)^2 \times 0.01$$

…

$$\Pr(t_n = n) = (0.99)^{n-1} \times 0.01$$

$$\Pr(t_n = n+1) = (0.99)^n$$

None of the n keys is 42.

the first t keys are not 42 and the t-th is

$$\Pr(t_n = t) = \begin{cases} (0.99)^{t-1} \times 0.01 & 1 \le t \le n \\ (0.99)^n & t = n+1 \end{cases}$$

Now we are ready to compute the average-case running time -- E[t_n]

$$\Pr(t_n = t) = \begin{cases} (0.99)^{t-1} \times 0.01 & 1 \le t \le n \\ (0.99)^n & t = n+1 \end{cases}$$

$$E[t_n] = \sum_{t=1}^{n+1} t \cdot \Pr(t_n = t)$$

$$= \sum_{t=1}^{n} t \cdot (0.99)^{t-1} \times 0.01 \quad + (n+1) \cdot (0.99)^n$$

$$= (n+1) \cdot (0.99)^n + 0.01 \cdot \sum_{t=1}^{n} t \cdot (0.99)^{t-1}$$

$$= 100 - 99 \times (0.99)^n$$

This sum needs a bit of a trick, but can be done!

Calculate the sum (after-class reading)

$$S = \sum_{t=1}^{n} t \cdot 0.99^{t-1} = 1 + 2 \cdot 0.99 + 3 \cdot 0.99^2 + \cdots + n \cdot 0.99^{n-1}$$

$$0.99S = \sum_{t=1}^{n} t \cdot 0.99^{t} = \qquad 1 \cdot 0.99 + 2 \cdot 0.99^2 + \cdots + (n-1) \cdot 0.99^{n-1} + n \cdot 0.99^{n}$$

take the difference of the above two equations

$$0.01S = \sum_{i=0}^{n-1} 0.99^i - n \cdot 0.99^n = \frac{1 - 0.99^n}{1 - 0.99} - n \cdot 0.99^n$$

$$= 100 - (100 + n) \cdot 0.99^n$$

sum of geometric series

You **should** be comfortable with this type of calculations.

The final result

Input distribution: for each key in the linked list, we pick an integer between **1** and **100** (inclusive), **uniformly at random.**

The average-case running time of **SearchFortyTwo(L)** (measured by counting Line #2) is:

$$E[t_n] = 100 - 99 \cdot (0.99)^n$$

If n = 0, then $E[t_n] = 1$, since it's always 1 comparison

If n is very large (e.g., 1000000), $E[t_n]$ is close to 100, i.e., the algorithm is expected to finish within **100** comparisons, even if the worse-case is 1000000 comps.


ONE DOES NOT SIMPLY
TALK ABOUT AVERAGE-CASE RUNNING TIME
WITHOUT TALKING ABOUT INPUT DISTRIBUTION
imgflip.com

asymptotic

upper bound

tight bound

lower bound

Asymptotic notations

O(f(n)): the set of functions that grow **no faster** than **f(n)**

→if $g \in O(f)$, then we say g is asymptotically **upper bounded** by f

Ω(f(n)): the set of functions that grow **no slower** than **f(n)**

→if $g \in \Omega(f)$, then we say g is asymptotically **lower bounded** by f

Asymptotic notations

if **g** $\in$ **O(f)** ***and*** **g** $\in$ **Ω(f),**
then we say **g** $\in$ **Θ(f)**

Θ(f(n)): the set of functions that grow **no slower and no faster** than **f(n)**

we call it the **tight** bound.

The ideas behind asymptotic notations

➔We only care about the **rate** of growth, so **constant factors** don't matter.

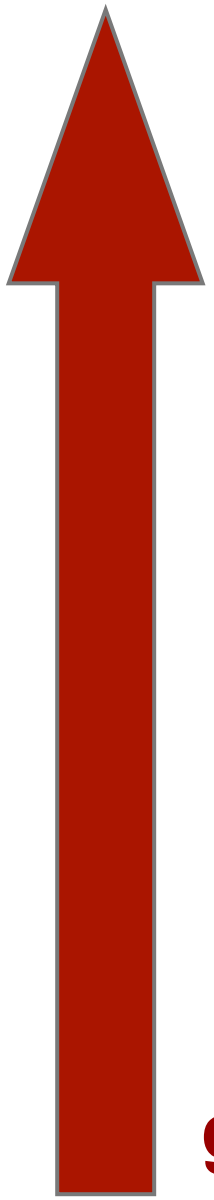
- $100n^2$ and n^2 have the same rate of growth (both are quadrupled when n is doubled)

➔We only care about **large inputs**, so only the **highest-degree** term matters

