CS3230

Lecturer: Sanjay Jain (COM2 03–59; sanjay@comp.nus.edu.sg)
Tutor: Li Xiaohui (COM1 01–11; lixiaohui@comp.nus.edu.sg)
Web Page: http://www.comp.nus.edu.sg/∼sanjay/cs3230.html
Text Book: R. Johnsonbaugh and M. Schaefer — Algorithms.
Tutorials:

- start next week.
- Questions for tutorial will be given in class (previous week). Please hand in tutorials on monday (lecture time) in the week of the tutorial. Please mention your tutorial group in the submission.

Prerequisites
CS1101, CS1102, CS1231
Basically need to know substantial programming, data structures, discrete mathematics, mathematical maturity (to follow proofs, come up with and write proofs etc).

Grading:

- Tutorial Assignments and Class Participation (10 %)
- 1 midterm (30 %) (date: 15 October 2010, 6PM)
- 1 final (60 %)

Acknowledgements:
Thanks to Yu Haifeng and David Hsu, who taught this course in previous years, for some useful course material.

Some Major Topics/Techniques we will cover

- Divide and Conquer
- Greedy Algorithms
- Dynamic Programming
- P vs NP

This lecture is based on topics covered in the textbook in:
Chapter 1.1, 2.1, 2.2, 2.3, 2.4
You should read on your own background materials in chapter 2.5, 2.6 and all of chapters 3 and 4. (These topics were covered in the prerequisite courses)

Algorithms

- Input
- Output
- Unambiguous and Executable: The steps (instructions) are precisely and unambiguously stated; Instructions can be "executed", followed by a "machine", clear how to implement them
- *Deterministic: Execution of each step is unique and gives a result determined only by the input to the step.

  [Randomness, nondeterminism, ..., Quantum]
- Termination: The algorithm terminates after finitely many instructions are executed
- Generality: Applicable to a wide range of inputs
- Correctness:

- Performance/Efficiency/Complexity: Time and Space (Memory)
  (Depends on input size and content, machine speed, ...)
- Number of Basic Operations (correlated with time)
- Asymptotic Analysis
- Worst Case
- Average Case

Pseudocode

We can give a program. This is usually unambiguous and clear.
However, this is tedious. Sometimes difficult to understand.
Pseudocode.
Slightly informal.
Still precise enough to understand exactly what steps are taken.

Example: Finding maximum value in an array

Input: Array A ($A[1], A[2], \ldots, A[n]$)
Output: Largest element in the array.
Algorithm:

Findlargest(A)

$m = A[1]$.
For $i = 2$ to n {
If $A[i] > m$, then $m = A[i]$
}

End

Number of operations:
$n - 1$ comparisons, n assignments (worst case)
$n - 1$ assignments to the loop variable, ...
Roughly takes time proportional to n or about $C * n$.

Analysis of Algorithms

- Induction
- Recurrence Relations
- Mathematical Tools
- Invariants, Loop Invariants

Some Basic notations and results

- log, lg (base 2), ln (base e=2.718....)
- $\log_b a = n$ such that $b^n = a$.
- $\log_b a = \frac{\log a}{\log b}$
- $\log \frac{x}{y} = \log x - \log y$
- open interval (a, b), semi-open interval $(a, b]$ and closed interval $[a, b]$.
- Alphabet $\Sigma = \{a, b, \ldots\}$
- Strings (word) over the alphabet $ababba$,
- null string ϵ
- Sequences
- Boolean expressions: A and (B or $\neg C$)

- binomial coefficients:
  $(x+y)^n = \binom{n}{0}x^0y^n + \binom{n}{1}x^1y^{n-1} + \ldots \binom{n}{i}x^iy^{n-i} + \ldots + \binom{n}{n}x^ny^0$.
  Also use the notation ${}^nC_i = \binom{n}{i}$
- $\frac{b^{n+1}-a^{n+1}}{b-a} = \Sigma_{i=0}^{n} a^i b^{n-i}$

Consider $(b-a) * \Sigma_{i=0}^{n} a^i b^{n-i}$.
We get,
$(\Sigma_{i=0}^{n} a^i b^{n+1-i}) - (\Sigma_{i=0}^{n} a^{i+1} b^{n-i})$.
$= b^{n+1} + (\Sigma_{i=1}^{n} a^i b^{n+1-i}) - (a^{n+1} + \Sigma_{i=0}^{n-1} a^{i+1} b^{n-i})$.
$= b^{n+1} - a^{n+1} + (\Sigma_{i=1}^{n} a^i b^{n+1-i}) - (\Sigma_{i=1}^{n} a^i b^{n+1-i})$.
$= b^{n+1} - a^{n+1}$

