EL-GY9343

Data Structure and Algorithm

Lecture 1: Syllabus, Introduction, and Asymptotic notation

Instructor: Pei Liu

Welcome

- Instructor: Pei Liu
  - Contact: peiliu@nyu.edu
  - Office: 370 Jay Street, Room 935 (N/A Spring 2022)

- TA: Xinzhu Han
  - Email: xh1341

- All office hours will be virtual over Zoom.
  - Link and time can be found on Brightspace;
  - For homework related questions, it is recommended to go to the office hour hosted by the grader on Tuesday.

Class Recordings

- Only first two week lectures will be broadcast and recorded via Zoom
    - For the rest of the semester, no Zoom lectures;
    - I have uploaded videos from previous semester, which could be outdated;
    - Make sure you check the slides for the latest announcements such as homework, exam schedule.
- Whiteboard will be available online
    - https://1drv.ms/u/s!AhGPBmraRihnkGD8UnfdP2pau5Qp

Why taking this course

▸ You can learn

    ▸ Classic algorithms for classic problems, mathematical insights

    ▸ Techniques for analyzing performance of various algorithms

    ▸ How to design good algorithms for solving real-world problems

▸ Improving programming skills

    ▸ Then you can rock in classes, job interviews, etc.

▸ It's also fun!

Prerequisites

▸ Basic knowledge of fundamental data structures

   ▸ stacks, queues, heaps, ...

▸ Some programming experience

   ▸ C, C++, Python, Java, etc.

▸ Discrete mathematics, probabilities, ...

▸ Should not take this course if you have taken a similar course e.g. CS6033 with a B or better grade.

Textbook

Introduction to Algorithms, 3rd Edition, by Thomas H. Cormen, Charles E. Leiserson, Rondald L. Rivest and Clifford Stein, MIT Press, 2009; ISBN-13: 9780262033848. It is known as CLRS.

Free access to CLRS (on the library web site *https://library.nyu.edu*, search the book name under "Books & More").
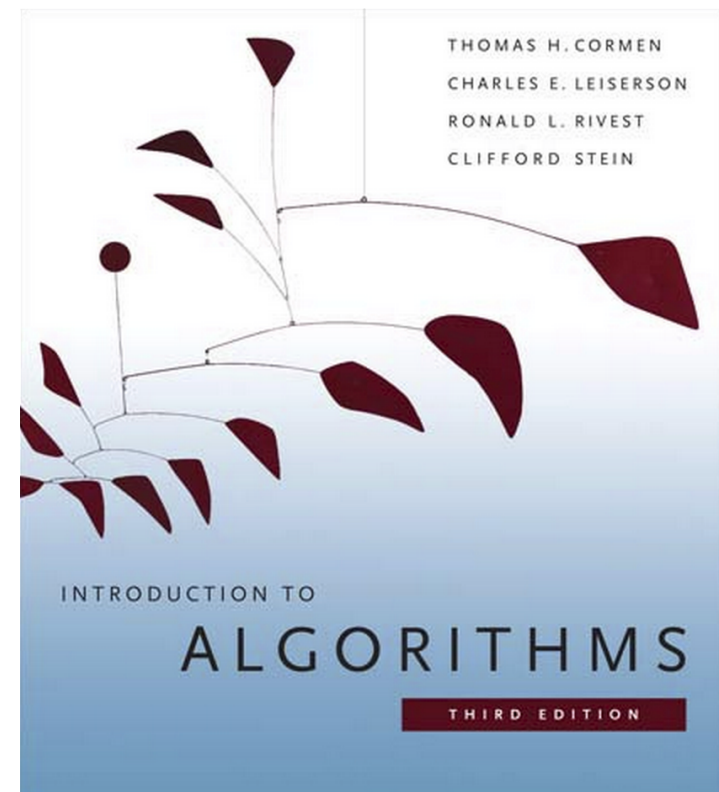
Grading policy

▸ Your final grade will be calculated as:

Homework	10%
Midterm	40%
Final	50%

▸ No extra work to improve grade!

Homework

▸ Key component to mastering the course material

  ▸ Very good exercise and practice

  ▸ Will not do well on exams if you have not done the hw

  ▸ Will be assigned weekly or biweekly

▸ Submission guidelines

  ▸ Homework is due in one week after it is released

  ▸ You need to download an APP called "Adobe Scan" on your phone, and use it to scan your homework.

