CSC263 Winter 2020

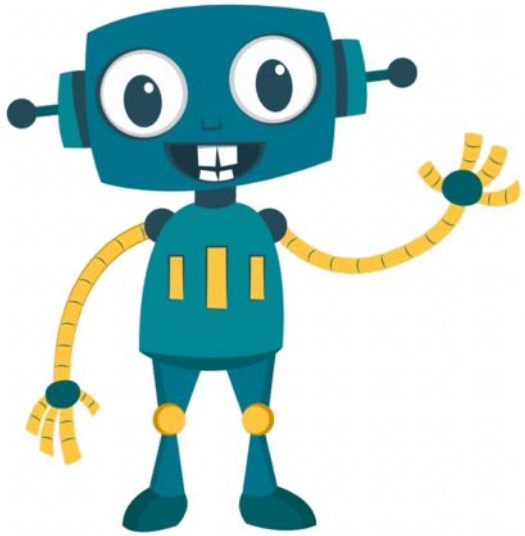
Data Structures and Analysis

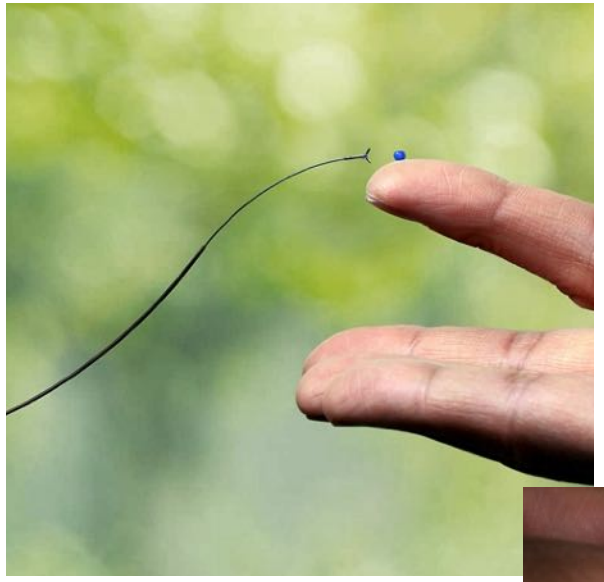
Week 1

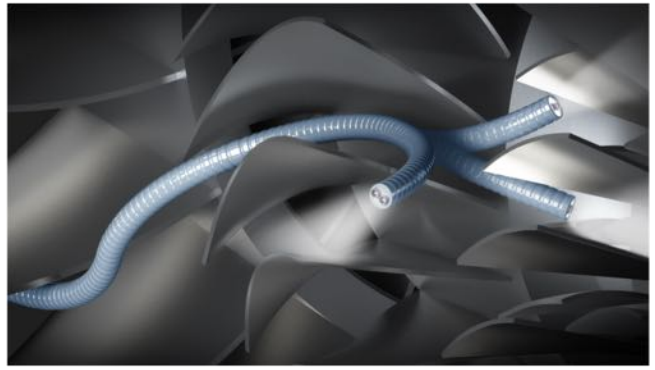
Sushant Sachdeva (Coordinator)
sachdeva@cs.toronto.edu
Office: DH-3092

Jessica Burgner-Kahrs
jessica.burgnerkahrs@utoronto.ca
Office: DH-3064

About me







https://mcs.utm.utoronto.ca/crl/

Outline for Today

Why take CSC263?

What is in CSC263?

How to do well in CSC263?

Start with some basics.

Why take CSC263?

Why take CSC263?

To nail job interviews!

Scenario: The Interview

Interviewer You are given a set of courses, like *CSC236*, *STA256, CSC258, CSC263, CSC309,* where each course has a list of prerequisites. Devise an algorithm that returns a valid ordering of taking these courses.

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

1. For a valid ordering to exist, there must be a course **X** that has no prerequisite.
2. Choose **X** first. Remove **X** from the set of courses, and all other courses' prerequisite list.
3. Find the next course in the set that has no prerequisite
4. Repeat this until all courses are removed from the set.

(This is actually a correct algorithm)

THANKS...

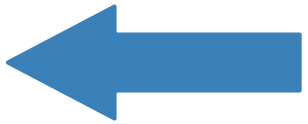
Scenario: The Interview, Take 2

Interviewer You are given a set of courses, like *CSC236*, *STA256, CSC258, CSC263, CSC309,* where each course stores a list of prerequisites. Devise an algorithm that returns a valid ordering of taking these courses.

