OntarioTech Science

Kourosh Davoudi
kourosh@ontariotechu.ca

Lecture 1: Introduction

CSCI 3070U: Analysis and Design of Algorithms

Welcome to 3070U !

In today's class:

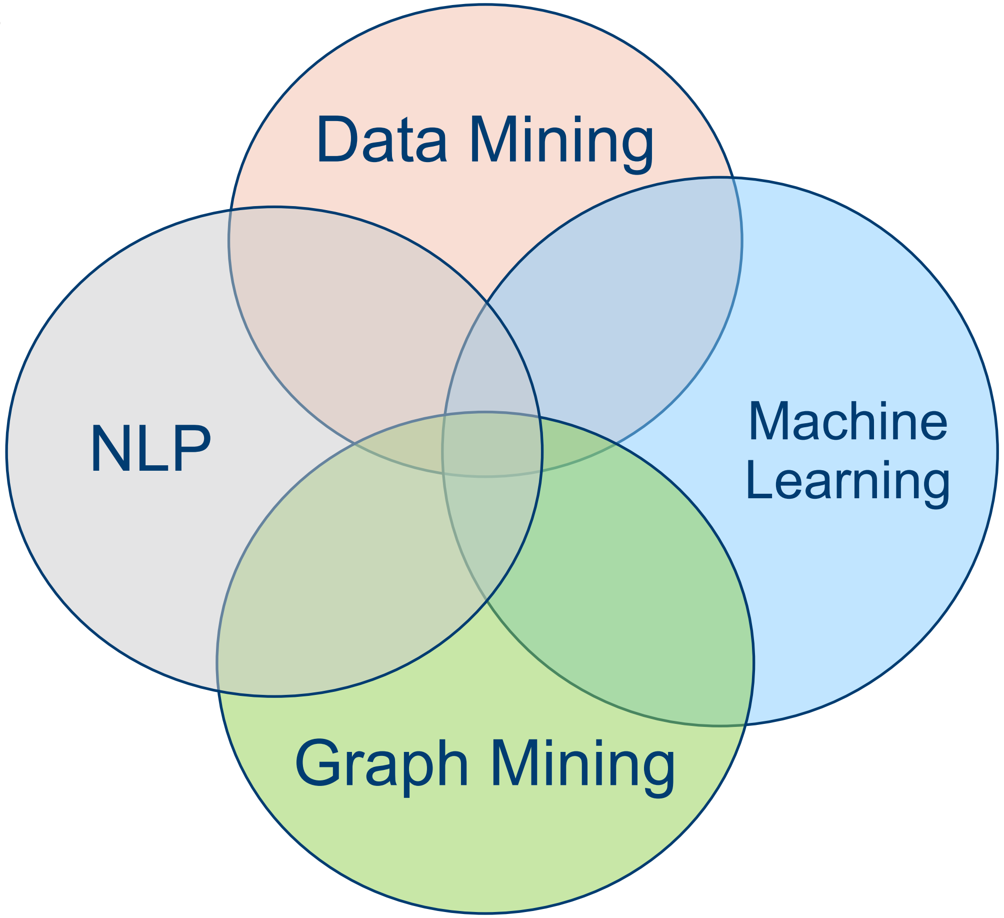
- We get to know each other
- We learn about:
  - Course Objectives
  - Course Structure
  - Course Content
- Warm up !
  - Basic Concepts
  - Case Study: Insertion sort

Kourosh Davoudi

- Assistant Professor in Computer Science (Ontario Tech University)
- Postdoctoral: Computer and Management Science (University of Waterloo)
- PhD: Computer Science (York University)
- Previous/current industry partners:

Kourosh Davoudi

- Areas of interest:

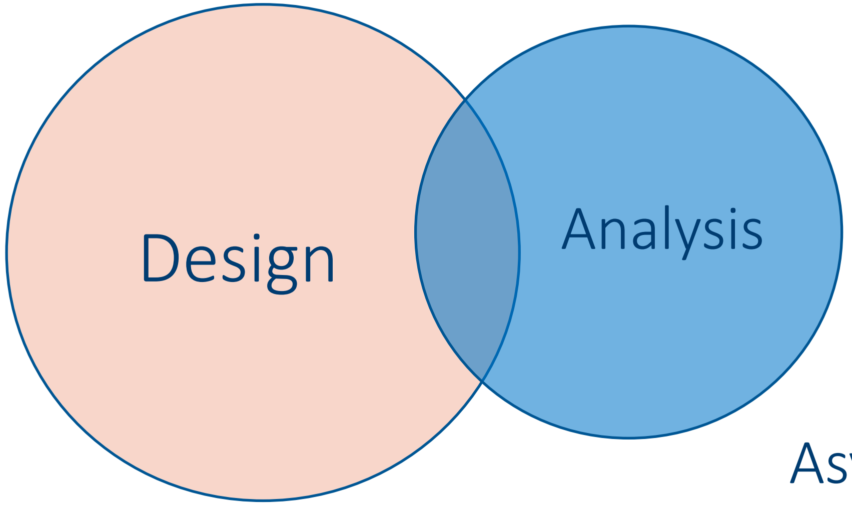


How about you?

