

Asymptotic Analysis

CHANG Yi-Jun Steven HALIM PRASHANT Nalini Vasudevan

School of Computing
National University of Singapore

CS3230 Lec01b; Tue, 12 Aug 2025

Overview

Introduction

Problem-Solving Example: Fibonacci

Model of Computation: RAM

Asymptotic Analysis

Big O (upper bound) - from CS2040S

New notation Ω (lower bound)

New notation Θ (tight bound)

New notations: Little-o and ω

Taking Limits

Appendixes

Math Refresher

Conditional Probability

Random Variable

Algorithm

Algorithm

“A sequence of unambiguous and executable instructions for solving a problem (given a valid input, obtain a valid output)”

Let's elaborate:

- ▶ What are the valid inputs?
- ▶ What is the meaning of unambiguous instructions?
- ▶ What is the meaning of executable instructions?
- ▶ Are all algorithms deterministic?
- ▶ Are all algorithms terminate?

Pseudocode

We can give an algorithm already written in a particular programming language, pros and cons:

- ▶ Unambiguous
- ▶ Clear
- ▶ Quite tedious
- ▶ Harder to understand

Alternative: Pseudocode (we will use this going forward)

- ▶ Slightly informal
- ▶ Still precise enough to understand exactly what instructions are, and how to implement it in some programming language

An Example

In Python (source code)

```
A = [(1, 2, 3), (4, 5, 6)]  
[*zip(*A)]
```

In Pseudocode:

Do you know what is this?

Some Properties of Good Algorithms

There can be many possible algorithms for solving a problem

Given the choices, we prefer:

- ▶ Correctness (the most important property)
- ▶ Efficiency (time/space/resources)
- ▶ Generality: Applicable to a wide range on inputs and not dependent on a particular computer/device
- ▶ Usability as a 'subroutine' for other problems
- ▶ Simplicity: so that it is easy to code, understand, debug, etc
- ▶ Well documented (easy to understand and to extend it)

Some objectives may have trade-offs: simplicity vs efficiency

Design and Analysis of Algorithms

Designing an algorithm is both science and art
You need to know the relevant techniques
But you also need creativity, intuition, perseverance

Paradigms

- ▶ Complete Search (for example, using brute force, backtracking, branch and bound)
- ▶ Divide and Conquer (D&C)
- ▶ Deterministic versus non-deterministic strategies
- ▶ Dynamic Programming (DP)
- ▶ Greedy Algorithm
- ▶ Iterative Improvement (details in future courses, e.g., CS4234)

Problem-Solving

The general steps:

1. Understand the problem
2. Design a method to solve the problem
3. Convert it into an algorithm/pseudocode
4. Choose data structures
5. Prove correctness of the algorithm
6. Analyze the complexity of the algorithm
(time/space/resources needed)
7. PS: Implement that correct and efficient algorithm

Fibonacci Numbers

- ▶ $Fib(0) = 0$
- ▶ $Fib(1) = 1$
- ▶ For $n > 1$, $Fib(n) = Fib(n - 1) + Fib(n - 2)$
- ▶ First 10 terms: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

Problem: Given n as input, compute $Fib(n)$

We will look at two algorithms:

- ▶ Recursive algorithm
- ▶ Iterative algorithm

PS: Yes, there are other (faster) algorithms

Recursive algorithm to compute $Fib(n)$

```
define Fib(n)
  if n <= 1
    return n
  else
    return Fib(n-1)+Fib(n-2)
```

Simple, direct recursive implementation from the $Fib(n)$ definition

Iterative algorithm to compute $Fib(n)$

```
define IFib(n)
  if n <= 1
    return n
  else
    prev2 = 0
    prev1 = 1
    for i = 2 to n
      temp = prev1
      prev1 = prev1+prev2
      prev2 = temp
  return prev1
```

Analysis of an Algorithm

We analyze the resources needed by an algorithm:

- ▶ Time – in this course, we will mostly concentrate on time
- ▶ Space – in this course, we assume all data fits in memory