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

What's in CSC263?

(1) **Data Structures**



and

(2) **Analysis**

What are Data Structures?

Data structures are **smart ways** of **organizing data** / information, to **facilitate access** and **modifications**.

Will allow us to develop **efficient algorithms** easily, in ways that people who don't take CSC263 can't even imagine.

Design algorithms **like a pro**!

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

“Bad programmers worry about the code.

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

-- Linus Torvalds

Which Data Structures?

Heaps, **Binary search trees** -- Data structures for fast priority queues and sorting

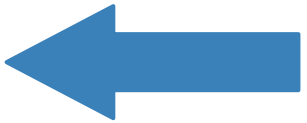
Balanced search trees -- Binary trees that cannot become unbalanced

Graphs -- powerful algorithms for path-finding and exploration

Hash tables, Disjoint set, ...

(1) **Data Structures**

and

(2) **Analysis** 

What Kind of Analyses?

Worst-case analysis

Worst-case sometimes misses the overall picture of what usually happens in practice

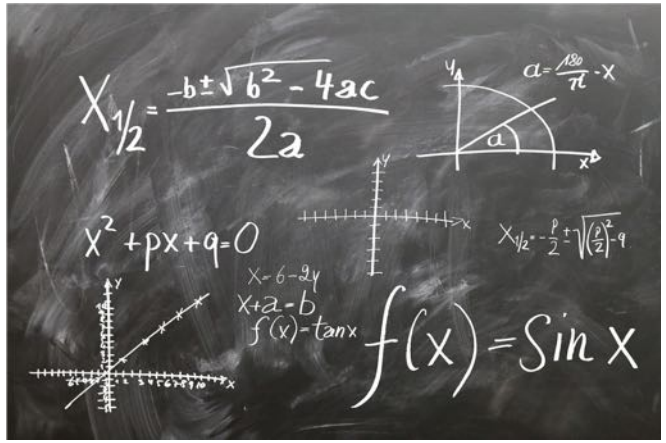
So we also study

Average-case analysis

Amortized analysis

Expected worst-case analysis for randomized algorithms ...

Math and Proofs



Truth

Data structures are fun to learn,

but **analyses** are the real secret sauce.

Cooking

A data structure is like a dish.

Analysis allows you to discover the effect of each ingredient.

Analyses enable you to invent your own dish.

Background (Required)

Theory of Computation

- ❑ Induction
- ❑ Recursive functions, Master Theorem
- ❑ Asymptotic notation, “big-Oh” O, Θ, Ω

Probability Theory

- ❑ Probabilities and counting
- ❑ Random variables
- ❑ Distribution
- ❑ Expectations

How to do well in CSC263?

Be interested.


TEAMWORK

Course Web Page

http://mcs.utm.utoronto.ca/~263/

All course materials can be found on the course web page.

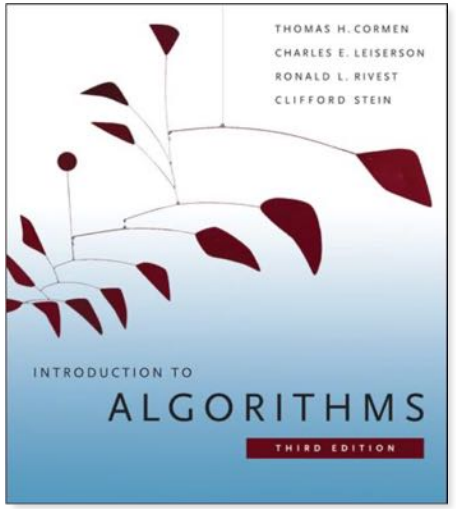
Textbooks

CSC263 Course Notes by David Liu

Short and easy to read, has good exercises.

CLRS: Second and third editions are both fine.
Available online at UofT library

Reading for each week on course web page.



Lectures

Sushant	Wednesday	9am – 11am (IB-245)
		3pm – 5pm (IB-245)
Jessica	Friday	11am – 1pm (IB-245)

passive learning mode mostly

Tutorials

Tuesdays, in various times and places

active learning mode;
problem solving with the help of TA and your classmates

Lectures & Tutorials: Both Important!

Our ultimate goal is to be able to **solve problems!**

Lectures will help you understand the material which the problems are based on.

Tutorials will help you learn how you can solve problems based on the material covered in class.

