EE 360C – ALGORITHMS

Lecture 5

Complexity - 1

Vallath Nandakumar

University of Texas at Austin

> "You'll only use the terms 'surjection,' 'injection,' and 'bijection' if you drink your beer out of a frosted glass. If you drink it from the bottle, you'll use 'one-to-one' and 'onto.'"
>
> — Proof-writing professor

Agenda

- ☞ **Last class**
  - **Stable matching**
- ☞ **This class**
  - **Time complexity of algorithms**

DEFINITION OF ALGORITHM

☞ Definition of Algorithm
- Any well-defined computational procedure
- Takes some value or set of values as input
- Produces some value or set of values as output

☞ Algorithm
- Sequence of computational steps that transforms input to output
- Tool for solving well-specified computational problem
- “Correct” if for every input instance halts with correct output
- “Solve” computational problem if correct

ALGORITHM ANALYSIS BASICS

☞ Analyzing Algorithm

- Predicting resources that algorithm requires
- Need model of implementation technology to underlie analysis

☞ Goal

- Develop (correct) efficient algorithms as solutions to well-defined problems

ALGORITHM EFFICIENCY

- ☞ Definition 1: Algorithm efficient if when implemented, it runs **quickly** on real input instances
- ☞ Limitations of Definition 1
  - Vague
  - Even bad algorithm can run fast when applied to small test cases
  - Good algorithms can run slow when implemented poorly
  - What is “real” input instance?
    - May not know *a priori*
  - Definition doesn’t consider how well algorithm performance scales as problem size grows

ALGORITHM EFFICIENCY

☞ Ideally, would like definition of algorithm efficiency that is
  - Platform-independent
  - Instance-independent
  - Predictive value with respect to increasing instance sizes

☞ Example
  - Stable matching problem has size N, total sizes of input preferences lists
    - n men and n women, each with preference list of n length
    - $N = 2n^2$

ALGORITHM EFFICIENCY

Goedel, Escher, Bach

☞ Input Size

- Definition depends on particular computational problem
  - Generally running time grows with size of input

Bark

☞ Running Time

- Number of primitive operations or steps executed
  - Should be machine independent
  - Assume constant amount of time required to execute each line of pseudocode

ALGORITHM EFFICIENCY

- Worst-case running time
  - Worst possible running time algorithm could have over all inputs of size *N*
- Average-case running time
  - Average running times over “random” instances

[1, 2, 3, 4]

4, 3, 2, 1

ALGORITHM EFFICIENCY

- ☞ Definition 2: Algorithm efficient if achieves qualitatively better worst-case performance at analytical level than brute-force search
- ☞ Limitations of Definition 2
  - Little vague – What qualifies as "qualitatively better"?
- ☞ Example: Consider stable matching algorithm
  - Brute-force search generates all *n!* possible pairings
  - Running time on order of n^2 better

ALGORITHM EFFICIENCY

- Definition 3: Algorithm efficient if has polynomial running time
  - Precise definition and also negatable

- Limitation of Definition 3
  - Running time of n^{100} not great
  - Running time of $n^{1+.02(\log n)}$ not bad

$$\frac{1}{1} + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \cdots$$

43

ALGORITHM EFFICIENCY

☞ Brute Force

- For many non-trivial problems, natural search algorithm that checks every possible solution
- Typically takes 2^N time or worse for size N
- Unacceptable in practice for large input size

☞ Desirable Scaling Property

- When input size doubles, algorithm should only slow down by some constant factor C
  - There exists constants $c > 0$ and $d > 0$ such that on every input size N
    - Running time bounded by cN^d steps
- Algorithm considered poly-time if property holds

RUNNING TIMES

- Running times for algorithms
  - On processor executing million instructions per second

