

Introduction and Course Outline

Lecture 1

Why Study Algorithms?

- Important for all other branches of computer science (networks, database, bioinformatics, public key cryptography, computer graphics)
- Plays key role in modern technological innovation
- Provides novel “lens” on processes outside of computer science and technology
 - Quantum mechanics, economic markets, evolution
- Challenging (i.e., good for brain)
- fun

Why Study Algorithms?

- plays a key role in modern technological innovation

“Everyone knows Moore’s Law a prediction made in 1965 by Intel cofounder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years.... in many areas, performance gains due to improvements in algorithms have vastly exceeded even the dramatic performance gains due to increased processor speed.”

- Excerpt from *Report to the President and Congress: Designing a Digital Future*, December 2010 (page 71).

Algorithms as technology

- Insertion sort runs in $O(n^2)$ time
- Merge sort runs in $O(n \lg n)$ time
- Suppose computer A executes 1 billion instructions per second
- Suppose computer B executes 10 million instructions per second
- Computer A is 100 times faster than Computer B
- Suppose input size = 1 million = 10^6
- To make difference more dramatic, suppose insertion sort is coded by a very good programmer so leading constants are small = $2n^2$
- Suppose merge sort is coded by average programmer so constants are large = $50 n \lg n$

Algorithms as technology

- Run insertion sort on faster computer A (1 billion instructions per sec)
- Run merge sort on slower computer B (10 million instructions per second)
- Insertion sort number of instructions = $2n^2$
- Merge sort number of instructions = $50 n \lg n$
- Input size = $n = 10^6$
- What is running time in seconds for both programs?

Algorithms as technology

- Running time for insertion sort = $2 * (10^6)^2 / 10^9$ instructions per second
- = 2000 seconds
- Running time for merge sort = $50 * (10^6 \lg 10^6) / 10^7$ instructions per second
- = 100 seconds

- Merge sort runs 20 times faster than insertion sort even if we run it on 100 times slower computer

Course Topics

- Vocabulary for design and analysis of algorithms
- Divide and conquer design paradigm
- Randomization in algorithms
- Dynamic programming design paradigm
- Greedy algorithms design paradigm
- Graph algorithms

Skills you will learn

- Become a better programmer
- Sharpen your mathematical and analytical skills
- Start thinking algorithmically
- Literacy with computer science's “greatest hits”
- Ace your technical interviews

The Algorithm Designer's Mantra

- “Perhaps the most important principle for the good algorithm designer is to refuse to be content”
 - Aho, Hopcroft and Ullman, The Design and Analysis of Computer Algorithms, 1974

Can we do better?
[than the obvious method]

Analyzing Algorithms

- Has come to mean predicting the resources that the algorithm requires
- Usually computational time is resource of primary importance
- Aims to identify best choice among several alternate algorithms
- Requires an agreed-upon “model” of computation
- Shall use a generic, one-processor, random-access machine (RAM) model of computation

Random-Access Machine

- Instructions are executed one after another (no concurrency)
- Admits commonly found instructions in “real” computers, data movement operations, control mechanism
- Uses common data types (integer and float)

Algorithm Analysis

- Time resource requirement depends on ***input size***
- ***Input size*** depends on problem being studied; frequently, this is the number of items in the input
- Running time: number of primitive operations or “steps” executed for an input
- Assume constant amount of time for each line of pseudocode

Algorithms Analysis Guiding Principles

Guiding Principle #1

- “worst – case analysis” : our running time bound holds for every input of length n .
 - Particularly appropriate for “general--purpose” routines
 - As Opposed to
 - --“average--case” analysis
 - --benchmarks
- } Requires domain Knowledge
- BONUS : worst case usually easier to analyze.

Guiding Principle #2

- Won't pay much attention to constant factors, lower-order terms

Justifications

1. Way easier
2. Constants depend on architecture / compiler / programmer
3. Lose very little predictive power
(as we'll see)

Guiding Principle #3

- Asymptotic Analysis : focus on running time for large input sizes n

Eg : $6n \lg n$ “better than” $2n^2$

Merge sort

Insertion sort

Justification: Only big problems are interesting!