Lectures and tutorials address different parts of learning.
It is important to have **both**!

Tutorials

Tutorial Problems will be posted shortly before the tutorial, so you may work on it beforehand if you want.

Solutions for tutorials will not be posted.

You should NOT skip tutorials.

“I'll just skip the gym for this week and work out at home instead” -- this almost never works.

Tips for Lectures and Tutorials...

Get involved in classroom **interactions**

- ➔ ask /answer a question
- ➔ make a guess / bet / vote
- ➔ back-of-the-envelope calculations
- ➔ take notes!

Emotional involvement makes the brain learn better!

Discussion Board (Discourse)

Link is available on the course website

For all course related discussions.

All announcements will be posted on the board.

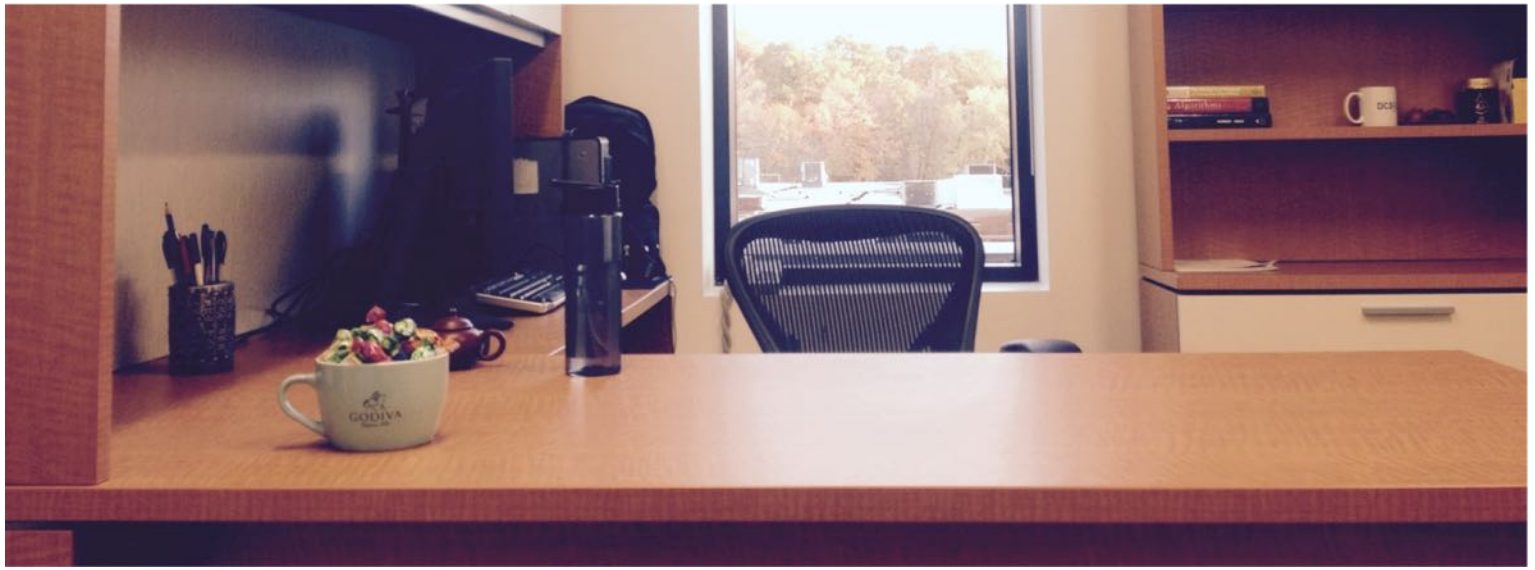
Daily reading is required.

Communicate smartly!

Don’t discuss homework solutions before due dates.

Office Hours

Sushant:	Wednesday	11am – noon
		1:30pm-2:30pm
Jessica:	Friday	2pm - 3pm



Grading Scheme

3 problem sets:	10% each
Out of class Midterm:	20% (or 25%)
Final exam:	50% (or 45%)
TOTAL	**100%**

If your midterm is better than your exam, then midterm is worth 25% and exam is worth 45%

If your exam is better than your midterm, then midterm is worth 20% and exam is worth 50%

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

Problem Sets: Submission

Due typically on Monday, 10pm

Submissions need to be typed using **LaTeX**

Both the **PDF** and the **TeX** source must be submitted to **MarkUS** (link at course web page).