Sometimes, we do trade-offs:

- ▶ If space is not an issue, most of the time, we sacrifice (or use more) space to gain faster time
- ▶ For some applications (e.g., Big Data), we may have to sacrifice time so that we are able to process the data

Actual time needed to run an algorithm depends on the machine used, and this is not easy to calculate/measure

Model of Computation: RAM

Random-Access Machine (RAM) model is simple and close to how real computers work:

- ▶ Each instruction takes a constant amount of time: fetch the instruction, execute, store back the results in the memory
- ▶ We count the number of basic instructions needed
- ▶ The time complexity is based on input size (more details soon)

RAM, Continued

- ▶ Word is basic unit of memory
In this course, you can usually assume each number (or relevant item) can be stored in one word
- ▶ RAM is an array of words, storing instructions and data
It takes one unit of time to access any word (this is important)
- ▶ Each arithmetic or logical operation ($+$, $-$, $*$, $/$, mod, AND, OR, NOT, etc) takes a constant amount of time (notice that exponent operation is not constant – see D&C lecture later)
- ▶ Details of word size and different time taken by different instructions are important, but **USUALLY** do not have a large impact; so we usually ignore it, unless it makes a difference
- ▶ We need to be careful: when numbers are very large (and thus cannot fit in one word), the complexity depends on number of bits/words needed to store the number

For our $Fib(n)$ and $IFib(n)$ analysis

For large computation of $Fib(n)$,
the resulting number can be very large

To address the above, one can consider computing the Fibonacci numbers modulo some m (for example $2^{wordsize}$)

We omit this detail in our first analysis to simplify discussion

Analysis of recursive algorithm to compute $Fib(n)$

```
define Fib(n)
  if n <= 1
    return n
  else
    return Fib(n-1)+Fib(n-2)
```

See the recursion tree@VisuAlgo
(which is a big tree for large n)

This 'brute force' $Fib(n)$ is bad

Let $T(n)$ be the number of operations done by $Fib(n)$

$T(0) = T(1) = 2$
(if+return)

For $n \geq 2$, $T(n) =$
 $T(n-1) + T(n-2) + 8$
(if+else+two calls+two minuses+add+return)

So $T(n) \geq Fib(n)$

We can show that
 $Fib(n) \geq 2^{\frac{n-2}{2}}$ (How?)

$T(n)$ is exponential in n

Analysis of iterative algorithm to compute $Fib(n)$

```
define IFib(n)
  if n <= 1
    return n
  else
    prev2 = 0
    prev1 = 1
    for i = 2 to n
      temp = prev1
      prev1 = prev1+prev2
      prev2 = temp
    return prev1
```

See the recursion DAG@VisuAlgo
(small and proportional to n)

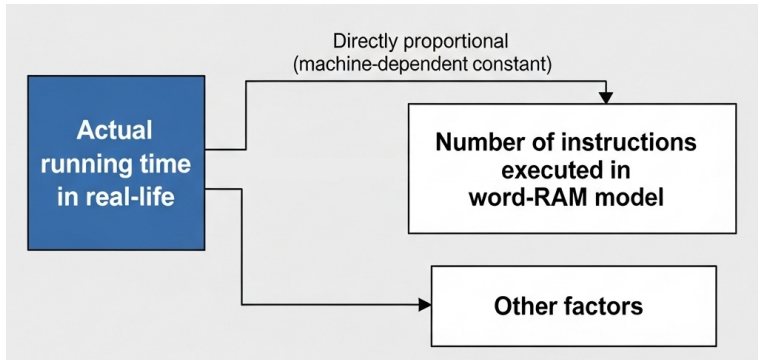
This is 'Dynamic Programming'
(DP) (to be revisited later)

For $n \geq 2$,
 $T(n) \approx 4 + (n - 1) * 5 + 1$
(if+else+two assignments
+ $(n - 1)$ iterations,
each takes ≈ 6 steps
(assignment, addition,
assignment, assignment,
increment, comparison)
+return)