What Is a “Fast” Algorithm?

This Course : adopt these three biases as guiding principles

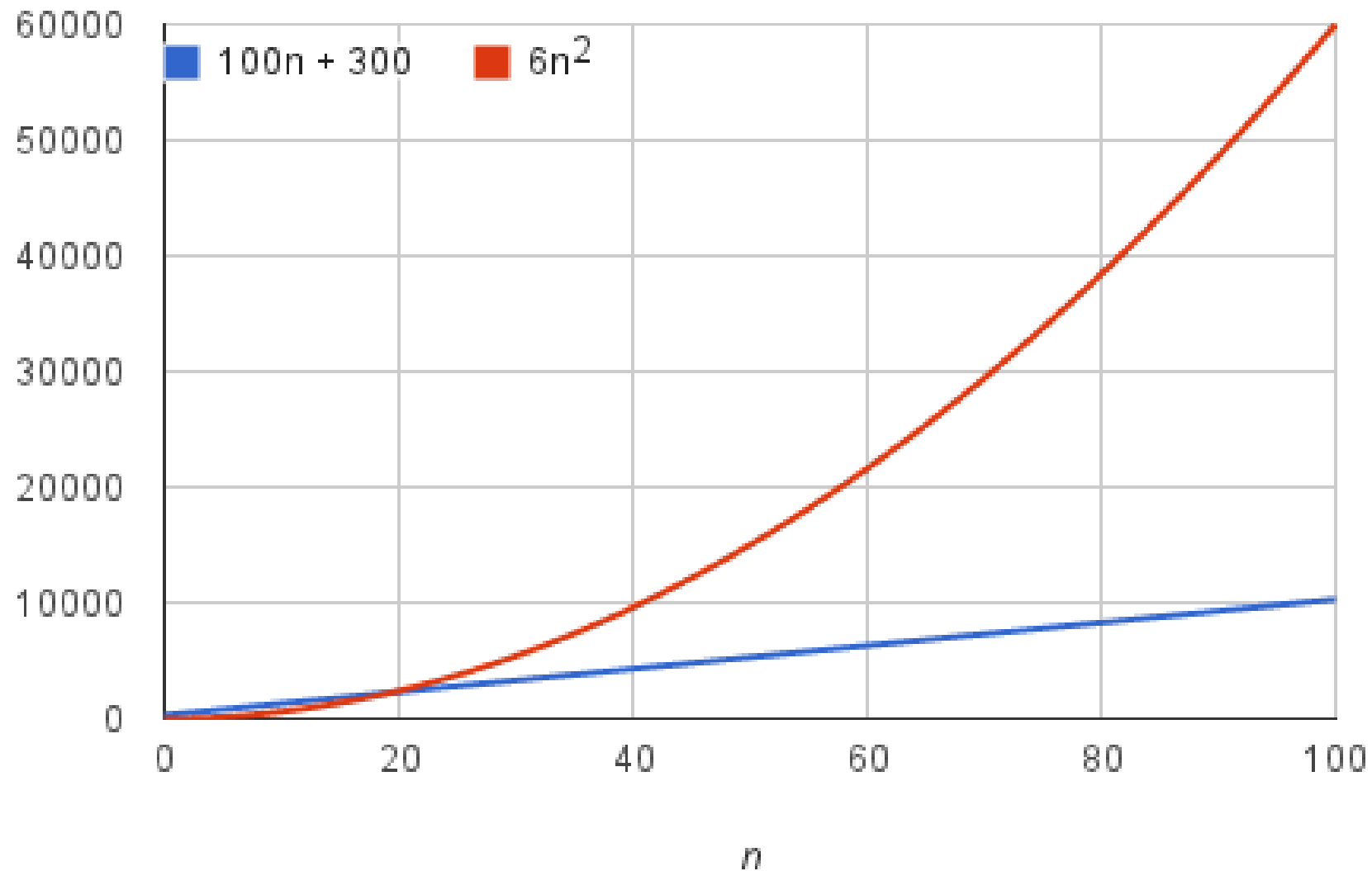
fast algorithm \approx worst--case running time
grows slowly with input size