  ▸ Please submit in a single PDF file, instead of multiple files.

Exams

▸ One mid-term, date TBD

▸ One final exam, date TBD

▸ Close-book and two page cheatsheets

▸ Attendance at exams is mandatory

▸ **DON'T make travel arrangement for before last day of the semester.**

Remember: If you miss an exam without a valid excuse (need documents to prove), you will receive a grade of zero on that exam.

What is an algorithm?

▸ An *algorithm* is any well-defined computational procedure that takes some values as *input* and produces some values as *output*.

▸ Provide a step-by-step method for solving a computational problem. (Like cooking recipes)

▸ Not dependent on a particular programming language, machine, system, or compiler. (Unlike programs)

Example on sorting

▸ Problem: Sorting

  ▸ **Input:** sequence of n numbers $(a_1, a_2, \cdots, a_n)$

  ▸ **Output:** a permutation $(a'_1, a'_2, \cdots, a'_n)$ of the input instance such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$

  <u>For example:</u>

  Input: 53, 12, 35, 21, 59, 15
  Output: 12, 15, 21, 35, 53, 59

▸ Algorithms: Insertion sort, merge sort, quick sort, . . .

Issues in algorithm analysis & design

▸ Two fundamental issues: correctness and efficiency

▸ Steps to analyze and design an algorithm
  ▸ Formally define a problem
  ▸ Clearly describe an algorithm
  ▸ Prove correctness of the algorithm
  ▸ Analyze the efficiency of the algorithm

Efficiency of algorithms

▸ Goal

  ▸ To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirement, programmer's effort).

▸ Running time analysis

  ▸ Determine how running time increases as the size of the problem increases.

How do we compare algorithms?

▶ Need to define a number of objective measures

(1) Compare execution time?
Not good: time is specific to a particular computer !!

(2) Count the number of statements executed?
Not good: number of statements vary with the programming language as well as the style of the individual programmer.

▶ Ideal solution

  ▶ Express running time as a function of the input size n (i.e., *f(n)*).
  ▶ Compare different functions corresponding to running time.
  ▶ Such an analysis is independent of machine time, programming style, etc.

Random-Access Machine (RAM)

▶ A computational model

 ▶ All memory equally expensive to access

 ▶ No concurrent operations

 ▶ All reasonable instructions take unit time

  ▶ Except, of course, function calls

Example

- Associate a "cost" with each statement.

- Find the "total cost" by finding the total number of times each statement is executed.

Algorithm	Cost
sum = 0;	c_1
for(i=0; i<N; i++)	c_2
for(j=0; j<N; j++)	c_2
sum += arr[i][j];	c_3

$$c_1 + c_2 \times (N+1) + c_2 \times N \times (N+1) + c_3 \times N^2$$

Input size (number of elements in the input)

▶ How we characterize input size is problem-specific:

  ▶ Sorting: number of input items

  ▶ Multiplication: total number of bits

  ▶ Graph algorithms: number of nodes & edges

  ▶ ...

Common orders of magnitude

Table 2.1 The running times (rounded up) of different algorithms on inputs of increasing size, for a processor performing a million high-level instructions per second. In cases where the running time exceeds 10^{25} years, we simply record the algorithm as taking a very long time.

	n	$n \log_2 n$	n^2	n^3	1.5^n	2^n	$n!$
$n = 10$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	4 sec
$n = 30$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	< 1 sec	18 min	10^{25} years
$n = 50$	< 1 sec	< 1 sec	< 1 sec	< 1 sec	11 min	36 years	very long
$n = 100$	< 1 sec	< 1 sec	< 1 sec	1 sec	12,892 years	10^{17} years	very long
$n = 1,000$	< 1 sec	< 1 sec	1 sec	18 min	very long	very long	very long
$n = 10,000$	< 1 sec	< 1 sec	2 min	12 days	very long	very long	very long
$n = 100,000$	< 1 sec	2 sec	3 hours	32 years	very long	very long	very long
$n = 1,000,000$	1 sec	20 sec	12 days	31,710 years	very long	very long	very long

Types of analysis

▸ Worst case

  ▸ Provides an upper bound on running time

  ▸ An absolute guarantee that the algorithm would not run longer

▸ Best case

  ▸ Provides a lower bound on running time

  ▸ Input is the one for which the algorithm runs the fastest

$$Lower\ Bound \leq Running\ Time \leq Upper\ Bound$$

▸ Average case

  ▸ Provides a prediction about the running time

  ▸ Very useful, but treat with care: what is "average"?