So $T(n) \approx 6n$, linear in n

This is much faster than
the recursive version that
runs exponential in n

Actual Running Time



Running Time of an Algorithm

- ▶ We often give the running time in terms of the size of the input (usually parameter n)
- ▶ Size of the input can be the number of items (e.g., sorting n integers) or length of inputs coded in binary (e.g., integer n in $Fib(n)$ requires $\log n$ bits encoding – details in the second half)
- ▶ We usually perform these analysis:
 - ▶ Worst-case analysis: $T(n)$ is the maximum time needed for any input of size (at most) n
 - ▶ Average-case analysis: $T(n)$ is the expected time taken over all inputs of size n ; either all inputs are equally probable, or we know the probability distribution over the inputs of size n
 - ▶ We usually do not consider best-case analysis, as inputs that trigger best-case are usually not the typical ones
- ▶ It is difficult to compute the exact number of operations (as seen earlier), thus we often give upper bounds instead

Question 2 at VisuAlgo Online Quiz

Which algorithm is more efficient?

$$T1(n) = 100n + 1000$$

$$T2(n) = n^2 + 5$$

Asymptotic Analysis

Why we do not measure the actual runtime:

- ▶ Different machines have different speeds, i.e., new gaming desktop is fast vs 10-years old laptop is slow
- ▶ Different programming languages have different runtimes, i.e., C++ is fast vs Python is slow

We prefer to do asymptotic analysis:

- ▶ For large inputs, how does the runtime behave?
- ▶ Comparison of algorithms is based on the asymptotic analysis
- ▶ We often ignore lower terms and constant multiplicative factors in the asymptotic analysis

Most common asymptotic notation: Big O (upper bound)

For the following discussion on asymptotics, assume f and g are functions of one parameter n

$f \in O(g)$ if there exists constant $c > 0$ and $n_0 > 0$ such that for all $n \geq n_0 : 0 \leq f(n) \leq c \cdot g(n)$

Interpretation: g is an upper bound on f

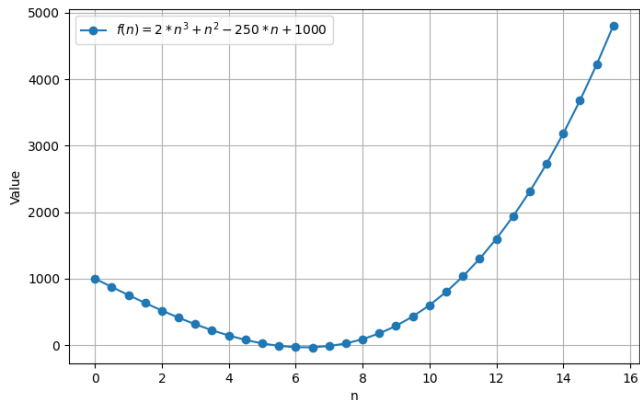
$O(g) = \{f: \text{there exists constant } c > 0 \text{ and } n_0 > 0 \text{ such that for all } n \geq n_0, 0 \leq f(n) \leq c \cdot g(n)\}$

We sometimes also write $f = O(g)$, though not 100% correct

We frequently write $f(n) = O(g(n))$, though technically, n should not have been used (there can be more than one parameter)

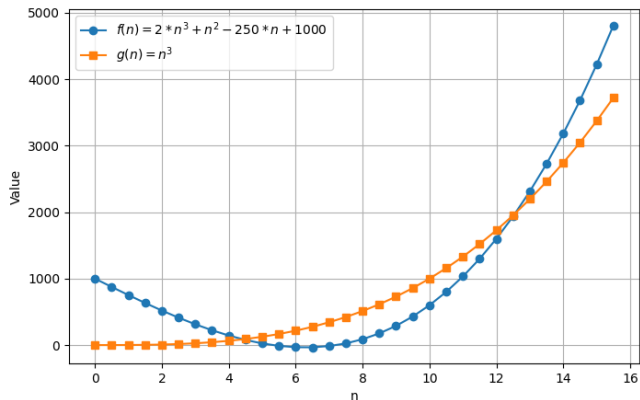
Similarly for other asymptotic notations; PS: we **accept** all versions

