Algorithm Complexity Notations

AUBURN UNIVERSITY

Hugh Kwon

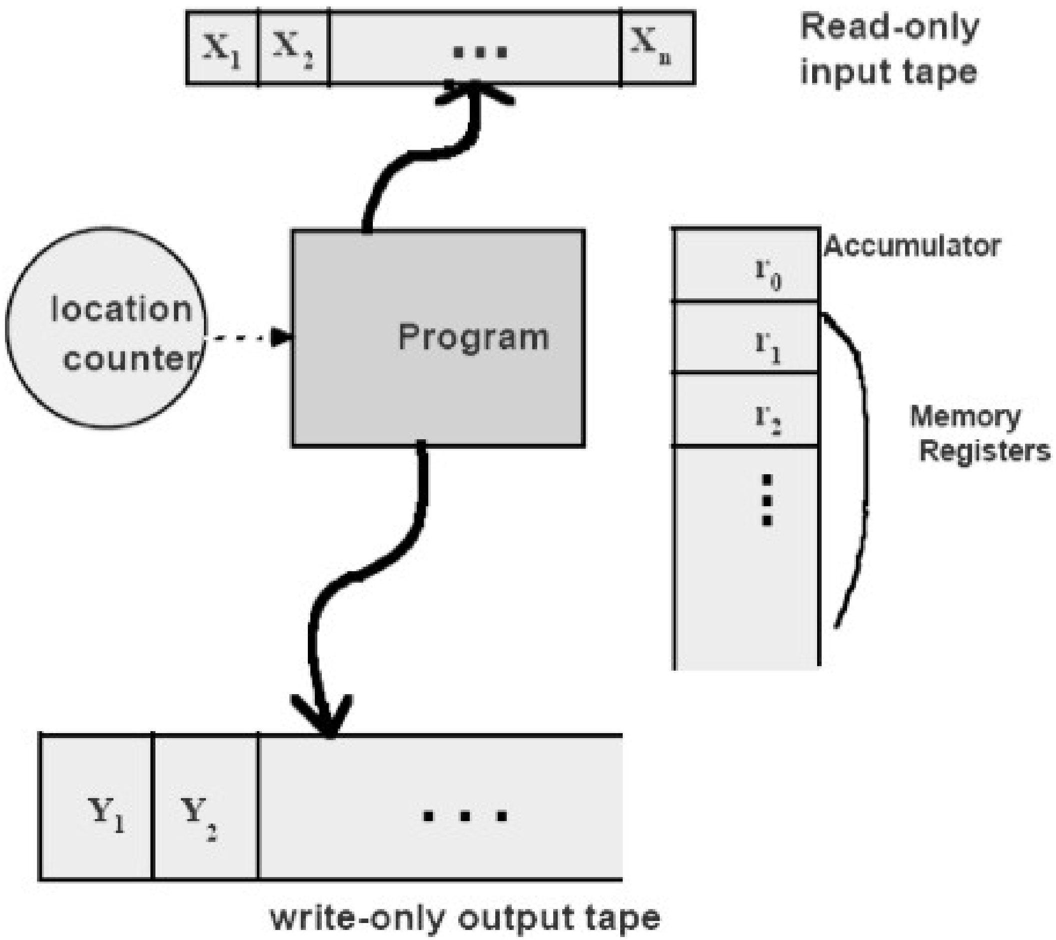
Slides adapted from Dr. Debswapna Bhattacharya's class

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



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

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

- Average case complexity
  for random inputs

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

- Worst case
  space complexity

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

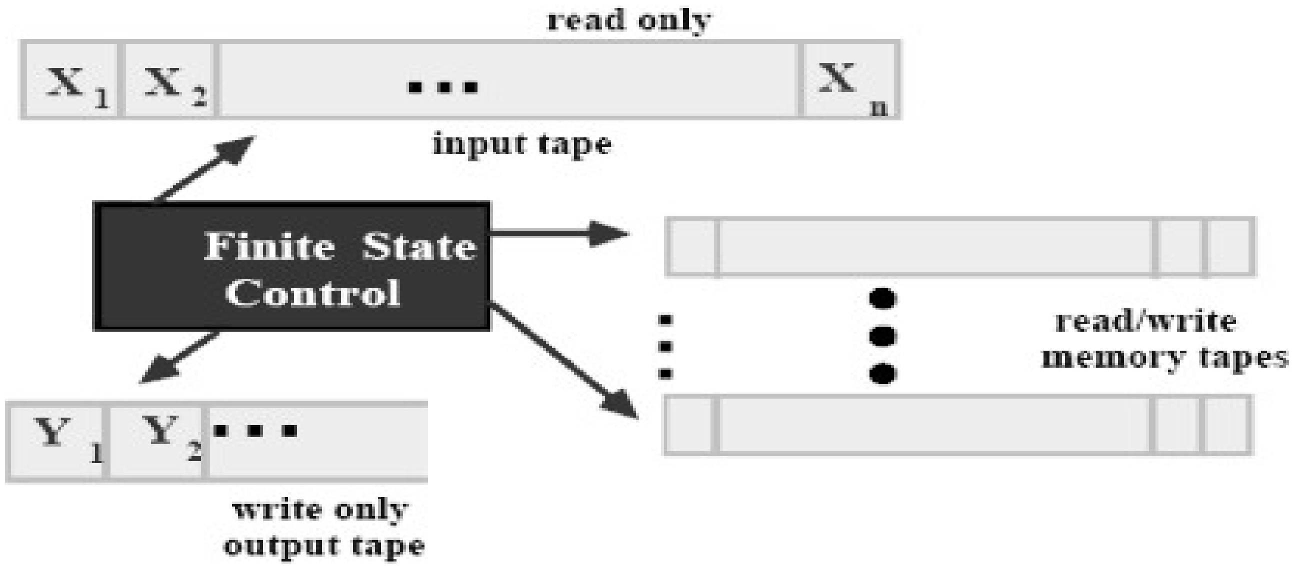
- Average case complexity
  for random inputs