    ▸ Random (equally likely) inputs

    ▸ Real-life inputs

Asymptotic Analysis

▸ To compare two algorithms with running times *f(n)* and *g(n),* we need a rough measure that characterizes how fast each function grows.

▸ Simplifications

  ▸ Ignore actual and abstract statement costs

  ▸ Order of growth: highest-order term is what counts

    ▸ Remember, we are doing asymptotic analysis

    ▸ As the input size grows larger, the high order term dominates

▸ For example: $5n^3 + 100n^2 + 10n + 50$ ~ n^3

Asymptotic notation: Big-Oh notation

▸ **Upper bounds.** T(n) is O(f(n)) if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \leq c \cdot f(n)$ for all $n \geq n_0$.

Ex. $T(n) = 32n^2 + 17n + 1$.
▸ T(n) is $O(n^2)$. ⟵ choose c=50, n_0=1
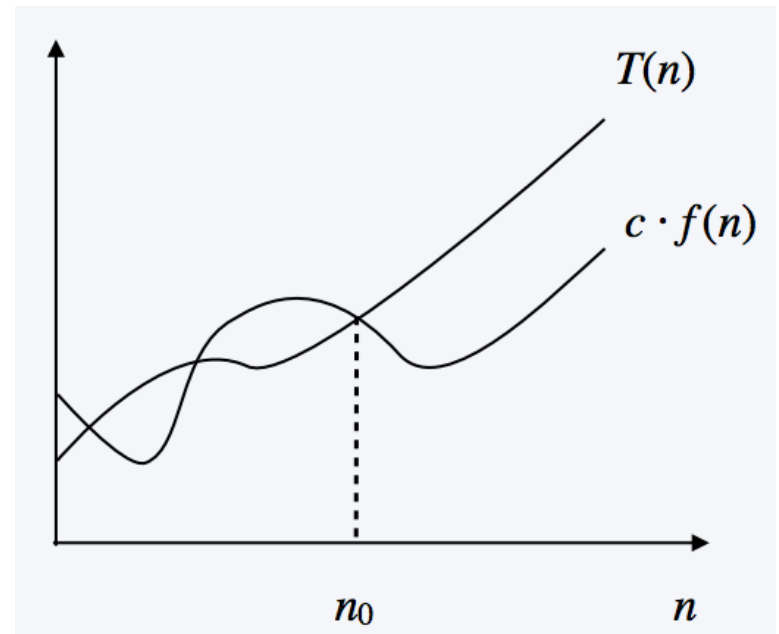▸ T(n) is also $O(n^3)$.
▸ T(n) is neither O(n) nor O(n log n).
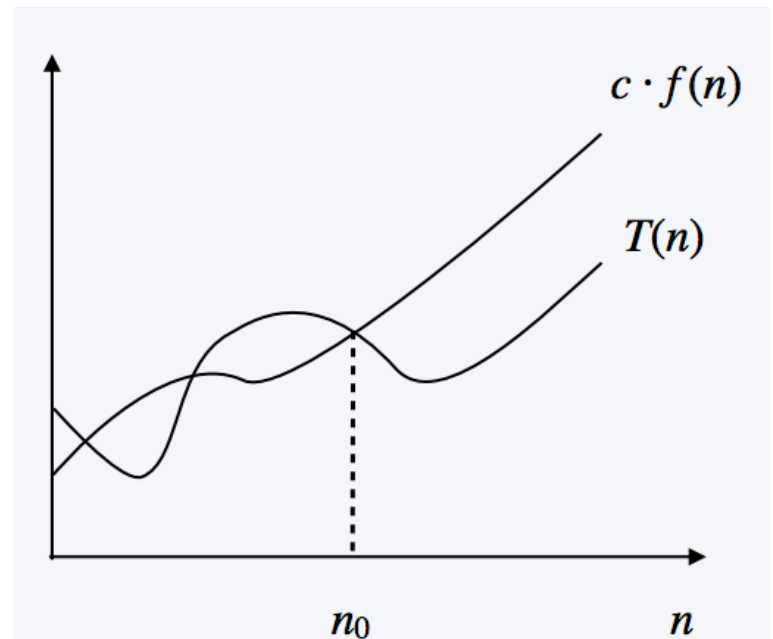
Asymptotic notation: Big-Omega notation