	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

ASYMPTOTIC NOTATION

☞ Asymptotic Efficiency

- How running time of algorithm scales with increasing input size in limit

☞ Goal to identify similar classes of algorithms with similar behavior

- Measure running times in number of primitive “steps” algorithm must perform

ASYMPTOTIC BOUNDS

☞ **Not necessary to be precise about running times**

- **Care about rates of growth**

☞ **Represent asymptotic running time of algorithm**

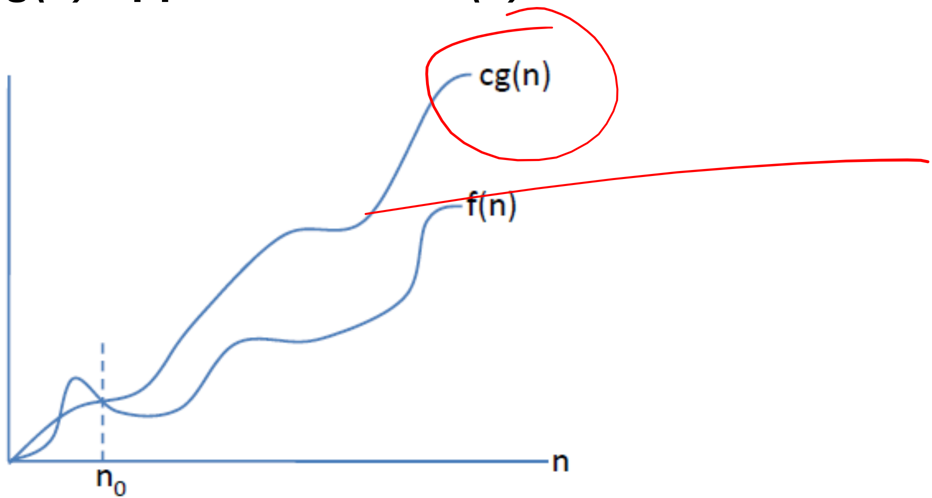
- **Growth rate relative to input size**
- **Independent of constant factors**

ASYMPTOTIC UPPER BOUNDS: *O*-NOTATION

- Given function $T(n)$ representing algorithm's running time
  - $T(n)$ is $O(f(n))$ ("$T(n)$ is order $f(n)$")
  - For sufficiently large n, $T(n)$ bounded above by constant multiple of $f(n)$
- Definition
  - Given $g(n)$, $O(g(n))$ denotes set of functions
  - $O(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$
- $O(g(n))$ is set, but usually notation abused
  - $f(n) = O(g(n))$

O-NOTATION

- **For all values of *n* to right of n_0**
  - **Value of *f(n)* on or below *cg(n)***
  - ***g(n)* upperbound for *f(n)***



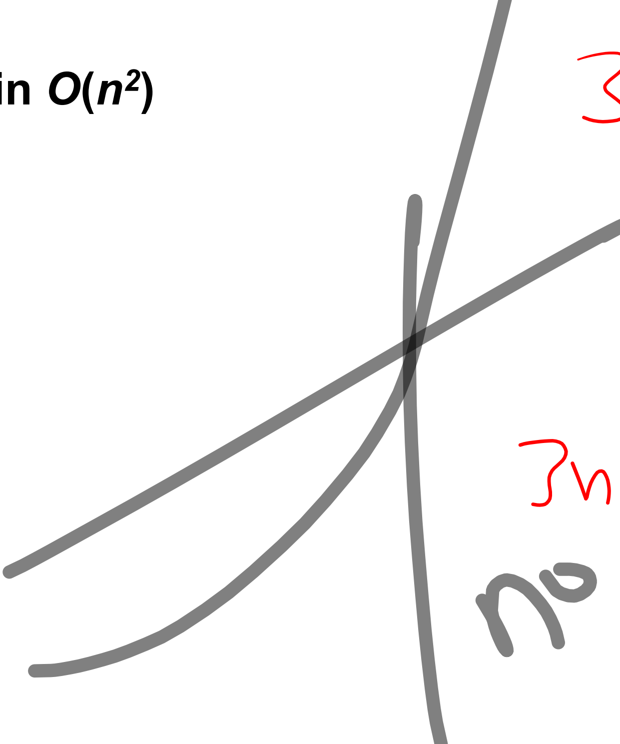
O-NOTATION: EXAMPLE

- ☞ $T(n) = pn^2 + qn + r$ is in $O(n^2)$
  - $T(n) = pn^2 + qn + r \leq pn^2 + qn^2 + rn^2 = (p + q + r)n^2$ for all $n \geq 1$

- ☞ Required definition of $O(\cdot)$:
  - $T(n) \leq cn^2$, where $c = p + q + r$