- How do you like your major?
- What is your favorite course?
- Which jobs in computer science are you interested in?
- What do you expect from this course?
- Which programming languages have you work with?
- …

Course Content

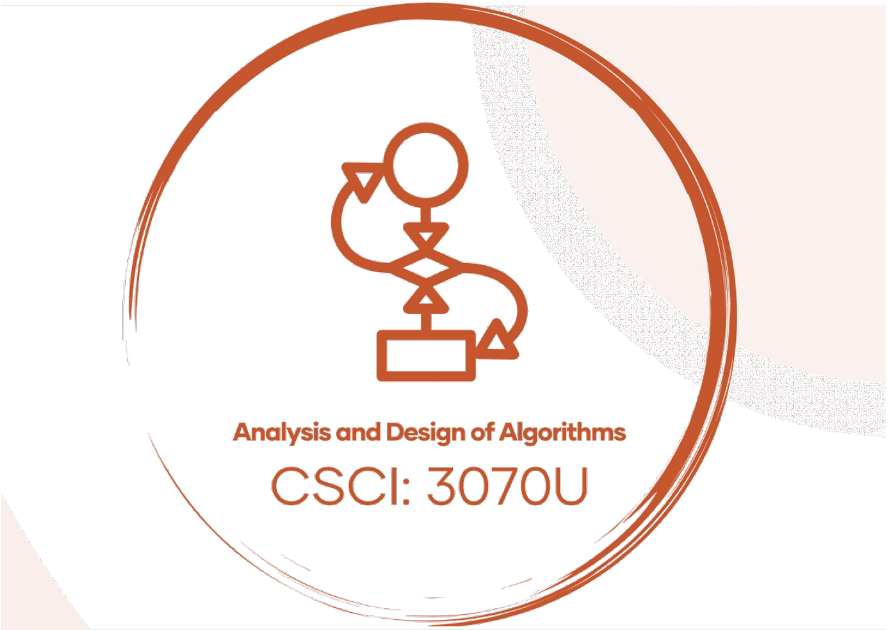
What is this course about?



Divide & conquer
Dynamics Programming
Greedy Algorithms
…

Asymptotic notion
Recurrence equations
…

Topics in a big picture (tentative)



CSCI 3070U Outline: Course Outline and Lectures (Fall 2022)

Week	Topic	Details	Deadline	Tutorial
1	Algorithm Time Complexity Analysis	• Introduction • Case Study: Insertion Sort • Case Study: Fibonacci Series • Asymptotic Notations		

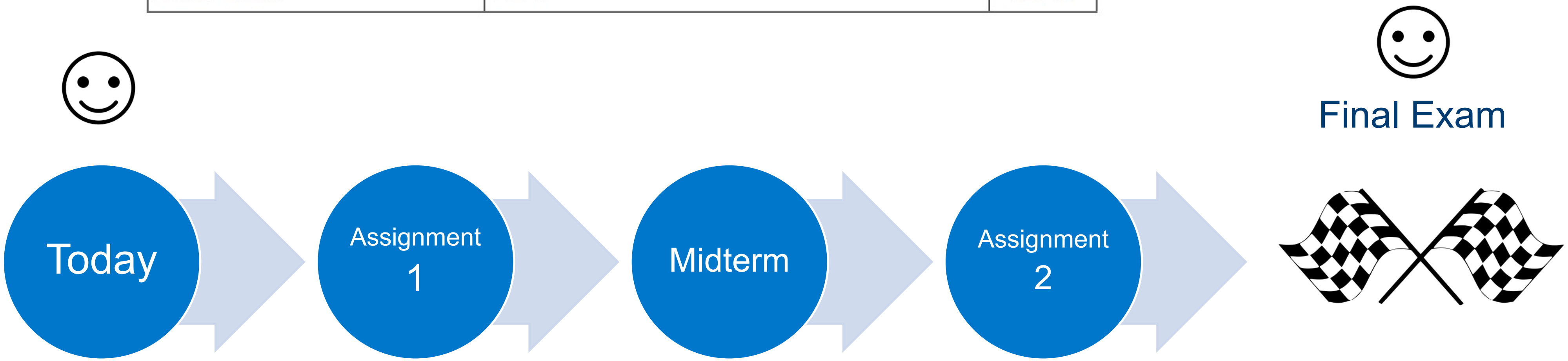
2	Divide and Conquer	• Binary Search
		• Merge Sort
		• Recurrence
		  ○ Substitution
		  ○ Recursion Tree
		  ○ Master Theorem
				Tutorial-1: (Sep-19)
				Running times, induction, asymptotic notation

3	Sort Algorithms	• Heaps
		  ○ Heap Sort
		  ○ Priority Queue
				Tutorial-2: (Sep-26)
				Complexity
				Bubble Sort,
				Binary Search