Usually : want as close to linear ($O(n)$) as possible

Rate of Growth

- An algorithm running on an input of size n , takes $6n^2+100n+300$ machine instructions.
- The $6n^2$ term becomes larger than the remaining terms, $100n+300$, once n becomes large enough

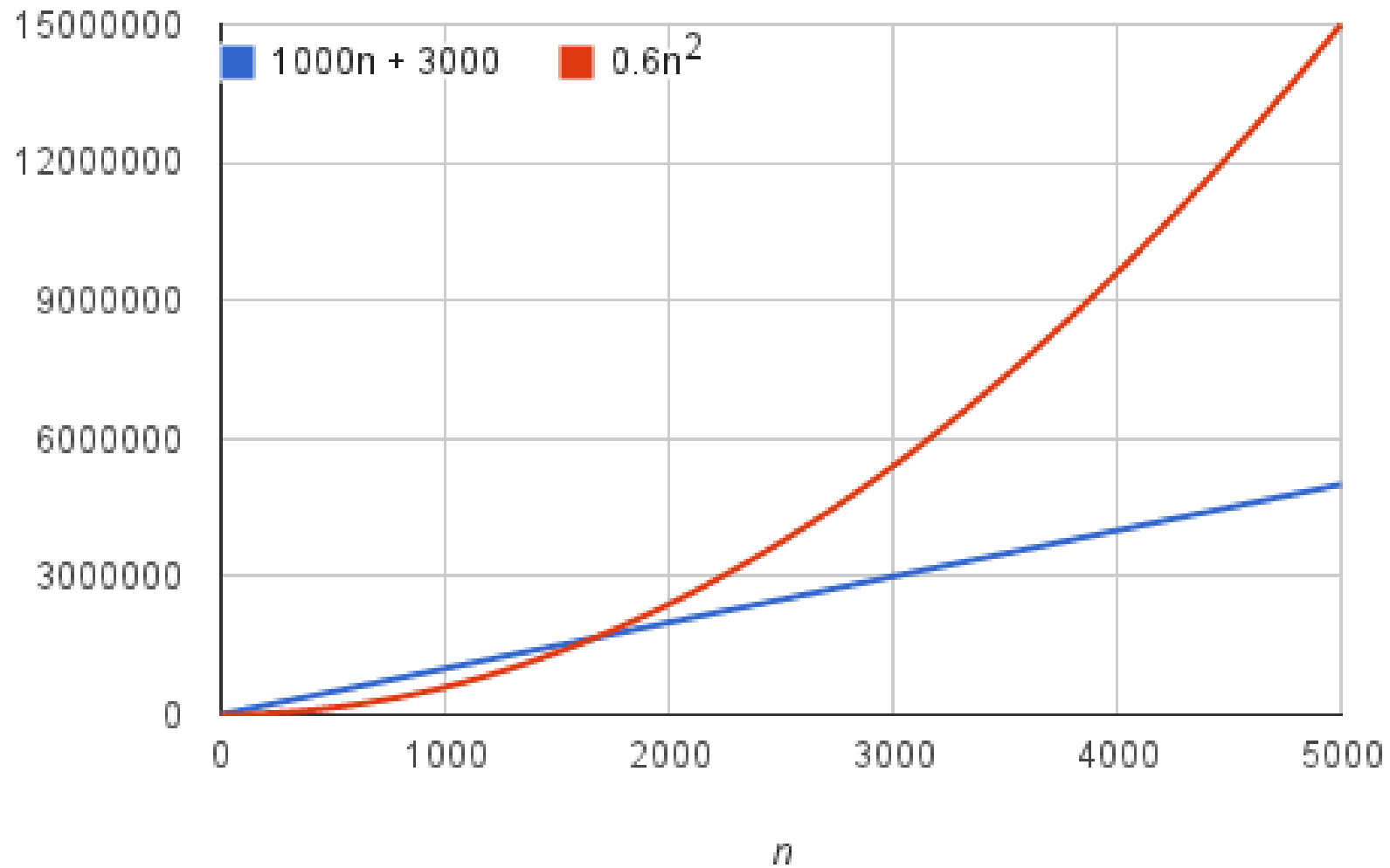
Comparison of $6n^2$ and $100n+300$



