Lecture-1
Course logistics & Introduction & Asymptotic Run Time

CS325

Course Logistics

- Instructor:
  - Prof. Xiaoli Fern (Kelley 3073)

    xfern@eecs.oregonstate.edu

    Office hour: Monday Wednesday 10:50-11:35
- TAs:
  - Aayam K Shrestha shrestaa@oregonstate.edu
  - William Maxwell maxwellw@oregonstate.edu
  - Office hours will be available on Canvas

Assessments

- 5 in-class quizzes – 25%
- 3 implementation (group) assignments – 20%
- In-class participation – 5%
- Midterm – 25%
- Final – 25%
- Dates posted on class calendar - be aware of these dates, schedule your things around them
- There will be practice questions for you to use as study guide for the quizzes, and exams

How is this class run?

- We will loosely follow the following **book**:

Algorithms by Dasgupta, Papadimitriou and Vazirani.

Freely available chapter by chapter at the following link, just change the chap #

https://people.eecs.berkeley.edu/~vazirani/algorithms/chap0.pdf

- **Lectures**
  - Slides will be posted on canvas
  - Many will be skeleton slides with details to be filled in during lecture
  - Lecture often use examples different from the book to provide you a variety of examples --- you should also read the corresponding chapters in the book for more examples and perhaps a different explanation
- **In-class problem solving**
  - Concept warehouse – a free OSU online service for answering questions in class, you will need a device for internet access in class
  - Do not grade for correctness but participation
- **TA-held recitations**
  - Walk through example problems from the practice problem-set to help you prepare for the quizzes

Tips for success in this class

- Do your reading and suggested prep before class
- Attend lectures and recitations
- Due to limited class time, we may not have time in class to provide more than one example, or even finish the complete steps of one example --- work through examples on your own at home
- Don’t hesitate to ask for help from me or the TA
- Use the discussion board on Canvas for questions and get help from your fellow students
- Find the right partner(s) to work on your group implementation assignment

What you will learn in this class

- Asymptotic runtime analysis of algorithms
  - Big-Oh notation
  - Analyzing iterative and recursive algorithms (recurrence relation)
  - Solving recurrence relations
- Prove the correctness of algorithms
  - Some basic proof techniques: proof by induction, proof by contradiction
- Design efficient algorithms
  - Divide and conquer
  - Dynamic programming
  - Greedy algorithms
  - Linear programming
- Limits of computation:
  - Concept of reduction
  - P vs. NP

What is algorithm?

- Algorithm
  - A term coined to honor Al Khwarizmi, a Persian mathematician who wrote the first foundational book on algebra
  - In his book, he moved from solving specific problems to a more general way of solving problems
  - Introduced **precise, unambiguous, mechanical, and correct procedures for solving general problems** – algorithms

Why studying algorithms

- Important for all branches of computer science (and other as well)
  - Computer Networking heavily rely on graph algorithms
  - Bioinformatics builds on dynamic programming algorithms
  - Cryptography – number theoretic algorithms
  - ….
- And it is extremely fun, challenging but fun! And it will help you for your job interview!!!
- Two fundamental and vital questions
  - Is the algorithm correct?
  - Is the algorithm efficient? Or, can we do better?

Efficiency: Run time analysis

- How long will an algorithm take to run may depend on a lot of things
  - Processor
  - Language
  - Implementation …
- We will abstract away from these details and study the run time at the **asymptotic level**

A simple example: Insertion sort

- Given an array A of n numbers, build the sorted array one element at a time

1. for $i \leftarrow 1$ *to* n
2. $\quad x \leftarrow A[i]$
3. $\quad j \leftarrow i - 1$
4. $\quad$ while $j > 0$ and $A[j] > x$
5. $\quad\quad A[j+1] \leftarrow A[j]$
6. $\quad\quad j \leftarrow j - 1$
7. $\quad$ end while
8. $\quad A[j+1] \leftarrow x$
9. end for

6 5 3 1 8 7 2 4

Runtime of insertion sort

1. for $i \leftarrow 1$ *to* n
2. $\quad x \leftarrow A[i]$
3. $\quad j \leftarrow i - 1$
4. $\quad$ while $j > 0$ and $A[j] > x$
5. $\quad\quad A[j+1] \leftarrow A[j]$
6. $\quad\quad j \leftarrow j - 1$
7. $\quad$ end while
8. $\quad A[j+1] \leftarrow x$
9. end for

Line 4-7

- Best case (already sorted): 1
- Worst case (reverse order): $i - 1$ for $A[i]$

The specific run time depends on the input

Runtime: worst-case

- We focus on the worst-case analysis: the run time of an algorithm gives us a running time bound that holds for every possible input.
  - If an algorithm runs in polynomial time for one input and linear time for all other inputs, the runtime is polynomial
- Appropriate for general purpose analysis
- “Average-case” analysis ?
  - Require more heavy machinery about probabilities– much harder
  - Requires a good understanding of the domain – what would average input look like?
- “best-case” analysis? Sorry, that is just wishful thinking …

Worst-case Runtime of insertion sort?