Visual Explanation of Big O notation (1)



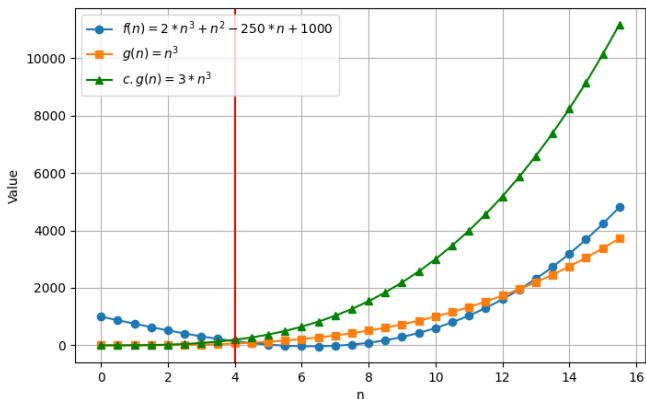
Suppose we have a 'complex' $f(n)$
and we want to upperbound it with a simple $g(n)$

Visual Explanation of Big O notation (2)



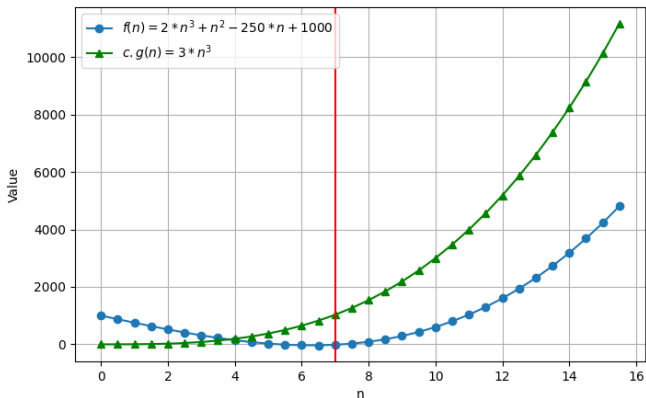
If we pick $g(n) = n^3$, $f(n) < g(n)$ only around $\approx [5..12]$
and $f(n) > g(n)$ again for larger n

Visual Explanation of Big O notation (3)



But if we scale $g(n)$ into $c \cdot g(n)$ with $c = 3$,
 $f(n) < c \cdot g(n)$ for $n > n_0 \approx 4$ (not easy to compute)

Visual Explanation of Big O notation (4)



We can just pick a clearer n_0 ,
e.g., $f(n) < c \cdot g(n)$ for $c = 3$ and $n > n_0 = 7$

Big O (upper bound)

Example: $100n + 1000 \in O(n^2)$

- ▶ $0 \leq 100n + 1000$ (for any positive n)
- ▶ $0 \leq 100n + 1000 \leq 101n$ (for $n \geq 1000$)
- ▶ $0 \leq 100n + 1000 \leq 101n \leq 101n^2$ (for $n \geq 1000$)
i.e., we can set $c = 101$ and $n_0 = 1000$

Hence, $100n + 1000 \in O(n^2)$

Question 3 at VisuAlgo Online Quiz

Let $f(n) = 10n^3 + 5n + 15$ and $g(n) = n^4$

We want to prove that $f(n) \in O(g(n))$ by showing that $0 \leq f(n) \leq c \cdot g(n)$ for all $n \geq n_0$

What should be the appropriate c and n_0 ? (there are > 1 answers)

- A). $c = 2, n_0 = 10$
- B). $c = 1, n_0 = 11$
- C). $c = 5, n_0 = 1$
- D). $c = 1, n_0 = 10$

New notation Ω (lower bound)

$f \in \Omega(g)$ if there exists constant $c > 0$ and $n_0 > 0$ such that
for all $n \geq n_0 : 0 \leq c \cdot g(n) \leq f(n)$

Interpretation: g is a lower bound on f

Ω (lower bound)