There is typically a programming question in each PS, and you will need to submit the code.

PS: Collaboration

Work in groups of 1-3 students

Collaborate intelligently!

> Remember, **everyone in the group** needs to pass the final exam.

Late Submission Policy

Each of you has total **3 free late days**

Can only use entire days, no partial days.

If a group submits an assignment x days late, all group members lose x free days each.

A group can use x free days on an assignment only if every group member has >=x free days remaining.

Late Submission Policy

If a submission is delayed **beyond the free late days available** to the group member with the fewest free late days remaining, the assignment will be **scored as a 0**.

Academic Integrity

You all know what we expect by now

Do not discuss problem sets outside of your group

Please don't “help” others by helping them solve the PS!

Please don't ask others for help on a PS

What You Can Do to Help

- Discuss examples from lecture and the course materials
- Answer queries on myBB

Learn LaTeX

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

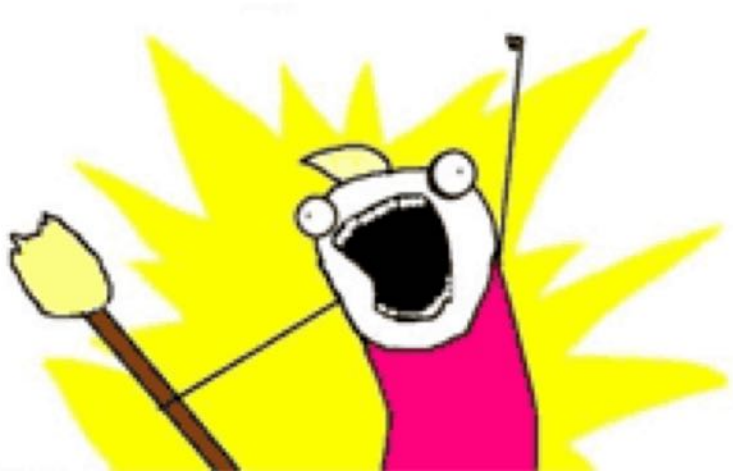
Check the course website for tutorials

Handy tools that do everything in the browser

- www.sharelatex.com
- www.overleaf.com

Problem Set 1 will be posted soon!

Due date: January 27



Exams

Out of Class Midterm

Friday, March 6, 6pm (IB 110)

Duration: 80 mins

Aid: one page double-sided 8.5” x 11” sheet

Final exam

Date to be announced.

Duration: 3 hours

Aid: one double-sided 8.5x11 sheet

Anonymous Feedback

Form available on course website.

My first time teaching this course

Constructive feedback → improved learning experience

Timely feedback → timely improvement

Checklist: How to do well

- ✓ Be interested.
- ✓ Check course web page and discussion board daily
- ✓ Attend lectures and take effective notes
- ✓ Actively solve problems in tutorials
- ✓ Read textbook and notes.
- ✓ Discuss on Discourse.
- ✓ Go to office hours.
- ✓ Work hard on homeworks, and submit on time.
- ✓ Give feedback
- ✓ Do well in exams.

We'll resume in 5 minutes

Abstract Data Type (ADT) and Data Structure

Two Related But Different Concepts

ADT is a **theoretical definition**

- ❏ what data is stored
- ❏ what operations are supported
- ❏ implementation-independent view

