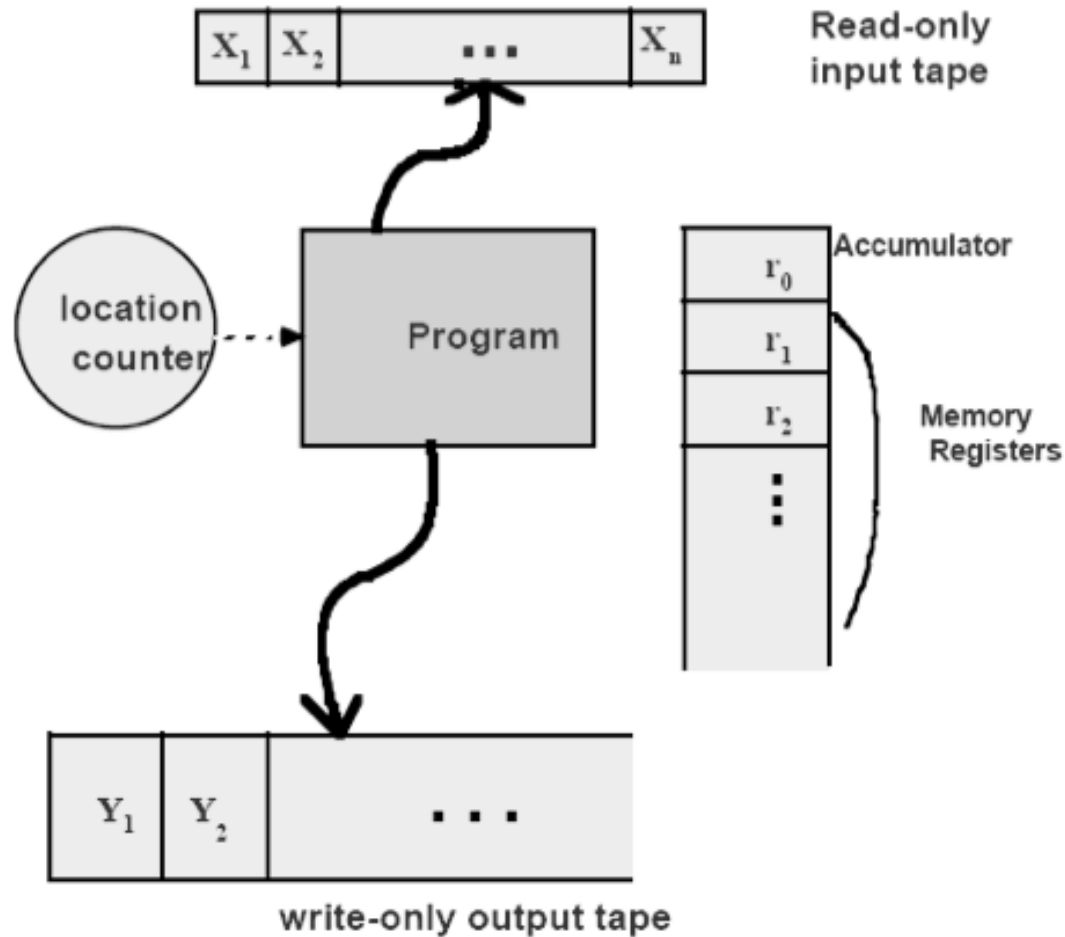# Algorithm Complexity Notations

Debswapna Bhattacharya, PhD
Assistant Professor
Computer Science and Software Engineering
Auburn University

Analyzing algorithms

- **Model of Computation:** specifies what operations an algorithms is allowed and cost (time) for each operation

- **Random Access Machine: the computer that runs the algorithms is a** one processor random access machine (RAM)

Random Access Machine

X₁ | X₂ | ... | Xₙ   Read-only input tape

location counter → Program

r₀ Accumulator
r₁
r₂ Memory Registers

Y₁ | Y₂ | ...

write-only output tape

RAM Assumptions

– Each register holds one data unit
– Program can't modify itself
– Memory instructions involve simple arithmetic
  – Addition, subtraction
  – Multiplication, division and control states (if-then, etc.)