Example: $n^2 \in \Omega(100n + 1000)$

We swap $f(n)$ and $g(n)$ from the earlier Big O example

▶ $0 \leq \frac{1}{101} \cdot (100n + 1000) \leq n^2$ for $n \geq 1000$

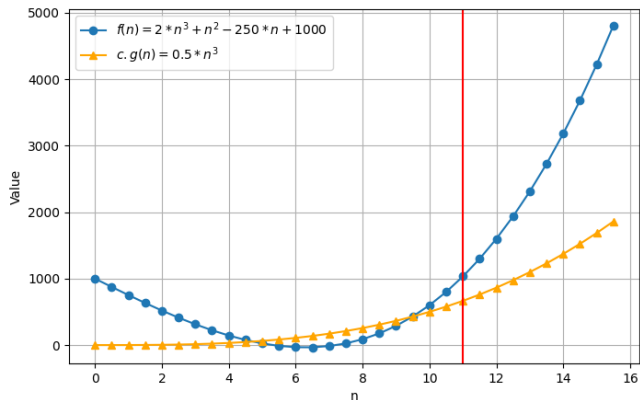
i.e., we can set $c = \frac{1}{101}$ and $n_0 = 1000$

just set this c to be the reciprocal of the c in Big O analysis

Again, there are many other possible c and n_0

PS: We usually have $f(n)$ as the more complex function and $g(n)$ to be the simpler one, i.e., $7n^2 + 5n + 77 \in \Omega(n^2)$

Visual Explanation of Ω -notation (1)



$$0 \leq c \cdot g(n) \leq f(n)$$

for $c = 0.5$ and $n > n_0 = 11$

New notation Θ (tight bound)

$f \in \Theta(g)$ if there exists constants $c_1, c_2 > 0$ and $n_0 > 0$ such that for all $n \geq n_0 : 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$

Interpretation: g is a tight bound on f

We will frequently do Θ analysis in CS3230
(but for some harder analysis, we may be content with just Big O)

Θ -notation (tight bound)

Example: $10n^2 + n \in \Theta(n^2)$

▶ $0 \leq c_1 \cdot n^2 \leq (10n^2 + n) \leq c_2 \cdot n^2$ for $n \geq n_0$

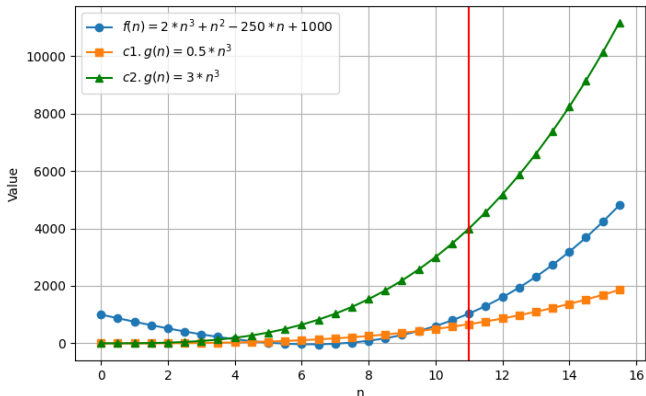
$0 \leq \frac{1}{2}n^2 \leq (10n^2 + n) \leq 11n^2$ for $n \geq 2$

i.e., $c_1 = \frac{1}{2}$, $c_2 = 11$, and $n_0 = 2$

again, these are not the only valid constants c_1 , c_2 , and n_0

Hence, $10n^2 + n \in \Theta(n^2)$

Visual Explanation of Θ -notation (1)



$$0 \leq c1 \cdot g(n) \leq f(n) \leq c2 \cdot g(n)$$

for $c1 = 0.5$, $c2 = 3$, and $n > n_0 = 11$

Relationship of O , Ω , and Θ

$$\Theta(g) = O(g) \cap \Omega(g)$$

Little-o (strict upper bound)

