CS206 Data Structures

Analysis Tools: Part 2

Sung-eui Yoon (윤성의)

Department of Computer Science
KAIST

http://sglab.kaist.ac.kr/~sungeui

Class Objectives

- ☐ Know the definition of big-Oh notation with big-Omega and big-Theta
- ☐ Can perform asymptotic analysis

Big-Oh and Growth Rate

- The big-Oh notation gives an upper bound on the growth rate of a function
- The statement "$f(n)$ is $O(g(n))$" means that the growth rate of $f(n)$ is no more than the growth rate of $g(n)$
- We can use the big-Oh notation to rank functions according to their growth rate

	f(n) is O(g(n))	g(n) is O(f(n))
g(n) grows more	Yes	No
f(n) grows more	No	Yes
Same growth	Yes	Yes

Big-Oh Rules

- If is $f(n)$ a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$" (although the latter is still correct)
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

Asymptotic Algorithm Analysis

- The asymptotic analysis of an algorithm determines the running time in big-Oh notation
- To perform the asymptotic analysis
  - We find the worst-case number of primitive operations executed as a function of the input size
  - We express this function with big-Oh notation
- Example:
  - We determine that algorithm arrayMax executes at most 8n – 2 primitive operations
  - We say that algorithm arrayMax "runs in O(n) time"
- Since constant factors and lower-order terms are eventually dropped anyhow, we can disregard them when counting primitive operations

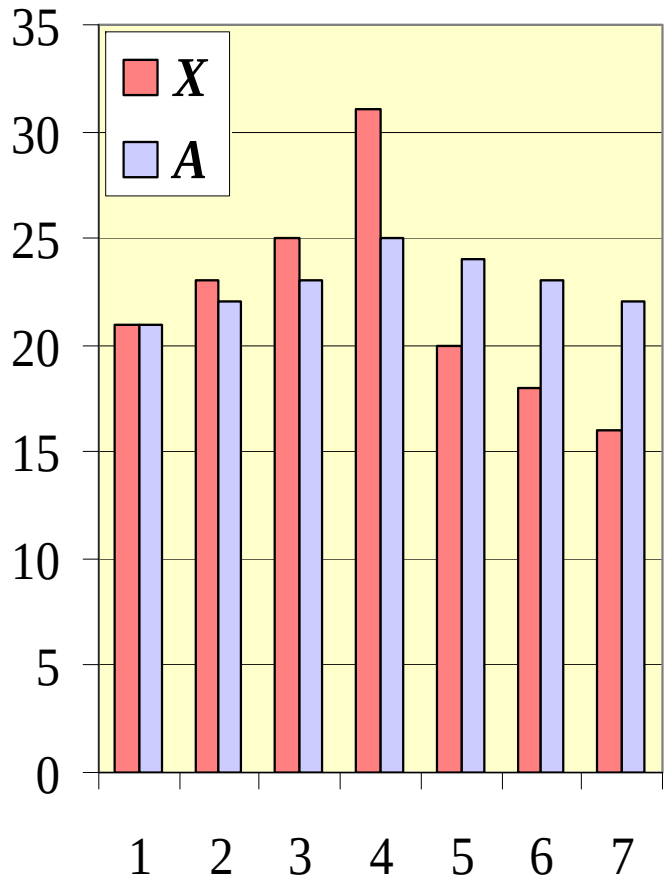
Computing Prefix Averages

- We further illustrate asymptotic analysis with two algorithms for prefix averages

- The i-th prefix average of an array X is average of the first (i + 1) elements of X:

$$A[i] = (X[0] + X[1] + \ldots + X[i])/(i+1)$$

- Computing the array A of prefix averages of another array X has applications to financial analysis



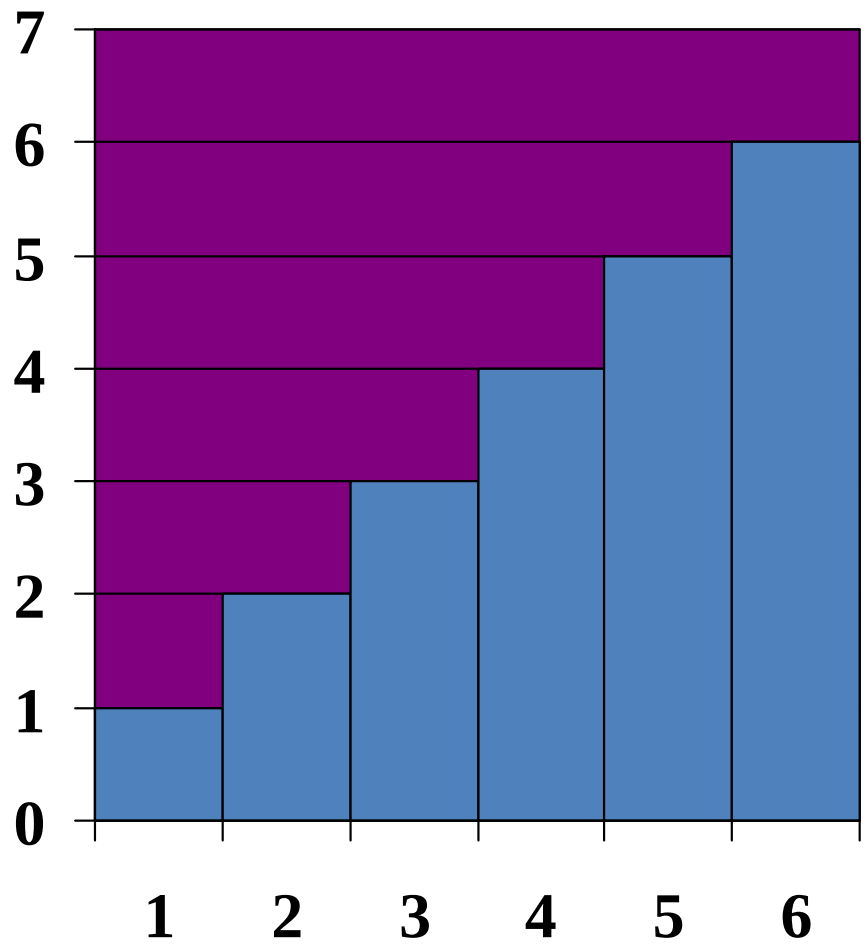
Prefix Averages (Quadratic)