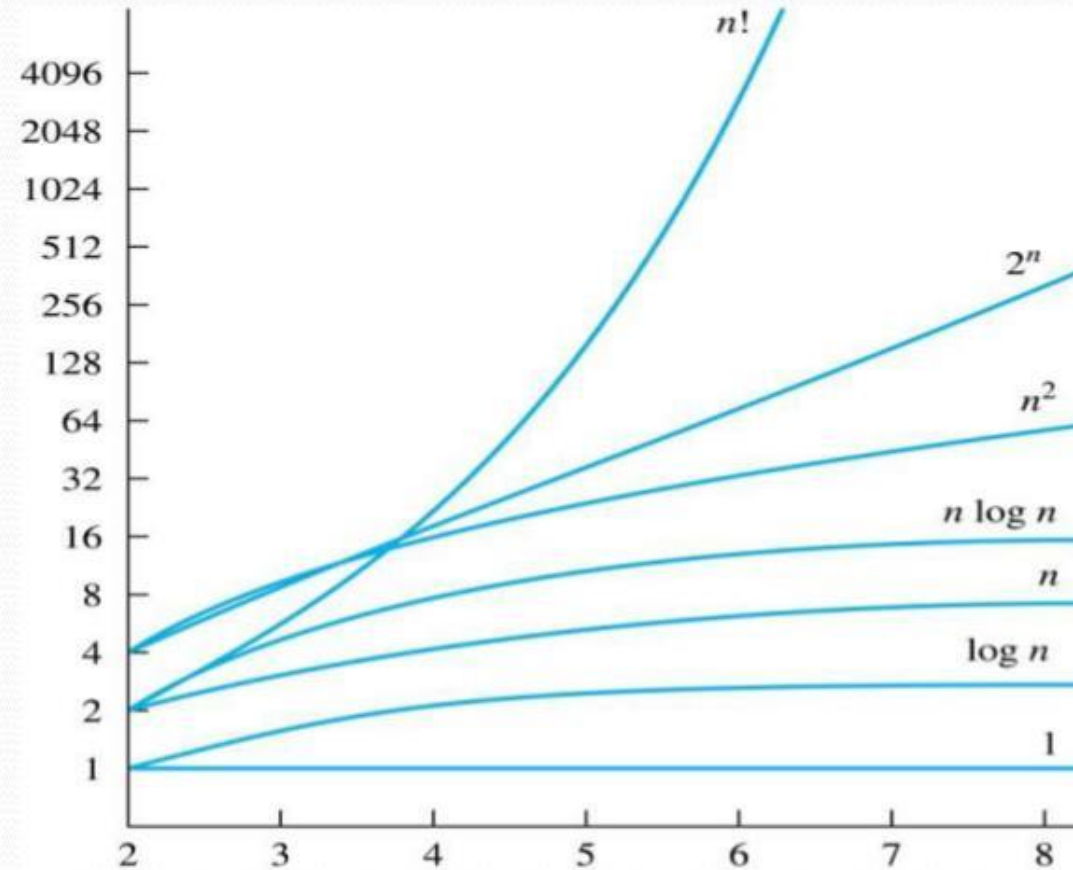
Rate of Growth

- It doesn't really matter what coefficients we use; as long as the running time is $an^2 + bn + c$ for some numbers $a > 0$, b , and c , there will always be a value of n for which an^2 is greater than $bn + c$, and this difference increases as n increases.