$f \in o(g)$ if **for any constant** $c > 0$, there exists $n_0 > 0$ such that for all $n \geq n_0 : 0 \leq f(n) < c \cdot g(n)$ (notice **for any constant** $c > 0$ instead of there exists constant $c > 0$ and $<$ instead of \leq)

PS: some textbooks define Little-o using \leq instead of $<$
This will only change the chosen c and/or n_0

Example: $n \in o(n^2)$

For any constant $c > 0$, let $n_0 = 1 + \frac{1}{c}$

Then, for $n \geq n_0$, $n < c \cdot n^2$

But $n^2 - n \notin o(n^2)$

Let's say we pick $c = \frac{1}{2}$ (just need to show one counterexample),
for any n_0 and large enough n , we have:

$$n^2 - n > \frac{1}{2}n^2$$

$$\frac{1}{2}n^2 > n$$

$$n^2 > 2n$$

ω (strict lower bound)

$f \in \omega(g)$ if **for any constant** $c > 0$, there exists $n_0 > 0$ such that for all $n \geq n_0 : 0 \leq c \cdot g(n) < f(n)$

Example: $n^2 - 36 \in \omega(n)$

For any constant $c > 0$, let $n_0 > \sqrt{36} + c$ (we can also set $n_0 = 7$),
 $0 \leq c \cdot n < n^2 - 36$

Asymptotic Notation: Taking Limits

Assume $f(n), g(n) > 0$, we have:

- ▶ $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in o(g(n))$
- ▶ $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) \in O(g(n))$
- ▶ $0 < \lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} < \infty \Rightarrow f(n) \in \Theta(g(n))$
- ▶ $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} > 0 \Rightarrow f(n) \in \Omega(g(n))$
- ▶ $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f(n) \in \omega(g(n))$

It is easier to show o , Θ , vs ω using limits

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f(n) \in o(g(n))$$

Proof:

By definition of limit, $\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = 0$, means

$\forall \epsilon > 0, \exists n_0 > 0$, such that $\forall n \geq n_0$,

$$\frac{f(n)}{g(n)} < \epsilon$$

Hence, for any constant $c > 0$ (i.e., we can set $c = \epsilon$), $\exists n_0 > 0$,
such that $\forall n \geq n_0$,

$$f(n) < \epsilon \cdot g(n), \text{ i.e.,}$$

$$f(n) < c \cdot g(n),$$

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

We will prove at least one other during Tut01

Example

By limit, show that $n^6 + 233n^2 \in \omega(n^2)$

$$\lim_{n \rightarrow \infty} \frac{n^6 + 233n^2}{n^2} = \lim_{n \rightarrow \infty} \frac{n^2 \cdot (n^4 + 233)}{n^2} = \lim_{n \rightarrow \infty} \frac{n^4 + 233}{1} = \infty$$

$$\Rightarrow f(n) \in \omega(g(n))$$

Asymptotic Notation: Some Properties

- ▶ Reflexivity: For O , Ω , and Θ ,
 $f(n) \in O(f(n))$, similarly for Ω and Θ
- ▶ Transitivity: For all five: O , Ω , Θ , o , and ω
 $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ implies $f(n) \in O(h(n))$
- ▶ Symmetry:
 $f(n) \in \Theta(g(n))$ iff $g(n) \in \Theta(f(n))$
- ▶ Complementary:
 $f(n) \in O(g(n))$ iff $g(n) \in \Omega(f(n))$
 $f(n) \in o(g(n))$ iff $g(n) \in \omega(f(n))$

We will prove some of these during Tut01

We end Lec01b here

See appendices for various math refresher

APPENDICES

Basic Math and Probability Revision

The next few slides are additional appendices to help students recall basic math and probability (to be used in future lectures and/or tutorials)

You can skip these slides if you still remember most of the content

Properties of Functions

- ▶ Exponentials
- ▶ Logarithms
- ▶ Summations
- ▶ Limits

Exponentials

$$a^{-1} = \frac{1}{a}$$

$$(a^m)^n = a^{mn}$$

$$a^m a^n = a^{m+n}$$

$$e^x \geq 1 + x$$

$$n^k \in o(a^n)$$

any exponential function with base $a > 1$ grows faster than any polynomial n^k for any constant $k > 0$