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

RAM Cost Criteria

- Cost Criteria
  - Time = # RAM instructions
  - space = # RAM memory registers

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

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

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

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

Example

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

First find a multiple c_1 *of* n^2 *so that* $c_1 n^2 \le T(n)$ *for all values of n greater than or equal to some value.*

$$E.g.,\ \frac{n^2}{14} \le \frac{n^2}{2} - 3n \text{ when } n \ge 7.\ \text{So } c_1 = \frac{1}{14}$$

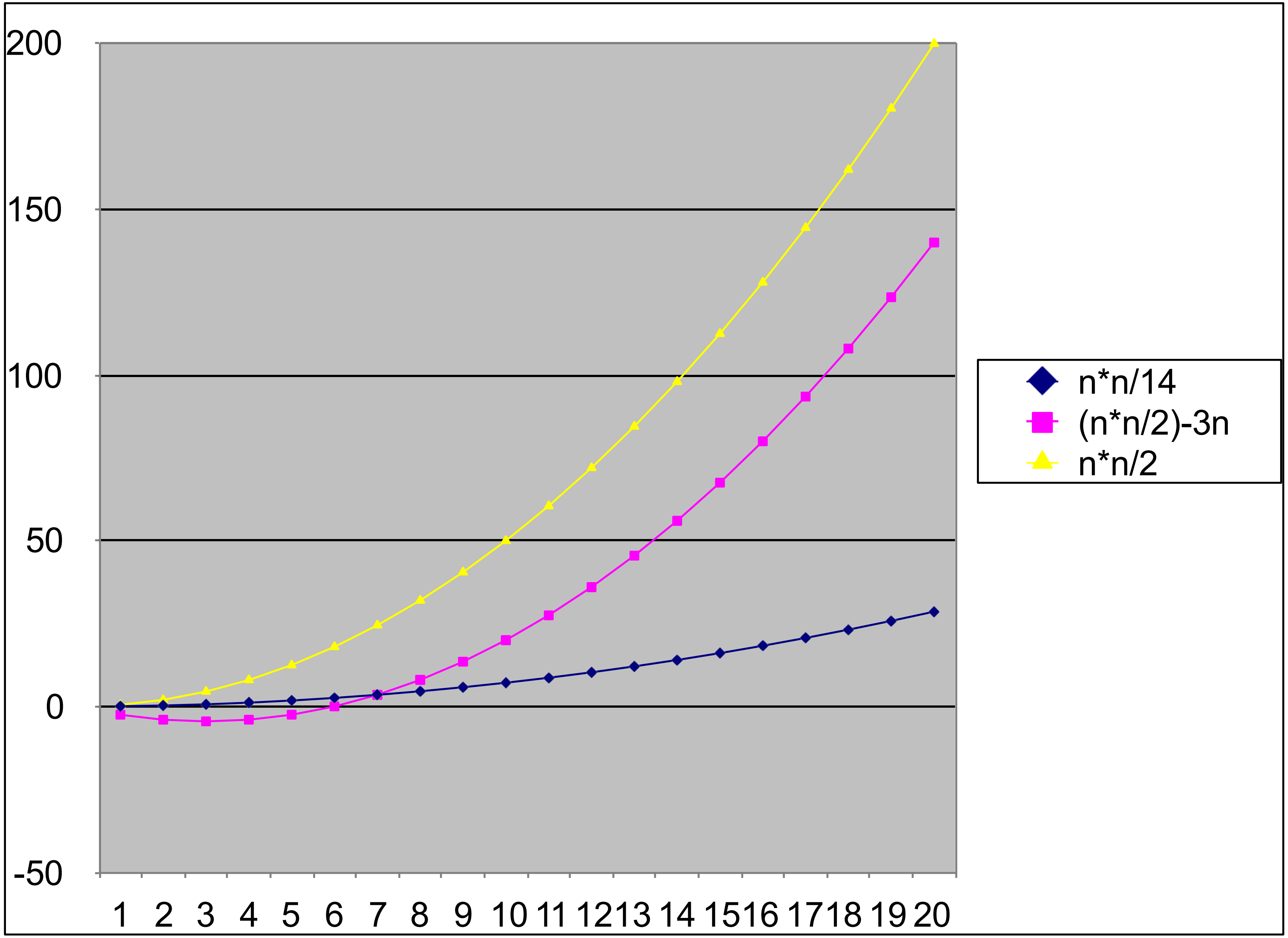
Now find a multiple c_2 *of* n^2 *so that* $c_2 n^2 \ge T(n)$ *for all values of n greater than or equal to some value.*

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

$$0 \le c_1 n^2 \le T(n) \le c_2 n^2 \text{ for all } n \ge n_0$$

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

Example



200
150
100
50
0
-50
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
n*n/14
(n*n/2)-3n
n*n/2

$T(n) = an^2 + bn + c$, a, b, c 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 $T(n) = \Theta(n^d)$.

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

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

Note the meanings of the two constants c & n_0

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

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

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"

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

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

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

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

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

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

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 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")

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

AUBURN UNIVERSITY