▸ **Lower bounds.** T(n) is $\Omega(f(n))$ if there exist constants $c > 0$ and $n_0 \geq 0$ such that $T(n) \geq c \cdot f(n)$ for all $n \geq n_0$.

Ex. $T(n) = 32n^2 + 17n + 1$.
▸ T(n) is $\Omega(n^2)$. ⟵ choose c=32,n_0=1
▸ T(n) is also $\Omega(n)$.
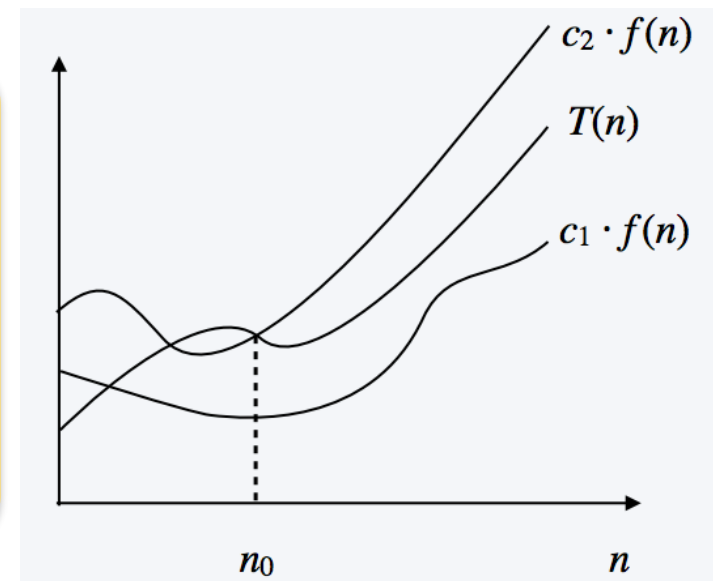▸ T(n) is neither $\Omega(n^3)$ nor $\Omega(n^3 \log n)$.

Asymptotic notation: Big-Theta notation

▸ **Tight bounds.** T(n) is $\Theta(f(n))$ if there exist constants $c_1 > 0$, $c_2 > 0$ and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.

Ex. $T(n) = 32n^2 + 17n + 1$.

▸ T(n) is $\Theta(n^2)$. ⟵ choose $c_1 = 32, c_2 = 50, n_0 = 1$

▸ T(n) is neither $\Theta(n)$ nor $\Theta(n^3)$.

Asymptotic notation: little-o, and little-ω

▸ **Little-o:** $T(n)$ is $o(f(n))$ if for any constant $c > 0$, there exists $n_0 \geq 0$ such that $T(n) < c \cdot f(n)$ for all $n \geq n_0$

▸ **Little-ω:** $T(n)$ is $\omega(f(n))$ if for any constant $c > 0$, there exists $n_0 \geq 0$ such that $T(n) > c \cdot f(n)$ for all $n \geq n_0$.

▸ Intuitively
  ▸ $o()$ is like $<$ $O()$ is like $\leq$
  ▸ $\omega()$ is like $>$ $\Omega()$ is like $\geq$
  ▸ $\Theta()$ is like $=$

Properties

- *Theorem:*

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

- Transitivity**:**

  - $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n)) \Rightarrow f(n) = \Theta(h(n))$

  - Same for O and Ω

- Reflexivity:

  - $f(n) = \Theta(f(n))$
  - Same for O and Ω

- Symmetry:

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

- Transpose symmetry:

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

What's next...

- Mathematical foundations
  - Review basic math functions
- Solving recurrence (Divide-and-Conquer)
  - Substitution, Iteration, Master method
  - Read CLRS Chapter 4