- n^2 and $n^2 + 263n + 3202$ are the nearly the same when n is very large

growth rate ranking of typical functions

$$f(n) = n^n$$
$$f(n) = 2^n$$
$$f(n) = n^3$$
$$f(n) = n^2$$
$$f(n) = n \log n$$
$$f(n) = n$$
$$f(n) = \sqrt{n}$$
$$f(n) = \log n$$
$$f(n) = 1$$



grow fast

grow slowly

a high-level look at asymptotic notations

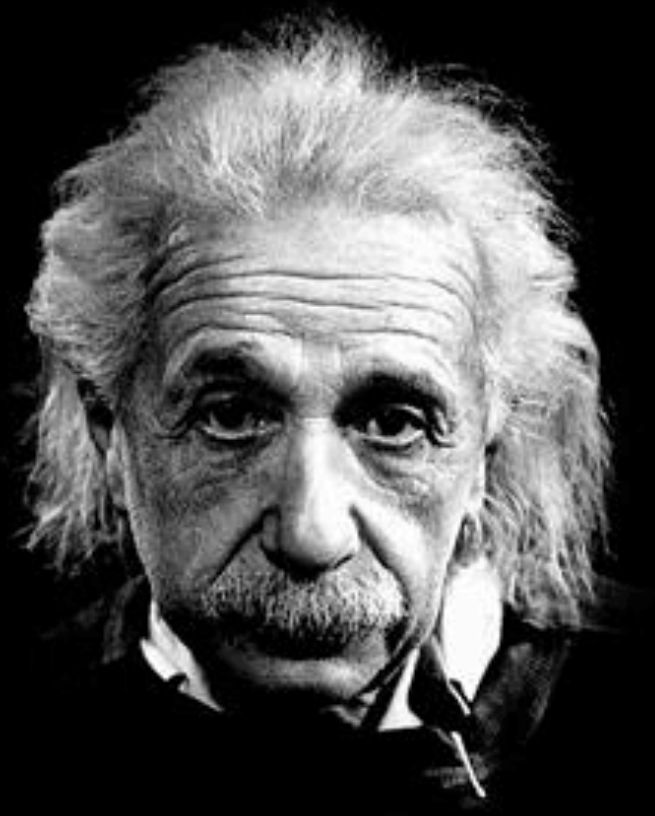
*It is a **simplification** of the "real" running time*

➔ *it does not tell the whole story about how fast a program runs in real life.*

- *in real-world applications, constant factor matters! hardware matters! implementation matters!*

➔ *this simplification makes possible the development of the whole **theory of computational complexity**.*

- ***HUGE idea!***

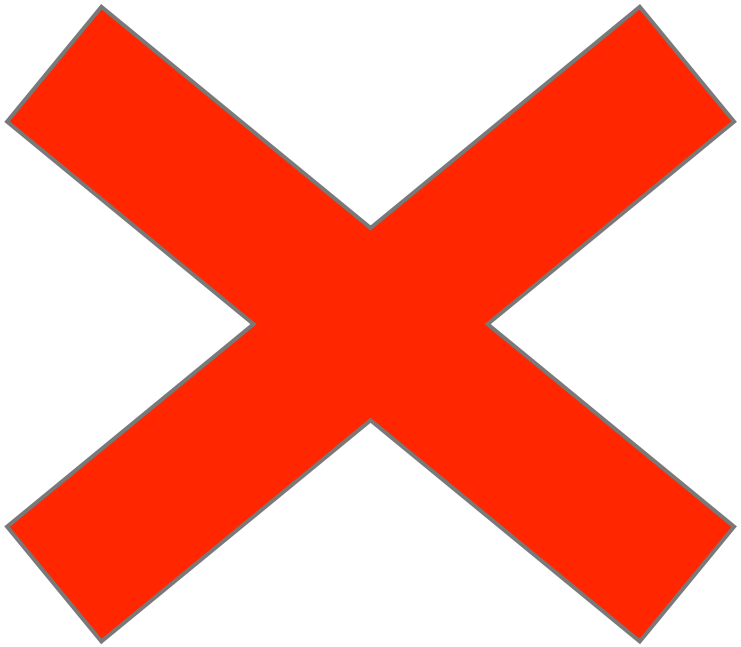


"Make everything as simple as possible, but not simpler."

—Albert Einstein

O is for describing **worst-case** running time

Ω is for describing **best-case** running time



O and **Ω** can **both** be used to upper-bound and lower-bound the **worst-case** running time

O and **Ω** can **both** be used to upper-bound and lower-bound the **best-case** running time

O and **Ω** can **both** be used to upper-bound and lower-bound the **average-case** running time

How to argue algorithm A(x)'s worst-case running time is in O(n^2)

We need to argue that, **for every** input x of size n, the running time of A with input x, i.e., t(x) is **no larger** than **cn^2**, where c > 0 is a constant.