Data structure is a **concrete** and **real** implementation

- ❏ how the data is stored
- ❏ how to perform the operations

Real-life Example

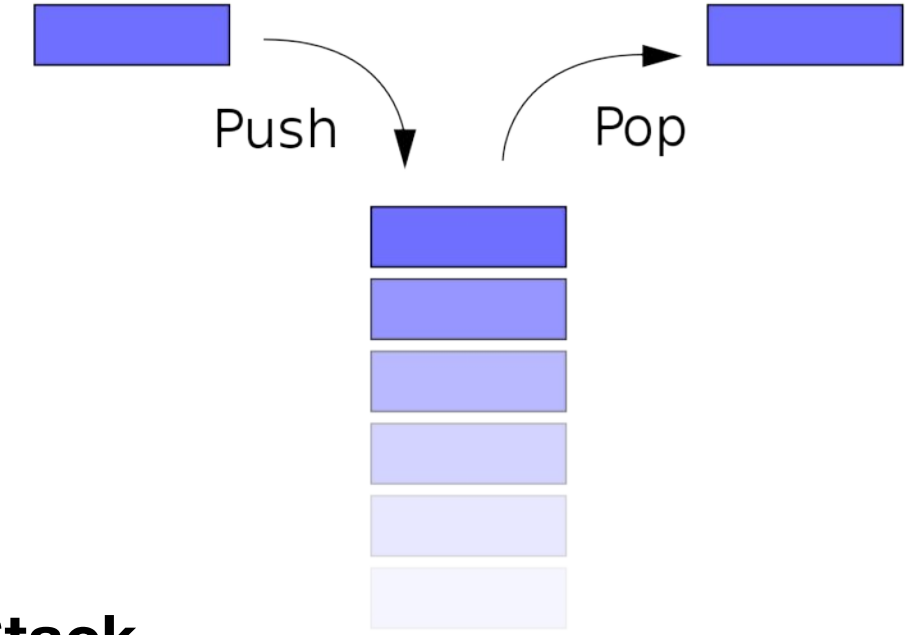
ADT

- ❏ 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 be 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.

Algorithm Complexity

Review

Complexity

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

Time Complexity - Number of **steps** ("**running time**") executed by an algorithm

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

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 measured by counting the number of times **all lines** are executed, or the number of times **some important lines** (such as Line #2 in `SearchFortyTwo`) are executed.

It's up to the problem, or what the question asks, so always read the question carefully.

worst-case
best-case
average-case

Worst-case Running Time

$t_A(x)$ running time of algorithm A with input **x**

When **A** is understood, we 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 want the algorithm

to do well on ALL inputs!

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.,...

$\mathbf{t_n}$ = random variable for the running time on inputs of length n.

It takes values between 1 and n+1

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

We need to know this!

Average-case Running Time

To know Pr($t_n = t$), we need to know the **probability distribution** on the inputs.

e.g. by specifying how inputs are generated

> *Example Distribution*
> For each key in the linked list, we pick an integer between **1** and **100** (inclusive), **independently, 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**.

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

when `head` is 42

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

`head` is not 42 and the second one is

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

$$\dots$$

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

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

none of the n keys is 42.

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

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

Compute Average-case Running Time

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

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

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

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

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

This sum needs a little work, but can be done!

Calculate the Sum (After-class Reading)

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

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

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: Each key in the list is an integer picked {1,2,...,99,100}, **independently 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., 10^6), $E[t_n]$ is close to 100, i.e., the algorithm is expected to finish within **100** comparisons, even if the worse-case is 10^6+1comps.

Average-case Running Time

Need to have **insight on the relative probability** of potential inputs to our algorithm.

Limited scope of average-case running time of an algorithm as it may not be apparent how an "**average**" input looks like.

Probabilistic analysis to yield expected running time.

asymptotic

upper bound

lower bound

tight bound

O

Ω

Θ

Asymptotic Notations

O(f(n)): the set of functions that grow

at most as fast as **f(n)**

➔ if ***g*** ∈ ***O(f)***, then we say ***g*** is asymptotically **upper bounded** by ***f***

Ω(f(n)): the set of functions that grow

at least as fast as **f(n)**

➔ if ***g*** ∈ **Ω*(f)***, then we say ***g*** is asymptotically **lower bounded** by ***f***

Asymptotic Notations

if **g ∈ O(f)** and **g ∈ Ω(f),**

then we say **g ∈ Θ(f)**

Θ(f(n)): the set of functions that grow **as fast as f(n)**

we call it an asymptotically **tight** bound.

Key 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 $n^2 + 26n + 320$ are nearly the same*
>
> *when n is very large*