4		• Quick Sort • Linear Time Sorts • Radix Sort • Bucket Sort • Review + Midterm	Midterm Oct-6 A2-Due Oct-09	Tutorial-3: (Oct-03) Solving Recurrence
		(Oct 10 -Oct 16)		
5	Dynamic Programming	• Fibonacci (revisit) • Matrix Chain Multiplication		Tutorial-4: (Oct-17) Divide & Conquer/Heap
6	Dynamic Programming	• Longest Common Subsequence • 0/1 Knapsack		Tutorial-5: (Oct-24) Review
7	Greedy Algorithms	• Counting Coins • Fractional Knapsack • Huffman Code		Tutorial-6: (Oct-31) Dynamic Programming, LCS
8	Branch and Bound	• Project Management • 0/1 Knapsack Problem		Tutorial-7: (Nov-07) Activity Selection
9	Graph Algorithms	• Graph Representation • Graph Search (BFS, DFS)	A2-Release Nov-14	Tutorial-8: (Nov-14) TSP
10		• Topological Sort • Minimum Spanning Tree (MSP)	A2-Due Nov-24	Tutorial-9: (Nov-21) Graph Search
11		• Shortest Path • Maximum Flow		Tutorial-10: (Nov-28) Minimum Spanning Tree
12		• Theory of Computation • Review		

Evaluation

Component	Due Date	Weight
Class Participation and Activities		10 %
Assignment 1	October 9, 2022, before 11:59 PM	20 %
Assignment 2	November 24, 2022, before 11:59 PM	20 %
Midterm Exam	October 6, 2022 @ 2:10 PM (**Location: TBA**)	20 %
Final Exam	TBA	30 %



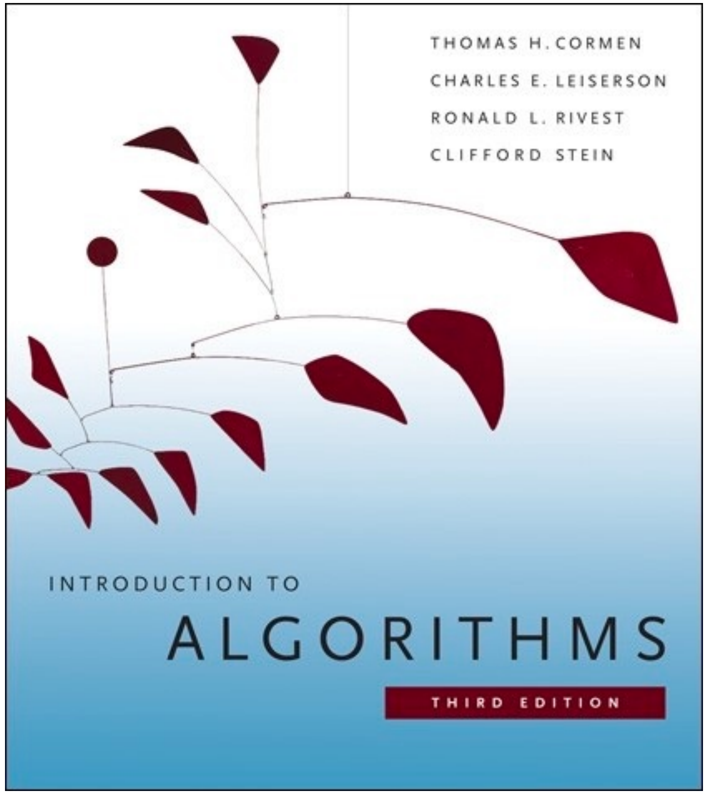
How to participate?

- Attending the lectures
- Participating in in-class/out-of-class Activities
- Participating in class discussion
- Posting questions/answers to piazza
- Presenting an interesting topic in class
    - You can coordinate with me!

Tutorials

- Schedule:
  - Friday : 12:40-14:00 (40267)
  - Thursday: 11:10-12:30 PM (44493):
  - Wednesday: 12:40-14:00 (44764) :
  - Thursday : 11:10-12:30 (44937):
  - Wednesday : 12:40-14:00 (45080):
  - Monday : 12:40-14:00 (45082):
- TAs:
  - Hooria Hajiyan (hooria.hajiyan@ontariotechu.net)
  - Tamilselvan Balasuntharam (tamilselvan.Balasuntharam@ontariotechu.ca)
  - Riley Weagant (riley.weagant@ontariotechu.net)

Textbook



Introduction to Algorithms, Third Edition

By: Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein

CLRS !