Logarithms

- ▶ Binary logarithm: $\log n = \log_2 n$
- ▶ Natural logarithm: $\ln n = \log_e n$
- ▶ Exponentiation: $\lg^k n = (\lg n)^k$
- ▶ Composition: $\lg \lg n = \lg(\lg n)$

$$a = b^{\log_b a}$$

$$\log_c(ab) = \log_c a + \log_c b$$

$$\log_b a^n = n \cdot \log_b a$$

$$\log_b a = \frac{\log_c a}{\log_c b}$$

$$\log_b \frac{1}{a} = -\log_b a$$

$$\log_b a = \frac{1}{\log_a b}$$

$$a^{\log_b c} = c^{\log_b a}$$

Base of logarithm does not matter in asymptotics, e.g.,

$$\log n \in \Theta(\ln n) \in \Theta(\log_{10} n)$$

Exponential of different bases differ by an exponential factor, e.g.,

$$4^n = 2^n \cdot 2^n \quad 7^n = \left(\frac{7}{4}\right)^n \cdot 4^n \quad 8^n = 4^n \cdot 2^n$$

Stirling's Approximation

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n \left(1 + \Theta\left(\frac{1}{n}\right)\right)$$

$$\log(n!) \in \Theta(n \log n)$$

► Arithmetic Series

$$\sum_{k=1}^n k = 1 + 2 + 3 + \dots + n = \frac{1}{2}n(n+1) \in \Theta(n^2)$$

▶ Geometric Series

$$\sum_{k=0}^n x^k = 1 + x + x^2 + \dots + x^n = \frac{x^{n+1} - 1}{x - 1}$$

▶ Infinite Geometric Series when $|x| < 1$

$$\sum_{k=0}^{\infty} x^k = \frac{1}{1 - x}$$

► Harmonic Series

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = \sum_{k=1}^n \frac{1}{k} = \ln n + O(1)$$

L'Hopital's Rule

$$\lim_{x \rightarrow \infty} \frac{f(x)}{g(x)} = \lim_{x \rightarrow \infty} \frac{f'(x)}{g'(x)}$$

Example:

$$\lim_{n \rightarrow \infty} \frac{n \log n}{n^2} = \lim_{n \rightarrow \infty} \frac{n \log n}{n \cdot n} = \lim_{n \rightarrow \infty} \frac{\log n}{n} = \lim_{n \rightarrow \infty} \frac{\frac{1}{n}}{1} = \lim_{n \rightarrow \infty} \frac{1}{n} = 0$$

$$\Rightarrow n \log n \in o(n^2)$$

Sample Space, Event

- ▶ A S is a set whose elements are called
- ▶ An A is a subset of the sample space S
- ▶ For an event A , the event $\bar{A} = S - A$
- ▶ Example: For throwing a dice:
The sample space is $\{1, 2, 3, 4, 5, 6\}$
The event of throwing an even number is $A = \{2, 4, 6\}$

Probability Distribution

- ▶ A $Pr()$ is a mapping from events to real numbers such that:
 - ▶ $Pr(A) \geq 0, \forall$ events A
 - ▶ $Pr(S) = 1$
 - ▶ $Pr(A \cup B) = Pr(A) + Pr(B)$ for events A and B .
- ▶ For dice, a uniform distribution is a reasonable model
 - $Pr(i) = 1/6$, for $i = 1, 2, 3, 4, 5, 6$
 - $Pr(\{2, 4, 6\}) = 1/6 + 1/6 + 1/6 = 3/6 = 1/2$

- ▶ In general: $Pr(A) \leq Pr(A \cup B) = Pr(A) + Pr(B) - Pr(A \cap B) \leq Pr(A) + Pr(B)$
 - ▶ Example: If $A = \{2, 4, 6\}$, $B = \{1, 2\}$, then
 $Pr(A \cup B) = Pr(A) \cup Pr(B) - Pr(A \cap B) = 1/2 + 1/3 - 1/6 = 2/3$
- ▶ Two events A and B are called independent if $Pr(A \cap B) = Pr(A) \cdot Pr(B)$
 - ▶ Example: We flip two fair coins
 Let A be the event that the first coin is heads
 Let B be the event that second coin is heads
 Then $Pr(A \cap B) = Pr(A) \cdot Pr(B) = \frac{1}{2} \cdot \frac{1}{2} = \frac{1}{4}$.