- The following algorithm computes prefix averages in quadratic time by applying the definition

Algorithm *prefixAverages1*(X, n)	#operations
Input array X of n integers	
Output array A of prefix averages of X	
$A \leftarrow$ new array of n integers	n
for $i \leftarrow 0$ **to** $n - 1$ **do**	n
$s \leftarrow X[0]$	n
for $j \leftarrow 1$ **to** i **do**	$1 + 2 + \ldots + (n - 1)$
$s \leftarrow s + X[j]$	$1 + 2 + \ldots + (n - 1)$
$A[i] \leftarrow s / (i + 1)$	n
return A	1

Arithmetic Progression

- The running time of ***prefixAverages1*** is $O(1 + 2 + \ldots + n)$
- The sum of the first n integers is $n(n + 1) / 2$
  - There is a simple visual proof of this fact
- Thus, algorithm ***prefixAverages1*** runs in $O(n^2)$ time



Prefix Averages (Linear)

☐ The following algorithm computes prefix averages in linear time by keeping a running sum

Algorithm *prefixAverages2*(*X*, *n*)	
Input array *X* of *n* integers	
Output array *A* of prefix averages of *X*	#operations
A ← new array of *n* integers	*n*
s ← 0	1
for *i* ← 0 **to** *n* – 1 **do**	*n*
s ← *s* + *X*[*i*]	*n*
A[*i*] ← *s* / (*i* + 1)	*n*
return *A*	1

- Algorithm ***prefixAverages2*** runs in $O(n)$ time

Math Tools for Analysis of Algorithms

- ☐ Summations
- ☐ Logarithms and Exponents
  - **properties of logarithms:**

    $\log_b(xy) = \log_b x + \log_b y$

    $\log_b (x/y) = \log_b x - \log_b y$

    $\log_b x^a = a\log_b x$

    $\log_b a = \log_x a / \log_x b$
  - **properties of exponentials**:

    $a^{(b+c)} = a^b a^c$

    $a^{bc} = (a^b)^c$

    $a^b / a^c = a^{(b-c)}$

    $b = a^{\log_a b}$

    $b^c = a^{c*\log_a b}$
- ☐ Proof techniques
- ☐ Basic probability

Relatives of Big-Oh

- big-Omega
  - f(n) is $\Omega(g(n))$ if there is a constant c > 0 and an integer constant n_0 ¿ 1 such that f(n) ¿ c•g(n) for n ¿ n_0
- big-Theta
  - f(n) is $\Theta(g(n))$ if there are constants c' > 0 and c'' > 0 and an integer constant $n_0 \geq 1$ such that $c' \cdot g(n) \leq f(n) \leq c'' \cdot g(n)$ for $n \geq n_0$

Intuition for Asymptotic Notation

- big-Oh
  - f(n) is O(g(n)) if f(n) is asymptotically *less than or equal to* g(n)
- big-Omega
  - f(n) is Ω(g(n)) if f(n) is asymptotically *greater than or equal to* g(n)
- big-Theta
  - f(n) is § (g(n)) if f(n) is asymptotically *equal to* g(n)

Example Uses of the Relatives of Big-Oh

☐ $5n^2$ is $\Omega(n^2)$

- f(n) is Ω(g(n)) if there is a constant c > 0 and an integer constant n_0 ¿ 1 such that f(n) ¿ c•g(n) for n ¿ n_0
- let c = 5 and n_0 = 1

☐ $5n^2$ is $\Omega(n)$

- f(n) is Ω(g(n)) if there is a constant c > 0 and an integer constant n_0 ¿ 1 such that f(n) ¿ c•g(n) for n ¿ n_0
- let c = 1 and n_0 = 1

☐ $5n^2$ is $\Theta(n^2)$

- f(n) is § (g(n)) if it is $\Omega(n^2)$ and $O(n^2)$.
  We have already seen the former, for the latter recall that f(n) is O(g(n)) if there is a constant c > 0 and an integer constant n_0 ¿ 1 such that f(n) < c•g(n) for n ¿ n0
- Let c = 5 and n_0 = 1

Class Objectives were:

- ☐ Know the definition of big-Oh notation with big-Omega and big-Theta
- ☐ Can perform asymptotic analysis

Next Time

☐ Stacks and queues

☐ HW:

- Go over the next lecture slides before the class
- Just 10 min ~ 20 min should be okay

Any Questions?

- ☐ Come up with one question on what we have discussed in the class and submit at the end of the class
  - 1 for typical questions
  - 2 for questions with thoughts or that surprised me

- ☐ Write questions at least 4 times

- ☐ You can type KLMS