Course Repository:

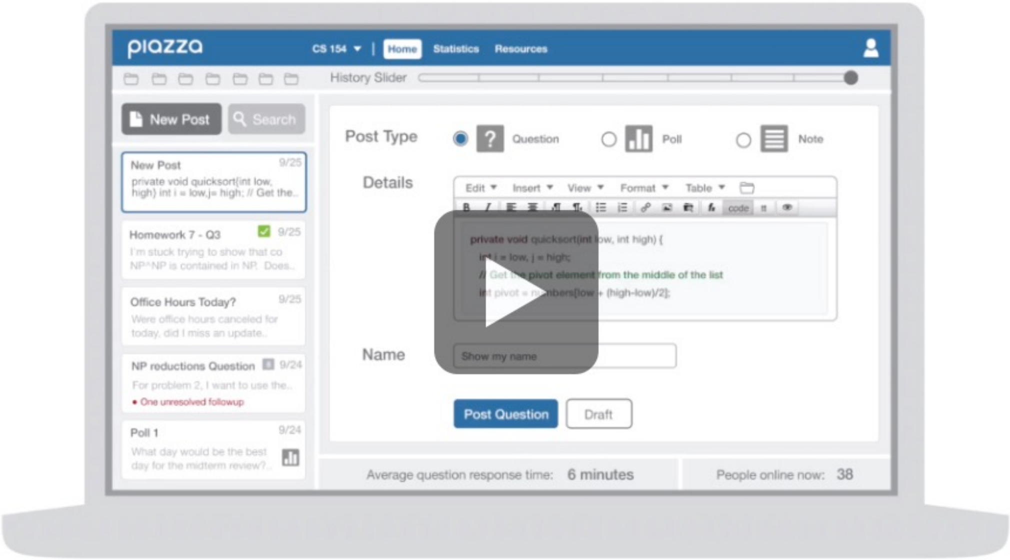
All materials will be posted on Canvas

(https://learn.ontariotechu.ca/)

Communication

piazza

- Please note that questions about lectures/assignments/exams should be posted to piazza.

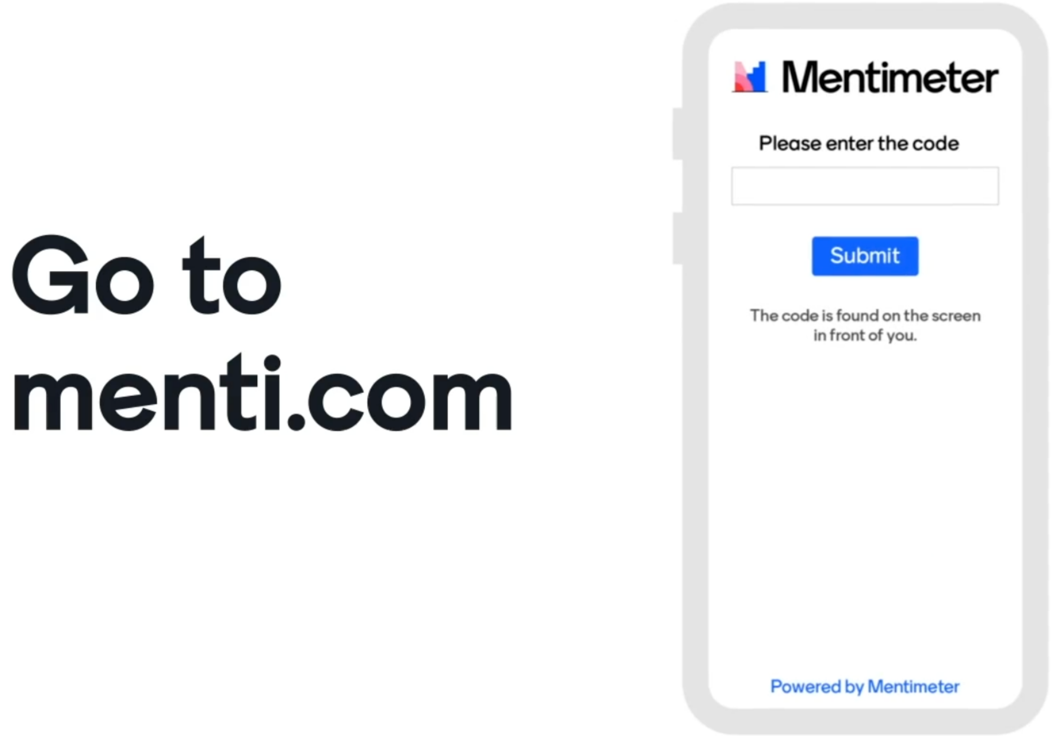


Sign in instruction is available in Canvas

Activities

We will use mentimeter for our class activities.

https://www.mentimeter.com/



Office Hours and Contacts

Course Instructor:

Dr. Kourosh Davoudi

- **Email:** kourosh@ontariotechu.ca
- **Office Location:** UA 4013
- **Office Hours:** Online by appointment

- **Phone:** (905) 721-8668 x 2779
- **Webpage:** http://dmlab.science.uoit.ca/hdavoudi/

How to approach this course?

Technical Outcomes for week 1:

- What is an algorithm?
- How to analysis computational time of an algorithm?
- Asymptotic notation: why do we need them?
- Examine two algorithms:
  - insertion sort and
  - Fibonacci Series

What is algorithm?

- An algorithm is any well-defined computational procedure that
  - Takes some value, or set of values, as input
  - Produces some value, or set of values, as output
- Example: Cooking a food !
- Example: Sorting algorithms:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \le a'_2 \le \cdots \le a'_n$.