Complexity Measures of Algorithms

Input X
size n=|X|

Algorithm A

output

Time$_A$ (X) = time cost of Algorithm A, input X

Space$_A$ (X) = space cost of Algorithm A, input X

Note: "time" and "space" depend on machine

Complexity Measures of Algorithms

– Worst case
time complexity

$$T_A(n) = \max_{\{x:|x|=n\}}(\text{Time}_A(X))$$

– Average case complexity
for random inputs

$$E(T_A(n)) = \sum_{\{x:|x|=n\}} \text{Time}_A(X)\text{Prob}(X)$$

– Worst case
space complexity

$$S_A(n) = \max_{\{x:|x|=n\}}(\text{Space}_A(X))$$

– Average case complexity
for random inputs

$$E(S_A(n)) = \sum_{\{x:|x|=n\}} \text{Space}_A(X)\text{Prob}(X)$$

RAM Cost Criteria

Cost Criteria

    Time = # RAM instructions

    space = # RAM memory registers

Varieties of Computing Machine Models
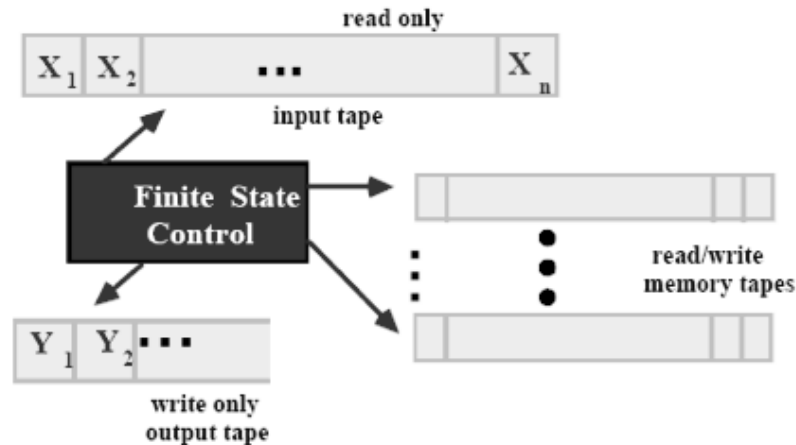
RAM
Straight line programs
Circuits
Bit vectors
Lisp machines
 …
Turning Machines: Invented by Turing (a Cambridge logician)
        Built by British for WWII cryptography !

Analyzing Algorithms

– The time needed by an algorithm to solve a problem depends on the steps of the algorithm that have to be executed (the number of primitive "instructions" or "operations" executed before it halts on a given input and produces an output) and how big the given input is.

– We specify this using an expression $T(n)$ where n is the size of the input.

– So if we know $T(n)$, we can estimate how much time that computer will take to solve that problem (see example on text page 11). Alternately, if we know how much time we have, we can decide the largest size of a problem that can be solved (see problem 1-1 in Chapter 1).

Topics

1. Complexity Notations
   What is the technical meaning of Big-Oh (O)? How about the other notations (small-oh o, theta Θ, omega Ω and small-omega ω)?
2. Approximate Big-Oh analyses of non recursive algorithms
3. Detailed complexity analyses of non recursive algorithms
4. Approximate Big-Oh analyses of recursive algorithms
5. Detailed complexity analyses of recursive algorithms
   1. Characterizing recursive algorithms by developing recurrence relations
   2. Analyzing their complexity by solving recurrence relations

Asymptotic notations of complexity orders: Theta

$$T(n) = \Theta(g(n))$$

$$\Theta(g(n)) = \{T(n) \mid \exists c_1, c_2, n_0 \text{ s.t. } 0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n)$$
$$\text{for all } n \geq n_0\}$$

Note the meanings of the three constants c_1, c_2 & n_0

Example:

$Suppose\, T(n) = \dfrac{n^2}{2} - 3n$

$First\ find\ a\ multiple\ c_1\, of\ n^2\ so\ that\ c_1 n^2 \leq T(n)\ for$
$all\ values\ of\ n\ greater\ than\ or\ equal\ to\ some\ value.$
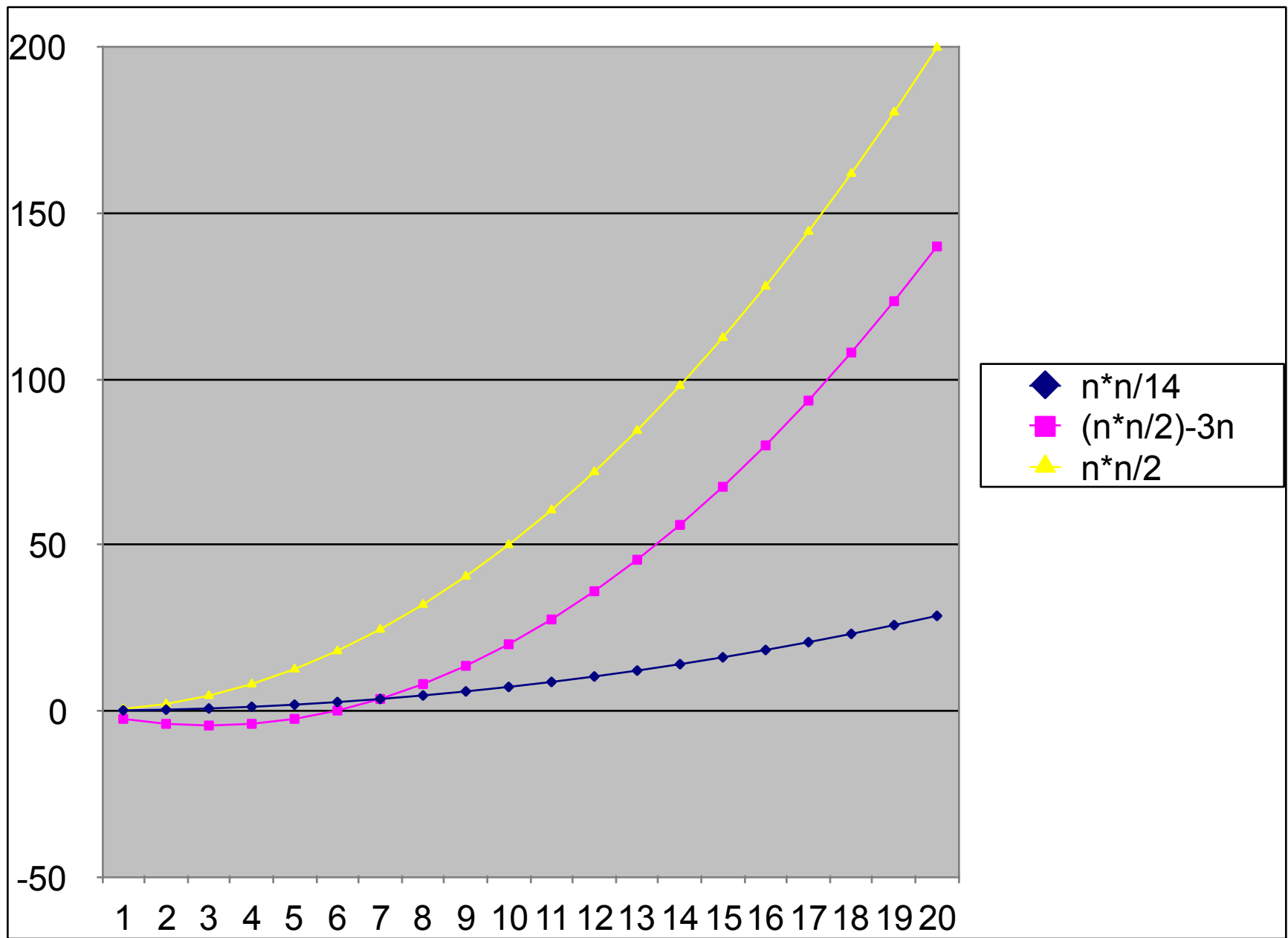
$E.g.,\ \dfrac{n^2}{14} \leq \dfrac{n^2}{2} - 3n\ when\ n \geq 7.\ So\ c_1 = \dfrac{1}{14}$

