Lecture-1
Course logistics & Introduction & Asymptotic Run Time

CS325

Course Logistics

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

    xfern@eecs.oregonstate.edu

    Office hour: Tuesday Thursday after class (3:20-4)
- TAs:
  - Juneki Hong (.49TA)
  - Hamed Shahbazi (.24TA)
  - Souti Chattopadhyay (.24TA)
  - Juan Liu (.24TA)
  - Office hours will be available on Canvas

Assessments

- 5 in-class quizzes – 25%
- 3 implementation (group) assignments – 25%
- 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

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

Consider the following segment of pseudo-code (given input n):

```
sum ← 0
for i = 1 to n
    for j = i to n
        sum ← sum + i x j
```

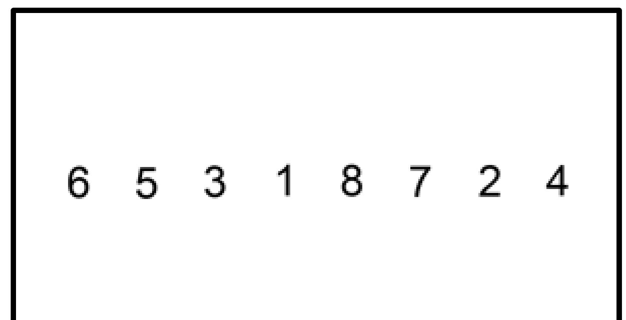
What does it compute?

Assuming basic operations like addition, multiplication takes unit time, what is the run time of the above code?

Another example: Insertion sort

- Given an array A of n numbers, build the sorted array one element at a time (similar to how we organize a hand of cards)

for $i \leftarrow 1\ to\ n$
    $x \leftarrow A[i]$
    $j \leftarrow i - 1$
    while $j > 0$ and $A[j] > x$
        $A[j+1] \leftarrow A[j]$
        $j \leftarrow j - 1$
    end while
    $A[j+1] \leftarrow x$
end for



Runtime of insertion sort

```
1. for i ← 1 to n
2.     x ← A[i]
3.     j ← i − 1
4.     while j > 0 and A[j] > x
5.         A[j + 1] ← A[j]
6.         j ← j − 1
7.     end while
8.     A[j + 1] ← x
9. end for
```

Line 4-7

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

Runtime depends on input

- In this class, we focus on the worst-case analysis: gives us a running time bound that holds for every possible input.
- Particularly appropriate for general purpose analysis
- "Average-case" analysis ?
  - Require more heavy machinery – much harder
  - Requires a good understanding of the domain – what would average input look like?
- "best-case" analysis? Sorry, that is just wishful thinking …

Constants and lower order terms don't matter

- Constant factors

$$2n \text{ vs. } 3n$$

- Lower order terms

$$2n^2 + 2n \text{ vs. } 3n^2$$

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

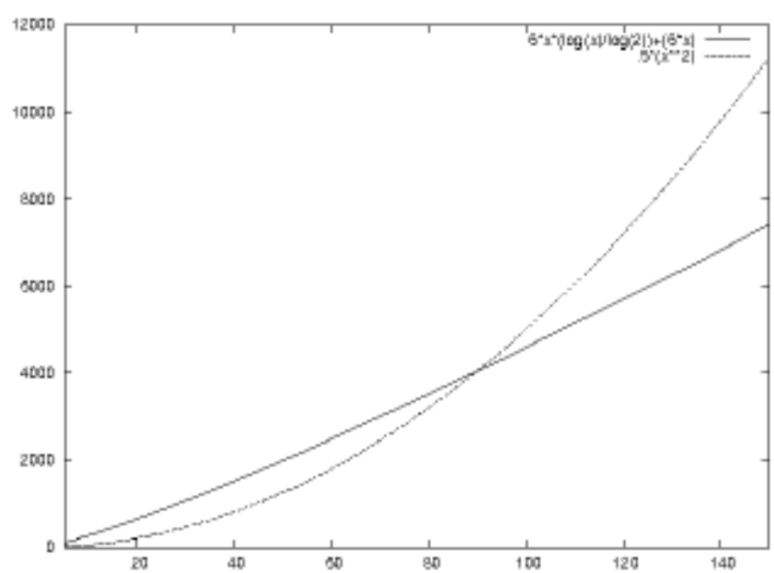
Asymptotic growth

- Example:

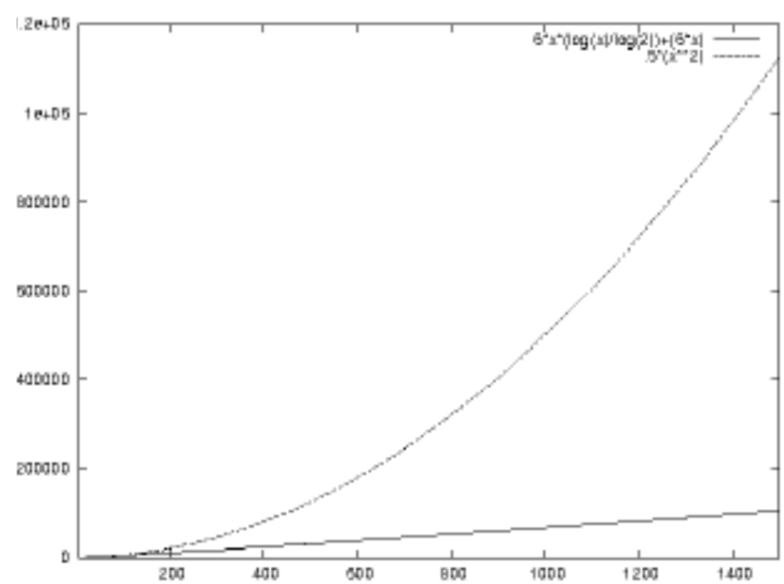
$6n\log_2 n \ + 6n$ versus $\frac{1}{2}n^2$

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

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

Example #2

- 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

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

for all $n \ge n_0$

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

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} 0 & f(n) = O(g(n)) \\ \mathrm{c} & f(n) = \Theta\big(g(n)\big) \\ \infty & f(n) = \Omega\big(g(n)\big) \end{cases}$$

where c is a finite nonzero constant

Note that: if $f(n) = \Theta\big(g(n)\big)$, it is also true that $f(n) = O\big(g(n)\big)$ and $f(n) = \Omega(g(n))$

Example

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

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

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

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

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

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

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

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

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: $\theta(n^2)$, $\theta(n^2 + 4n)$ …