Design and Analysis of algorithm

- Design: Method for developing algorithm
- Analysis: Abstract mathematical comparison of algorithms

- Two important aspects of an algorithm:
  - Correctness
  - Efficiency

Who needs algorithms?

- Google
  - Needs it to rank the webpages !
- Amazon
  - Needs it to find the best products !
- Facebook
  - Needs it to recommend the friendship !
- The Globe and Mail
  - Needs it to predict good users !
- A salesman
  - Needs it to save time when selling his products in different cities !

Is algorithm efficiency important?

- Generally, depends on the size of problem
  - For small inputs, the algorithm efficiency matters less
  - There are some exceptions !

- Algorithms are usually evaluated by their input size
  - But what is the input size?
    - Number of elements in input array
    - Number of line in the input file
    - …

Algorithm Analysis?

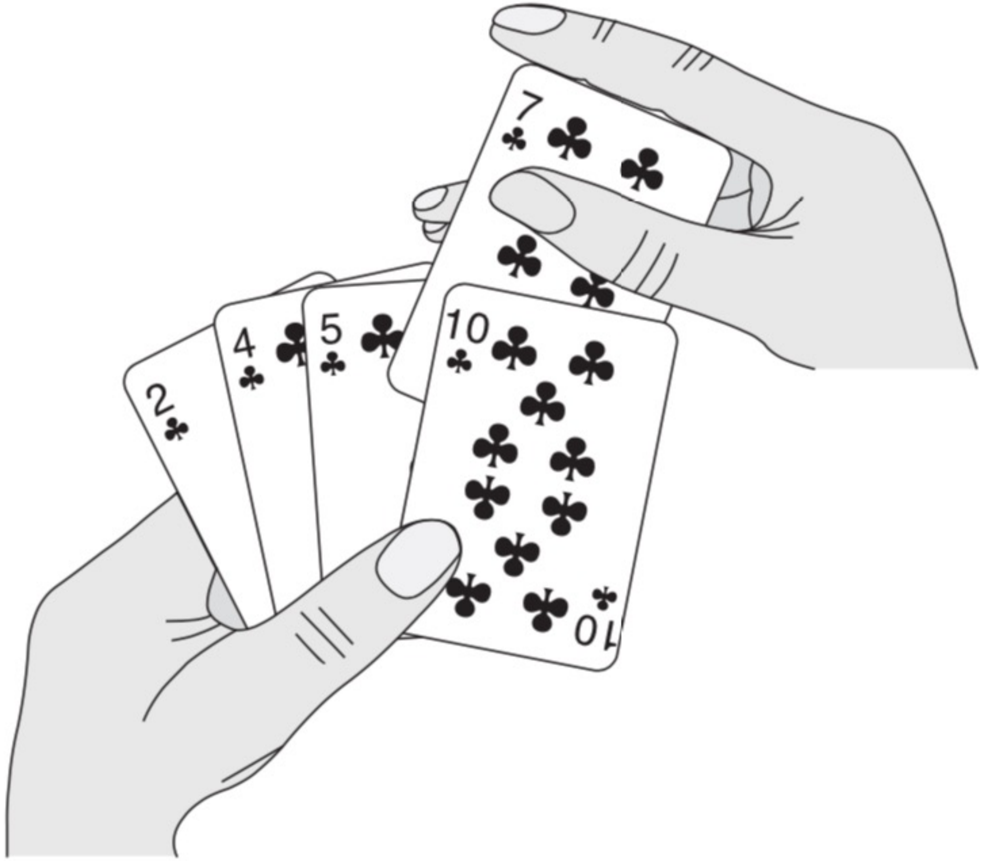
- What kinds of analysis?
  - Time Complexity (CPU)
  - Spatial Complexity (Disk & Memory)
  - Correctness
  - Termination
  - …
- We usually consider Time Complexity !
  - Challenge: One algorithm may be faster on a fast machine !
  - Challenge: Time complexity depends on input