$Now\ find\ a\ multiple\ c_2\ of\ n^2\ so\ that\ c_2 n^2 \geq T(n)\ for$
$all\ values\ of\ n\ greater\ than\ or\ equal\ to\ some\ value.$

$E.g.,\ \dfrac{n^2}{2} \geq \dfrac{n^2}{2} - 3n\ when\ 3n \geq 0\ or\ n \geq 0.\ So\ c_2 = \dfrac{1}{2}$

$0 \leq c_1 n^2 \leq T(n) \leq c_2 n^2\ for\ all\ n \geq n_0$

$where\ c_1 = \dfrac{1}{14}, c_2 = \dfrac{1}{2}\ and\ n_0 = 7; so\, T(n) = \Theta(n^2)$

$$T(n) = an^2 + bn + c, a, b, c \text{ constants}, a>0.$$

$$\Rightarrow T(n)=\Theta(n^2).$$

- In general:

if $T(n)$ *is a polynomial of the form* $\sum_{i=0}^{d} a_i n^i$ where a_i are constant with $a_d > 0$

Then $T(n) = \Theta(n^d).$

Big-Oh: upper bound

$$T(n) = O(g(n))$$

$$O(g(n)) = \{T(n) \mid \exists c, n_0 \text{ s.t. } 0 \leq T(n) \leq cg(n) \; \forall n \geq n_0\}$$

Note the meanings of the two constants c & n_0

Omega: lower bound

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

$$\Omega(g(n)) = \{T(n) \mid \exists c, n_0 \text{ s.t. } 0 \leq cg(n) \leq T(n) \; \forall n \geq n_0\}$$

Note the meanings of the two constants c & n_0

Small-Oh: strict upper bound

$$T(n) = o(g(n))$$

$$o(g(n)) = \{T(n) \mid \forall c, \exists n_0 \forall n \geq n_0, 0 \leq T(n) < cg(n)\}$$

Note the meaning of "for all c there exists n_0"

Small-Omega: strict lower bound

$$T(n) = \omega(g(n))$$

$$\omega(g(n)) = \{T(n) \mid \forall c, \exists n_0 \, \forall n \geq n_0, 0 \leq cg(n) < T(n)\}$$

Note the meaning of "for all c there exists n_0"

Understand why...

$$If \ T(n) = \frac{n^2}{2} - 3n \ then$$

$$T(n) = \Theta(n^2)$$

$$T(n) = O(n^2)$$

$$T(n) = o(n^3)$$

$$T(n) = \Omega(n^2)$$

$$T(n) = \omega(n)$$

Suppose we know the following about algorithms A1, A2 and A3, all of which correctly solve the same problem. A1 is $\Omega(n^2)$; A2 is $\Theta(n^2)$; A3 is $o(n^2)$. Mark the following statements as True, False or Can't Say based on the given information.

True/False/Can't-Say??

A1 is the least efficient algorithm
Can't Say

As input size increases, A3 will outperform all other algorithms
Yes

A1 may be <u>less</u> or <u>more</u> or <u>as</u> efficient as A2
True

In the absence of additional information, it is <u>reasonable</u> to <u>assume</u> that A3 is more efficient than A2, which is more efficient than A1.
True

Chapter 3 Reading Assignments

Introduction & Section 3.1

p. 43-51 (Omit the rest starting with "comparing functions")

3.2

you should also know some basic math (will not be covered in class): p. 53-57 (Omit the rest starting with "Factorials")

Chapter 3 Thinking Assignments

Do problems 3.1-3 & 3.1-4, p.53