Thus, if $0 \leq a < b$, then

$\frac{b^{n+1}-a^{n+1}}{b-a} = \Sigma_{i=0}^{n} a^i b^{n-i} \leq (n+1)b^n$

- Limit, Infimum and Supremum of a series

  $\lim_{n\to\infty} f(n)$
  $\lim_{m\to\infty} \min\{f(n) : n \geq m\}$
  $\lim_{m\to\infty} \max\{f(n) : n \geq m\}$

- Lower Bounds and Upper Bounds

  Worst Case:

  Lower Bound $f(n)$: For each n, for some input of size n, algorithm takes time at least $f(n)$.

  Upper Bound $f(n)$: For each n, for all inputs of size n, algorithm takes time at most $f(n)$.

- Asymptotic Upper Bounds

$f \in O(g)$, $f \in \Omega(g)$, $f \in \Theta(g)$.
f and g are functions from natural numbers to natural numbers.

$f \in O(g)$, iff there exist constants $c_1 > 0$ and c_2 such that
$f(n) \leq c_1 g(n) + c_2$ for all n.

Note: if $f \in O(g)$ and $g(n) > 0$, for all but finitely many n, then
for some constants c_1', c_3
$f(n) \leq c_1' g(n)$ for all $n \geq c_3$.

– Example:
$f(n) = 50n^2 + 200n$.
$g(n) = n^3$.
Then $f(n) \leq 51g(n) + 40000$
Thus, $f(n) \in O(g(n))$.

- Asymptotic Lower Bounds:

$f \in \Omega(g)$, iff there exist constants $c_1 > 0$ and c_2 (need not be ≥ 0) such that
$f(n) \geq c_1 g(n) + c_2$

- Asymptotic tight bounds

$f \in \Theta(g)$ iff $f \in O(g)$ and $f \in \Omega(g)$.

– Example
$f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$
Then, $f(n) \in \Theta(n^k)$.

Theorem: Suppose $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$.
(a) If $c = 0$, then $f(n) \in O(g(n))$, but $f(n) \notin \Omega(g(n))$.
(b) If $c = \infty$, then $f(n) \in \Omega(g(n))$, but $f(n) \notin O(g(n))$.
(c) If $0 < c < \infty$, then $f(n) \in \Theta(g(n))$.

Proof: By definition of limit, there exists an m such that for all $n > m$, $\frac{f(n)}{g(n)} \leq 1$.
Let $c_2 = \max\{f(n) : n \leq m\}$
Then, $f(n) \leq g(n) + c_2$.
Rest: Exercise

- L' Hospital's Rule
  $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$.
- Showing $f \in O(g)$
  - Explicit bounds as in definition
  - Using limit theorem

  You can use either, unless explicitly asked to use one of the methods ....
- Misuse notation and say $f(n) = O(g(n))$ etc.
- Recurrence relations

Example: Finding maximum value in an array

Input: Array A $(A[1], A[2], \ldots, A[n])$
Output: Largest element in the array.
Algorithm:

Findlargest(A[1:n])

If $n = 1$, then return $A[1]$.

Else,

let $m = Findlargest(A[1 : n-1])$.
If $m > A[n]$, then return m
Else return $A[n]$.

End

$T(n) \leq T(n-1) + C_1$.
$T(1) = C_2$

FindElement(A, i, j, key)

If $i > j$, return 0

Let $k = \frac{i+j}{2}$.

If $A[k] = key$, then return 1

Else, if $A[k] < key$, then return FindElement(A,k+1,j,key)

Else return FindElement(A,i,k-1,key)

End

$T(n) \leq T(n/2) + C.$

Some further recurrence examples:

$T(n) = T(n-1) + n$ for $n \geq 2$ and $T(1) = 1$

$T(n) = 2T(\lceil n/2 \rceil) + n$, for $n \geq 2$, and $T(1) = 1$.

$T(n) = 2T(n-1) + 1$, for $n \geq 2$, and $T(1) = 1$.

$T(n) = 9 + \Sigma_{i=1}^{n-1} T(i)$, for $n \geq 2$, and $T(1) = 7$.