Comparison of $0.6n^2$ and $1000n+3000$



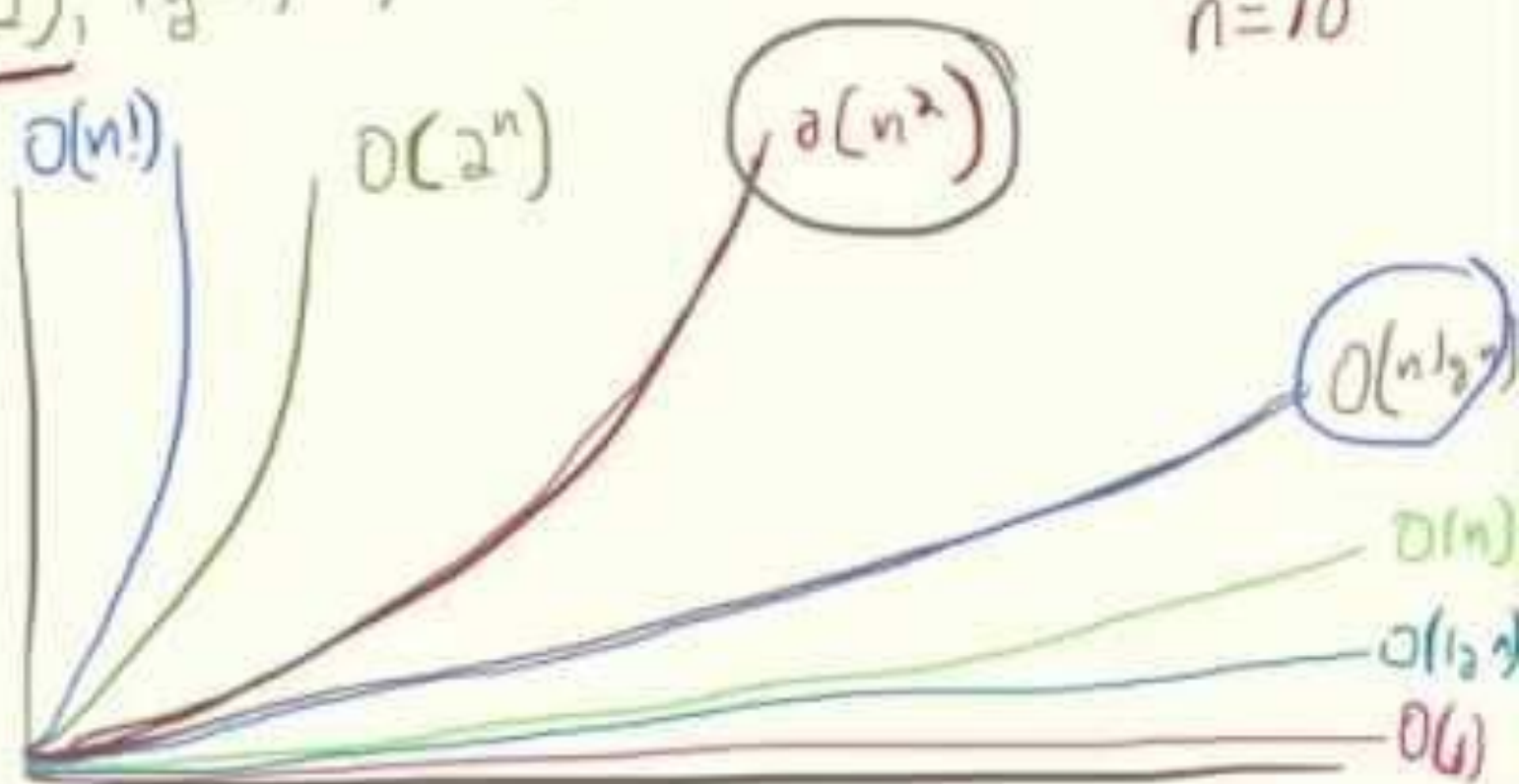
Display of Growth of Functions



Note the difference in behavior of functions as n gets larger

Growth of Functions

1 3.32 10 33.22 100 1024 $3M/M$
 $O(1)$, $\lg n$, n , $n \lg n$, n^2 , 2^n , $n!$
 $n=10$



How big is running time of exponential time algorithm ?