Conditional probability

- ▶ The probability of an event A given another event B (where $Pr(B) \neq 0$) is defined as

$$Pr(A|B) = \frac{Pr(A \cap B)}{Pr(B)}$$

- ▶ Bayes' Theorem:

$$\begin{aligned} Pr(A|B) &= \frac{Pr(A)Pr(B|A)}{Pr(B)} \\ &= \frac{Pr(A) \cdot Pr(B|A)}{Pr(A) \cdot Pr(B|A) + Pr(\bar{A}) \cdot Pr(B|\bar{A})} \end{aligned}$$

Bayes' Theorem: Example

We have one fair coin and one biased coin that always gives heads
We choose one coin out of these two coins, uniformly at random
When we toss the chosen coin twice, we get heads both times
What is the probability that the coin chosen was biased?

Let A be the event of choosing the biased coin and let B be the event that both coin tosses with the chosen coin are heads

$$Pr(A) = \frac{1}{2}, Pr(B|A) = 1, Pr(\bar{A}) = \frac{1}{2}, Pr(B|\bar{A}) = \frac{1}{4}.$$

$$Pr(A|B) = \frac{Pr(A) \cdot Pr(B|A)}{Pr(A) \cdot Pr(B|A) + Pr(\bar{A}) \cdot Pr(B|\bar{A})}$$
$$= \frac{(1/2) \cdot 1}{(1/2) \cdot 1 + (1/2) \cdot (1/4)} = \frac{4}{5}$$

Random Variable

A X is a function that maps the sample space S to real numbers

The function $f(x) = Pr(X = x)$ is the of X

Example: In a roll of a pair of dice, let X be the max of the two values shown on the dice. Then $Pr(X = 3) = 5/36$, because the elementary events which give $X = 3$ are $(1, 3)$, $(2, 3)$, $(3, 3)$, $(3, 1)$, $(3, 2)$, and there are a total of 36 elementary events in the sample space (each of which is equally likely)

Expectation

- ▶ The or of a random variable X is

$$E(X) = \sum_x x \cdot Pr(X = x)$$

- ▶ Example: Suppose X is the outcome of a dice

$$E(X) = \sum_i i \cdot Pr(X = i) = \sum_i (i/6) = 3.5$$

Linearity of Expectations

- ▶ For any two events X, Y and a constant a

$$E(X + Y) = E(X) + E(Y)$$

$$E(a \cdot X) = a \cdot E(X)$$

- ▶ If X and Y are independent then

$$E(XY) = E(X) \cdot E(Y)$$

Bernoulli Trial

- ▶ An instance of a Bernoulli trial has probability p of success and probability $q = 1 - p$ of failure
- ▶ Suppose we have a sequence of independent Bernoulli trials, each with probability p of success. Let X be the number of trials needed to obtain a success for the first time, then:

$$Pr(X = k) = q^{k-1}p$$

$$E(X) = \frac{1}{p}$$

- ▶ Suppose X is the number of successes in n Bernoulli trials, then:

$$Pr(X = k) = \binom{n}{k} p^k q^{n-k}$$

Acknowledgement

The slides are modified from previous editions of this course and similar course elsewhere.

List of credits: Erik D. Demaine, Charles E. Leiserson, Surender Baswana, Leong Hon Wai, Lee Wee Sun, Ken Sung, Arnab Battacharya, Diptarka Chakraborty, Steven Halim, Sanjay Jain, Chang Yi-Jun, Warut Suksompong, Eric Han Liang Wee.