Algorithm Time/Spatial Complexity

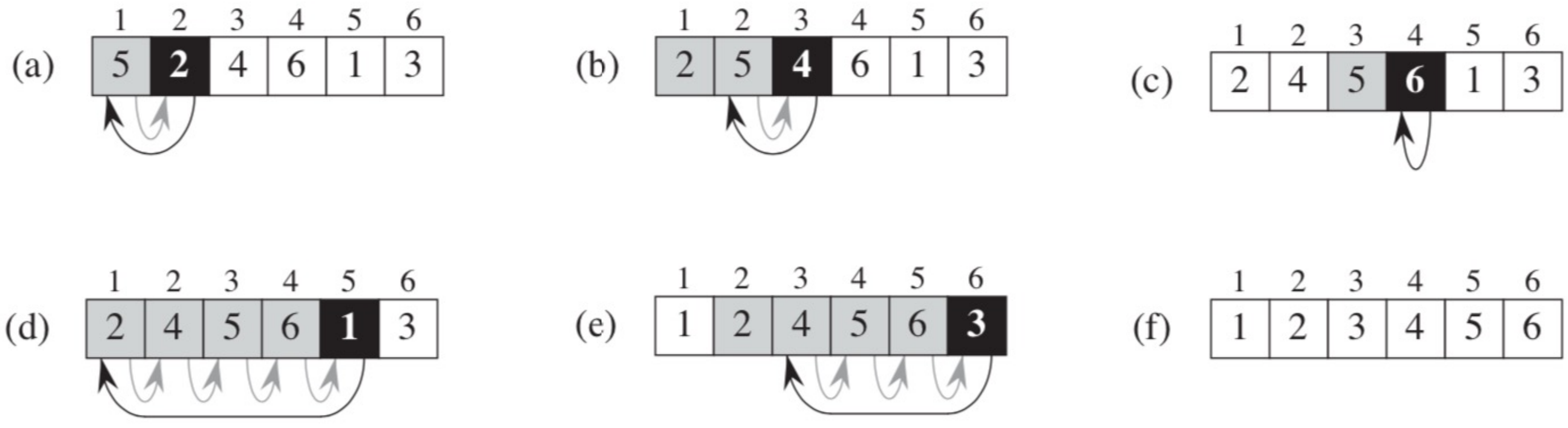
- We’ll examine temporal and spatial complexity:
  - Average case complexity
  - Worst case complexity (far easier)
  - Best case complexity (far easier)

- Random-Access Machine (RAM)
  - Help us avoid very specific machine
  - There is no concurrency
  - Each simple instruction such as +, - , * , / , = , >, ... takes constant amount of time

Case Study: Insertion Sort



Case Study: Insertion Sort



Insertion Sort in Action

https://visualgo.net/en/sorting

Try couple of inputs !

Case Study: Insertion Sort

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Case Study: Insertion Sort

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

t_j : The number of times the **while** loop test in line 5 is executed for that value of j

Case Study: Insertion Sort

	INSERTION-SORT(A)	*cost*	*times*
1	**for** $j = 2$ **to** $A.length$	c_1	n
2	$key = A[j]$	c_2	$n-1$
3	// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4	$i = j - 1$	c_4	$n-1$
5	**while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6	$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7	$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8	$A[i+1] = key$	c_8	$n-1$

t_j : The number of times the while loop test in line 5 is executed for that value of j

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

Case Study: Insertion Sort

INSERTION-SORT(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
$i = j-1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

- Best Case:

$$t_j = 1$$

- Worst Case:

$$t_j = j$$

$$\boxed{\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\ &\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) . \end{aligned}}$$

Case Study: Insertion Sort

- Best Case $t_j = 1$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\,.
\end{aligned}
$$

- A linear function of n $an + b$

Case Study: Insertion Sort

- Worst Case

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) . \end{aligned}$$

- Quadratic function of n

$$an^2 + bn + c$$

Case Study: Insertion Sort

- Correctness
  - We often use a *loop invariant* to help us understand why an algorithm gives the correct answer.
  - A loop invariant is a **formal statement** about the relationship between variables which is true

Loop Invariant :The sub $A[1\,..\,j-1]$ array consists of the elements originally in $A[1\,..\,j-1]$, but in sorted order.

Case Study: Insertion Sort

Loop Invariant :The sub $A[1\,..\,j-1]$ array consists of the elements originally in $A[1\,..\,j-1]$, but in sorted order.

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1..j-1].
    i = j - 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i - 1
    A[i + 1] = key
```

1. Initialization: ($j = 2$)

2. Invariant Maintenance (★)

3. Termination: ($j = A.length + 1$)

Case Study: Fibonacci Series

- We define the **Fibonacci numbers** by the following recurrence:

$$
\begin{aligned}
F_0 &= 1\,, \\
F_1 &= 1\,, \\
F_i &= F_{i-1} + F_{i-2} \qquad \text{for } i \geq 2\,.
\end{aligned}
$$

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ \ldots\,.$$

Case Study: Fibonacci Series