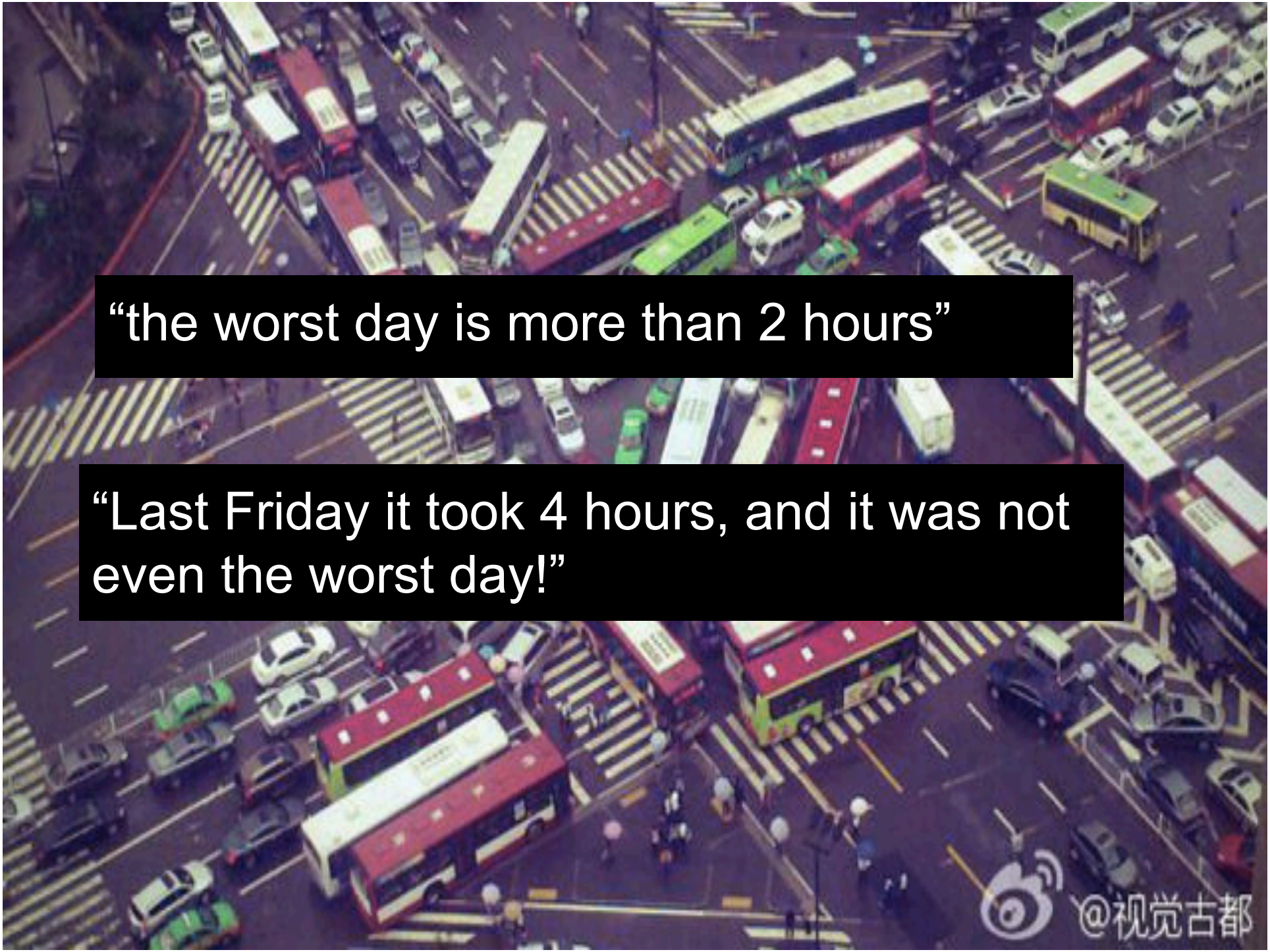
A. for every
B. there exists an
C. no larger
D. no smaller


think about the commuting time from school to home every day
“even the worst day is less than 2 hours”
that means every day is less than 2 hours
@视觉古都

How to argue algorithm A(x)'s worst-case running time is in $\Omega(n^2)$

We need to argue that, **there exists an** input x of size n, the running time of A with input x, i.e., t(x) is **no smaller** than $\mathbf{cn^2}$, where c > 0 is a constant.

A. for every
B. there exists an
C. no larger
D. no smaller


“the worst day is more than 2 hours”
“Last Friday it took 4 hours, and it was not even the worst day!”
@视觉古都

How to argue algorithm A(x)'s best-case running time is in O(n^2)

We need to argue that, **there exists an** input x of size n, the running time of A with input x, i.e., t(x) is **no larger** than **cn^2**, where c > 0 is a constant.

A. for every
B. there exists an
C. no larger
D. no smaller

How to argue algorithm A(x)'s best-case running time is in $\Omega(n^2)$

We need to argue that, **for every** input x of size n, the running time of A with input x, i.e., t(x) is **no smaller** than $\mathbf{cn^2}$, where c > 0 is a constant.

A. for every
B. there exists an
C. no larger
D. no smaller

In CSC263

➔Most of the time, we study the upper-bound on worst-case running time.

➔Sometimes we try to get a tight bound Θ if we can

➔Sometimes we study the upper bound on average-case running time.

Note: exact form & asymptotic notations

In CSC263 homework and exam questions, sometimes we ask you to express running time in **exact forms**, and sometime we ask you to express running time in **asymptotic notations**, so always read the question carefully.

If you feel rusty with probabilities, please read the Appendix C of the textbook. It is only about 20 pages, and is highly relevant to what we need for CSC263.

Appendix A and B are also worth reading.

next week

ADT: Priority queue
Data structure: Heap