O-NOTATION: EXAMPLE

☞ $an + b$ in $O(n^2)$



$3n+5$

$n > 100$

$n_0 = 100$

$3n+5 \leq n^2$

$n > 100$

O-NOTATION

- ☞ Some texts use *O* to informally describe asymptotically tight bounds (i.e., when we use Θ)
- ☞ *O*-notation useful in quickly and easily bounding running time of algorithm by inspection

iClicker – running time of insertion sort

1. O(n) 2. O(n^2) 3. O($n\log n$) 4. O(n^3)

In the worst case, the running time of insertion sort is:

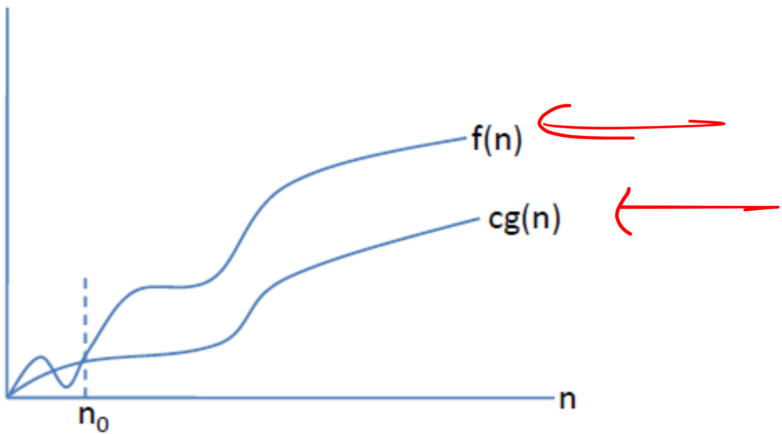
A. 1
B. 2
C. 1, 2, 3
D. 2, 4
E. 2, 3, 4

ASYMPTOTIC LOWER BOUNDS: Ω-NOTATION

- Lower bounds useful for stating algorithm's running time is at least some magnitude
- Definition
  - Given $f(n)$, $\Omega(g(n))$ denotes set of functions
  - $\Omega(g(n)) = \{f(n)$: there exists positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$ for all $n \geq n_0\}$

Ω-NOTATION

☞ **For all values of *n* to right of n_0**
- **Value of *f(n)* on or below *cg(n)***
- ***g(n)* lowerbound for *f(n)***



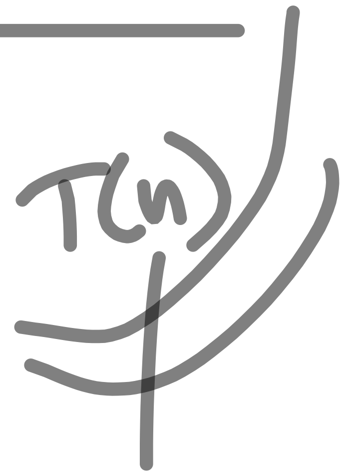
Ω-NOTATION EXAMPLE

☞ $T(n) = pn^2 + qn + r$ is in $\Omega(n^2)$

- $T(n) = pn^2 + qn + r \geq pn^2$ for all $n \geq 0$

☞ Required definition of $\Omega(\cdot)$:

- $T(n) \geq cn^2$, where $c = p$



iClicker – running time of insertion sort

1. $\Omega(n)$
2. $\Omega(n^2)$
3. $\Omega(\log n)$
4. $\Omega(\text{sqrt}(n))$

In the worst case, the running time of insertion sort will be:

A. 1, 2, 3
B. 1, 2, 3, 4
C. 2
D. 1, 3, 4

ASYMPTOTIC TIGHT BOUNDS: Θ-NOTATION

- If running time $T(n)$ both $O(f(n))$ and $\Omega(f(n))$, then tight bound

$3x^2 + 5x + 1 \in \Theta(x^2)$

- Definition
  - Given $f(n)$, $\Theta(g(n))$ denotes set of functions
  - $\Theta(g(n)) = \{f(n)$: there exists positive constants c_1, c_2, and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

$3x^2 + 5x + 1 \in O(n^2)$?

$4x^2$ is an upper bound $\Omega(n^2)$?

$n > 100$

$1.5x^2$ is a lower bound $n > 100$

Θ-NOTATION

☞ For all values of n to right of n_0

- Value of $f(n)$ at or above $c_1g(n)$ and at or below $c_2g(n)$
- $g(n)$ lowerbound for $f(n)$