```
FIB(n)
if n == 0 or n == 1
        return 1
else
        return FIB(n − 1) + FIB(n − 2)
```

$$
\begin{aligned}
T(0) &= c_1 \\
T(1) &= c_2 \\
T(n) &= T(n-1) + T(n-2) + c_3
\end{aligned}
$$

Asymptotic Notations

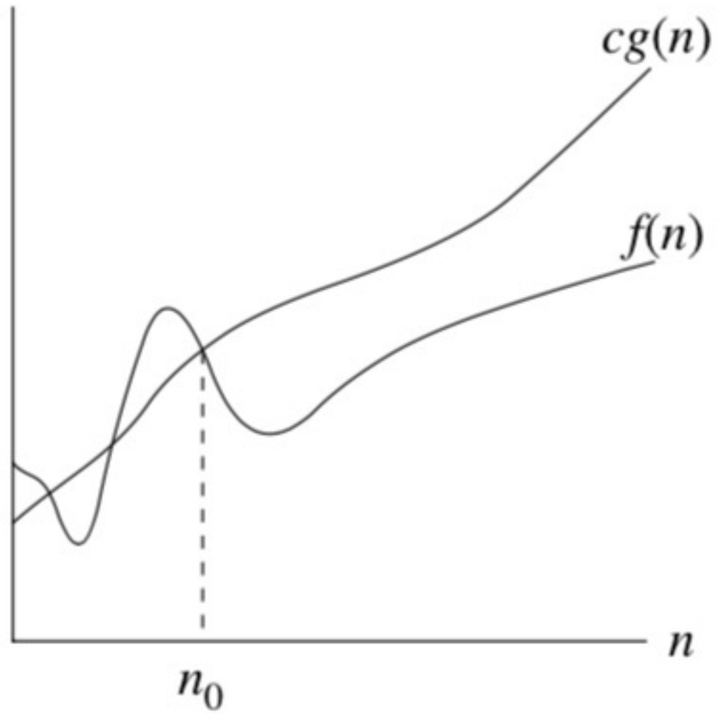
- Why asymptotic notations?
  - Remember Insertion sort time complexity:

$$an^2 + bn + c$$

  - They
    - Drop lower-order terms
    - Ignore the constant coefficient in the leading term
    - Provide another abstraction to ease analysis and focus on the important features

Asymptotic Notations

- O - notation



$$f(n) \in O(g(n))$$

or

$$f(n) = O(g(n))$$

$$O(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0\} .$$

Asymptotic Notations

- O - notation

$$O(g(n)) = \{ f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le f(n) \le cg(n) \text{ for all } n \ge n_0 \} .$$

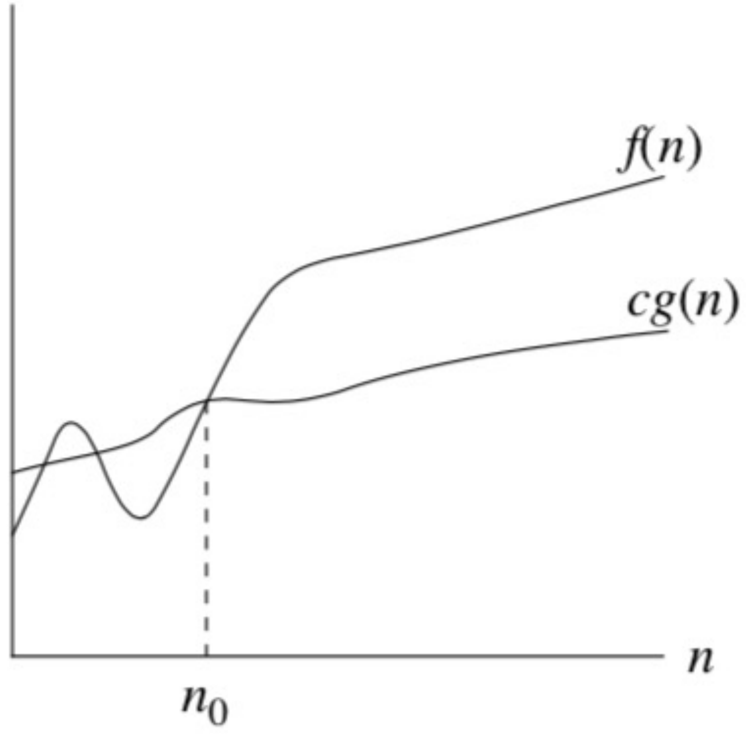
Example: $2n^2 = O(n^3)$, with $c = 1$ and $n_0 = 2$

Examples of functions in $O(n^2)$:

n^2	n
$n^2 + n$	$n/1000$
$n^2 + 1000n$	$n^{1.99999}$
$1000n^2 + 1000n$	$n^2/\lg\lg\lg n$

Asymptotic Notations

- Ω - notation



$$f(n) \in \Omega(g(n))$$

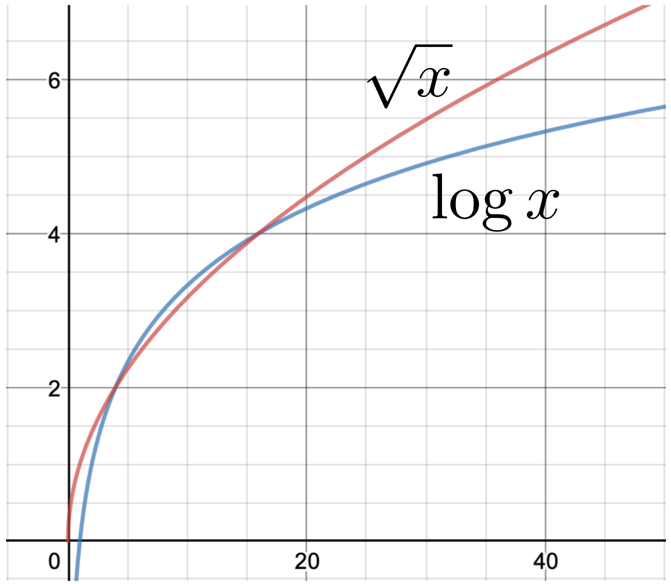
$$f(n) = \Omega(g(n))$$

$$\Omega(g(n)) = \{f(n) : \text{ there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\} .$$

Asymptotic Notations

- Ω - notation

$$\Omega(g(n)) = \{f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } 0 \le cg(n) \le f(n) \text{ for all } n \ge n_0\}.$$



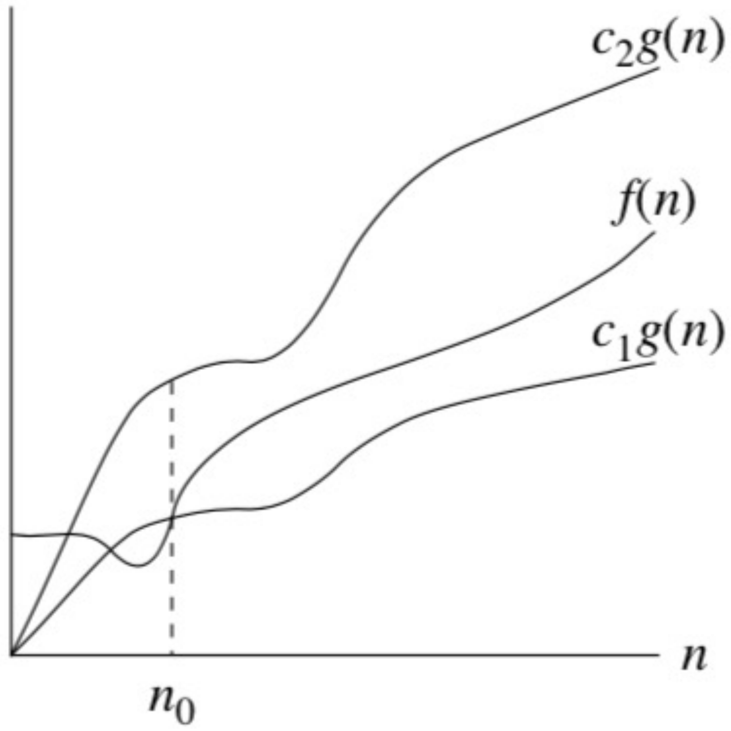
Example: $\sqrt{n} = \Omega(\lg n)$, with $c = 1$ and $n_0 = 16$.

Examples of functions in $\Omega(n^2)$:

n^2	n^3
$n^2 + n$	$n^{2.00001}$
$n^2 - n$	$n^2 \lg\lg\lg n$
$1000n^2 + 1000n$	2^{2^n}
$1000n^2 - 1000n$	