Growth Rate Ranking of Typical Functions

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

grows fast

grows slowly

A high-level Look at Asymptotic Notations

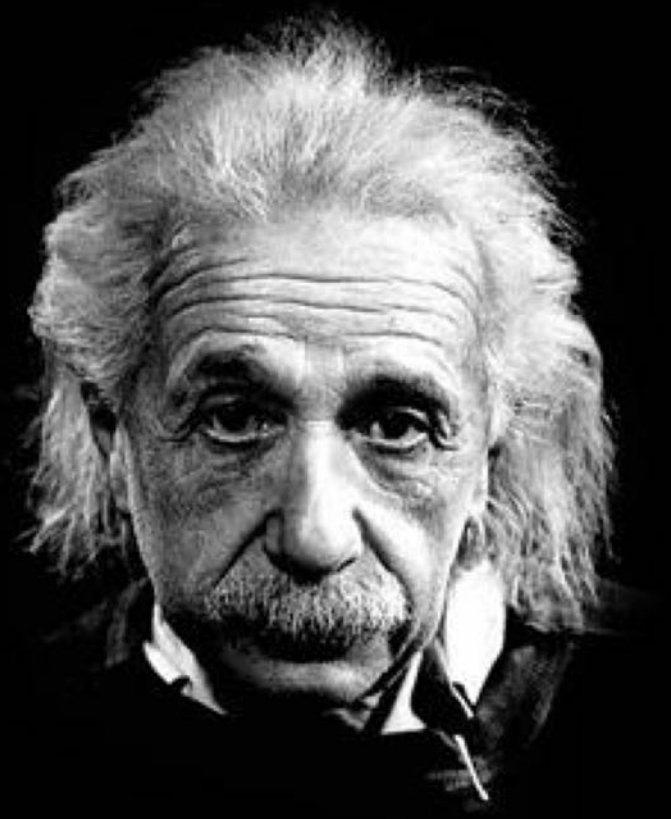
Simplification of the “real” running time

➔ 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 permits the development of the whole **theory of computational complexity**.

- **HUGE idea!**

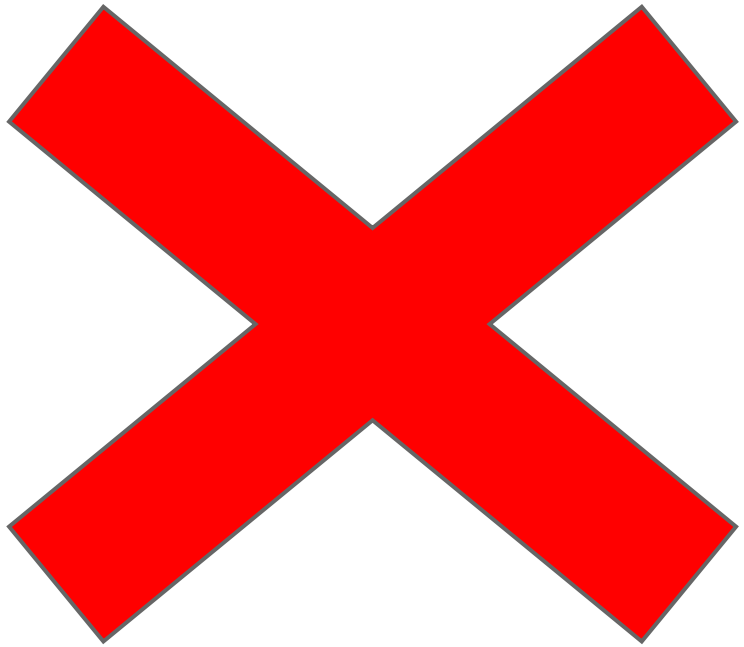


"Make everything as simple as possible, but not simpler."

—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 **worst / best / average** case running time

Runtime (best/worst/average case) is a function of n, and any function can be asymptotically upper or lower bounded.

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

Example

To prove “**the tallest person in the room is at most 2 metres**”.

You need to show **every person** in the room is **no taller** than 2 metres.

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

Example

To prove “**the tallest person in the room is at least 2 metres tall**”.

You just need to find **one person** in the room is **no shorter** than 2 metres.

This person may not even be the tallest one in the room.

How to argue algorithm A(x)'s best-case running time is in O(n²)

We need to argue that, _____ input x of size n, the running time of A with input x, i.e., t(x) is _____ than **cn²**, where c > 0 is a constant.

there exists an

no larger

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, _____ input x of size n, the running time of A with input x, i.e., t(x) is _____ than $\mathbf{cn^2}$, where $c > 0$ is a constant.

for every

no smaller

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

In CSC263

We usually study upper-bounds on worst-case running time.

We will try to get a tight bound Θ if we can.

We also study the upper/lower bounds on average-case running time.

Note: exact form & asymptotic notations

In CSC263 homework and exam questions, we'll sometimes ask you to express the running time in **exact forms**, and sometime, we'll ask you to express running time in **asymptotic notations**, so again, 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.

Today we learned / reviewed

ADT and Data structures

Best-case, worst-case, average-case analysis

Asymptotic upper/lower bounds

Next week

ADT: Priority queue

Data structure: Heap