1. for $i \leftarrow 1$ *to* n
2. $\quad x \leftarrow A[i]$
3. $\quad j \leftarrow i - 1$
4. $\quad$ while $j > 0$ and $A[j] > x$
5. $\quad\quad A[j+1] \leftarrow A[j]$
6. $\quad\quad j \leftarrow j - 1$
7. $\quad$ end while
8. $\quad A[j+1] \leftarrow x$
9. end for

Line 4-7

- Worst case (reverse order): $i - 1$ for $A[i]$

Simplification 1:
We don't care about the exact time each step takes, we only care about the number of basic operations (e.g., comparison, assignment etc)

Running time is expressed by counting the number of basic computer steps, as a function of the size of the input.

Simplification 2: Constants and lower order terms don't matter

- Constant factors

  $2n$ vs. $3n$

- Lower order terms

  $2n^2 + 2n$ vs. $3n^2$

Asymptotic analysis

- Focus on running time for very large input size n

- Why: with large n, we start to see the difference between computationally feasible vs. not feasible

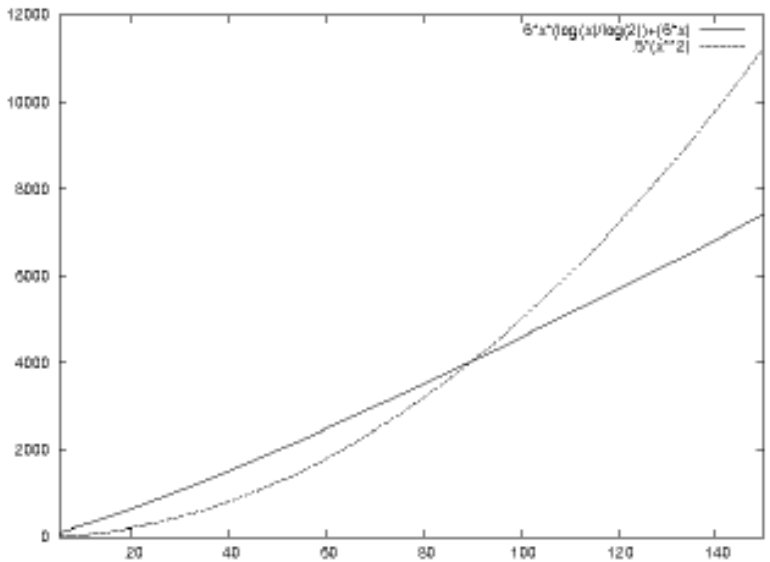
Asymptotic growth

- Example:

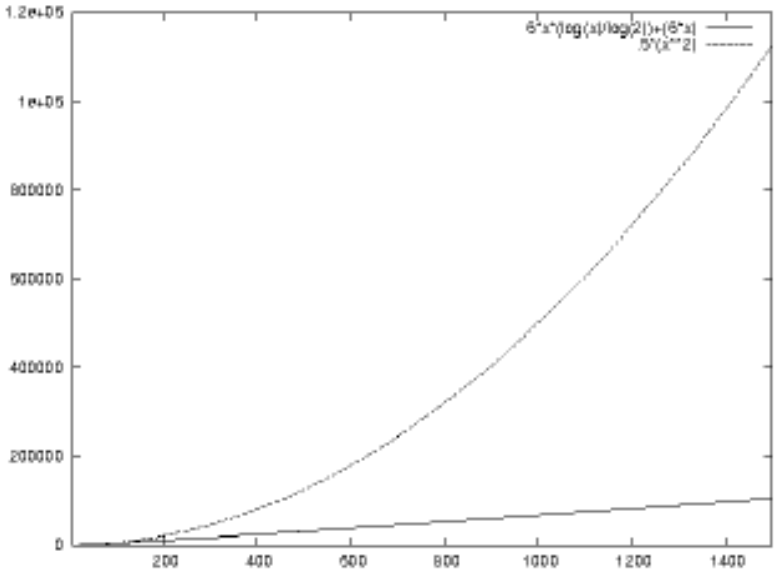
$6n \log_2 n + 6n$ versus $\frac{1}{2} n^2$

Merge sort

Insertion sort



Small n(1-150)



Larger n (200-1400)

Asymptotic Analysis: Big-Oh notation

English definition:

Let $f(n)$ and $g(n)$ be two functions

We say that $f(n) = O\big(g(n)\big)$ if eventually (**for all sufficiently large $\boldsymbol{n}$**), $f(n)$ is bounded above by a constant multiple of $g(n)$

Intuitive Meaning: $f(n)$ grows no faster (asymptotically no worse) than $g(n)$

Big-Oh: formal definition

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

there exist constants c, n_0 such that

$$f(n) \leq cg(n)$$

for all $n \geq n_0$

Note: c and n_0 cannot depend on n

Example #1

- $T(n) = 4n^k + 5n^{k-1}$
- Claim: $T(n) = O\left(n^k\right)$
- Proof:

Example #2

- $T(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0$
- Claim: $T(n) = O(n^k)$
- Proof:

Example #3

- Claim: for every $k \geq 1$, n^k is not $O(n^{k-1})$
- Proof by contradiction:

Big-Omega (Ω) notation

$f(n) = \Omega\big(g(n)\big)$ if and only if there exist constants c, n_0 such that

$$f(n) \geq cg(n)$$

for all $n \geq n_0$

Intuitive Meaning: $f(n)$ grows no slower (asymptotically no better) than $g(n)$

Theta (Θ) notation

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

there exist constants c_1, c_2, n_0 such that

$$c_1 g(n) \leq f(n) \leq c_2 g(n)$$

for all $n \geq n_0$

Or equivalently $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega\big(g(n)\big)$

Intuitive Meaning: $f(n)$ grows the same (asymptotically equivalent) as $g(n)$

Useful Limits

If $\lim_{n\to\infty} \frac{f(n)}{g(n)}$

- $! = \infty$ (bounded): $f(n) = O\big(g(n)\big)$
- $! = 0$ or ∞ (bounded, nonzero): $f(n) = \theta\big(g(n)\big)$
- $! = 0$ (nonzero): $f(n) = \Omega(g(n))$

Examples

$$f(n) = a_k n^k + a_{k-1} n^{k-1} + \cdots + a_0$$

$$g_1(n) = n^k, \qquad g_2(n) = n^{k-1}$$

$$\lim_{n\to\infty} \frac{f(n)}{g_1(n)} = \lim_{n\to\infty} \left(a_k + \frac{a_{k-1}}{n} + \cdots + \frac{a_0}{n^k} \right) = a_k$$

$$f = O(g_1)? \quad f = \theta(g_1)? \quad f = \Omega(g_1)?$$

$$\lim_{n\to\infty} \frac{f(n)}{g_2(n)} = \lim_{n\to\infty} \left(a_k n + a_{k-1} + \cdots + \frac{a_0}{n^{k-1}} \right) = \infty$$

$$f = O(g_2)? \quad f = \theta(g_2)? \quad f = \Omega(g_2)?$$

Quick in-class exercise

$T(n) = 2n^2 + 3n$, which of the following statements are true? (check all that apply)

- ❑ $T(n) = O(n)$
- ❑ $T(n) = O(n^3)$
- ❑ $T(n) = \Omega(n)$
- ❑ $T(n) = \Theta(n^2)$

Quick in-class practice

$f(n) = 2^{n+10}$ $g(n) = 2^n$

Which of the followings are correct?

- ❑ $f(n) = O\big(g(n)\big)$
- ❑ $f(n) = \Theta\big(g(n)\big)$
- ❑ $f(n) = \Omega\big(g(n)\big)$

Question

- We said that in asymptotic analysis we don't care about constants, but not all constants are unimportant
- For example: n^2 vs. n^3, 2^n vs. 3^n

- Which constants in the following expressions do we care about in asymptotic run time?

$$2(n+1)^3, \log_4 n, \log(n^5), (\log n)^6, 8^{7(\log_9 n)}$$

Useful facts about logs and exponentials

$$a^{x+y} = a^x \cdot a^y$$

$$a^{2x} = (a^x)^2$$

$$\log_2 a^x = x \log_2 a$$

$$\log(a \cdot b) = \log a + \log b$$

$$\log \frac{a}{b} = \log a - \log b$$

$$\log_a x = \log_a b \log_b x$$

Common Efficiency Class

Class	Name	
1	constant	No reasonable examples, most cases infinite input size requires infinite run time
$\log n$	Logarithmic	Each operation reduces the problem size by half, Must not look at the whole input, or a fraction of the input, otherwise will be linear
n	linear	Algorithms that scans a list of n items (sequential search)
$n \log n$	linearithmic	Many D&C algorithms e.g., merge sort
n^2	quadratic	Double embedded loops, insertion sort
n^3	cubic	Three embedded loops, some linear algebra algo.
2^n	exponential	Typical for algo that generates all subsets of a n element set.
$n!$	factorial	Typical for algo that generates all permutations of a n-element set

Summary of 1 st lecture

- Run time of an algorithm is a function of the input size
  - Input size : roughly viewed as the number of bits for representing the input
  - Sorting: n = the size of the array
  - Addition: input size = the number of bits for representing the numbers, for large numbers ($\log n$)
- The run time may depend on the input: Best case, worst case and average – worst case is what we care about
- Asymptotic complexity – order of growth
- O- upper bound. E.g., insertion sort: $O(n^2), O(n^3)$..
- Ω- lower bound. E.g., insertion sort: $\Omega(n^2), \Omega(n), \Omega(\log n)$…
- Θ – tight bound. E.g., insertion sort: $\theta(n^2)$, $\theta(n^2 + 4n)$ …