Asymptotic Notations

- Θ - notation



$$f(n) \in \Theta(g(n))$$

or

$$f(n) = \Theta(g(n))$$

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\} .$$

Asymptotic Notations

- Θ - notation

$$\Theta(g(n)) = \{f(n) : \text{there exist positive constants } c_1, c_2, \text{ and } n_0 \text{ such that } 0 \le c_1 g(n) \le f(n) \le c_2 g(n) \text{ for all } n \ge n_0\}.$$

Example: $n^2/2 - 2n = \Theta(n^2)$, with $c_1 = 1/4$, $c_2 = 1/2$, and $n_0 = 8$.

Theorem

$f(n) = \Theta(g(n))$ if and only if $f = O(g(n))$ and $f = \Omega(g(n))$

$$f(n) = \Theta(g(n))$$

Examples

(1) $3n^2 - 100n + 6 \quad = \quad O(n^2)$

$$3n^2 > 3n^2 - 100n + 6$$

(2) $3n^2 - 100n + 6 \quad = \quad \Omega(n^2)$

$$2.99n^2 < 3n^2 - 100n + 6$$

(3) $3n^2 - 100n + 6 \quad \neq \quad O(n)$

$$cn \geq 3n^2 - 100n + 6$$

You cannot find c !

(4) $3n^2 - 100n + 6 \quad = \quad \Theta(n^2)$

$$because\ O \quad and \quad \Omega$$

(5) $3n^2 - 100n + 6 \quad \neq \quad \Theta(n)$

$$because\ \Omega \quad only$$

Examples

(1) Is $2n + 1 = O(2n)$?

Is $2n + 1 \leq c * 2n$?

Yes, if $c \geq 2$ for all n

(2) Is $2^{2n} \neq O(2^n)$

$2^{2n} \leq c \cdot 2^n$ for all $n \geq n_0$?

$2^{2n} = 2^n \cdot 2^n \leq c \cdot 2^n$

$2^n \leq c$ But no constant is greater than all 2^n

YES, it is correct !

Theorems:

- For polynomial degree *d* $p(n) = \sum_{i=1}^{d} a_i \, n^i \quad (a_d > 0)$

$$p(n) \in \Theta(n^d)$$

Example: $n^3/1000 - 100n^2 - 100n + 3 \in \Theta(n^3)$

Wrap-up

- We leaned
  - Algorithm basics
  - How to analyze an algorithm:
    - Time complexity
    - Correctness
  - Asymptotic notations as a mathematical model of comparing time complexity
  - Recursive algorithms time complexity analysis needs solving recurrent equations !