Solve $T(n) = T(n-1) + 1$, for $n \geq 2$
$T(1) = 1$.

$$\begin{aligned} T(n) &= T(n-1) + 1 \\ &= T(n-2) + 1 + 1 \\ &= T(n-2) + 2 \\ &= T(n-3) + 3 \\ &= \cdots \\ &= T(1) + n - 1 \\ &= n \end{aligned}$$

$T(n) = T(n/2) + n$, for $n \geq 2$
$T(1) = 1$
Solve $T(n)$ for n being power of 2.

$$
\begin{aligned}
T(n) &= T(n/2) + n \\
&= T(n/4) + n/2 + n \\
&= T(n/8) + n/4 + n/2 + n \\
&= \cdots \\
&= T(1) + 2 + 4 + \cdots + n/4 + n/2 + n \\
&= 1 + 2 + 4 + \cdots + n/4 + n/2 + n \\
&= 2n - 1
\end{aligned}
$$

Need not want exactly, but only in order.
$T(n) = n + T(n-1), T(1) = 1$
$T(n) = n + (n-1) + (n-2) + \ldots + 1$
$T(n) \leq n * n = O(n^2)$.

Solving Recurrence Relations by Induction

- Guess the answer
- Prove by induction

Example:
$T(n) = T(n/3) + T(2n/3) + n$
$T(n) \leq C_2 n \log n$

Main Recurrence Theorem (Master Theorem)
– Useful for solving many cases (does not always work).
Theorem: Suppose $a \geq 1$, $b \geq 2$ are constants.
Upper Bound:
If $T(n) \leq aT(n/b) + f(n)$, where $f(n) = O(n^k)$, then

$$T(n) = \begin{cases} O(n^k), & \text{if } a < b^k; \\ O(n^k \log n), & \text{if } a = b^k; \\ O(n^{\log_b a}), & \text{if } a > b^k; \end{cases}$$

Lower Bound:
If $T(n) \geq aT(n/b) + f(n)$, where $f(n) = \Omega(n^k)$, then

$$T(n) = \begin{cases} \Omega(n^k), & \text{if } a < b^k; \\ \Omega(n^k \log n), & \text{if } a = b^k; \\ \Omega(n^{\log_b a}), & \text{if } a > b^k; \end{cases}$$

Exact Bound:
If $T(n) = aT(n/b) + f(n)$, where $f(n) = \Theta(n^k)$, then

$$T(n) = \begin{cases} \Theta(n^k), & \text{if } a < b^k; \\ \Theta(n^k \log n), & \text{if } a = b^k; \\ \Theta(n^{\log_b a}), & \text{if } a > b^k; \end{cases}$$

The theorem also holds if $b > 1$.

Proof Idea:
Consider the case of $n = b^r$ for some natural number r. Let $b^k = d$ in the following.

$$
\begin{aligned}
T(b^r) &\leq aT(b^{r-1}) + c[(b^r)^k] \\
T(b^r) &\leq aT(b^{r-1}) + c[d^r] \\
&\leq a[aT(b^{r-2}) + c[d^{r-1}]] + c[d^r] \\
&\leq a^2T(b^{r-2}) + c[d^r + a^1d^{r-1}] \\
&\quad \cdots \\
&\leq a^rT(b^{r-r}) + c[\sum_{i=1}^{r}[a^{r-i}d^i]] \\
&\leq c'[\sum_{i=0}^{r}[a^{r-i}d^i]]
\end{aligned}
$$

If $d > a$, then

$$\begin{aligned} T(b^r) &\leq c' \frac{d^{r+1} - a^{r+1}}{d - a} \\ &= O(d^r) = O(n^k) \end{aligned}$$

If $d < a$, then

$$\begin{aligned} T(b^r) &\leq c' \frac{a^{r+1} - d^{r+1}}{a - b} \\ &= O(a^r) = O(b^{r \log_b a}) = O(n^{\log_b a}) \end{aligned}$$

If $d = a$, then

$$\begin{aligned} T(b^r) &\leq c'[(r+1) * d^r] \\ &= O(n^k \log n) \end{aligned}$$

Consider $H(n) = aH(n/b) + cn^k$; $H(1) = T(1)$.

Note that $T(n) \leq H(n)$.
$H(n)$ is increasing function.
The bound we obtained above would give,
$H(n) \leq ....$, for $n = b^r$.
If $b^{r-1} < n' < b^r$, then
$T(n') \leq H(n') \leq H(b^r) \leq .....$

For example, if we have $H(n) \leq c' * n^k$, for n of the form b^r,
then for $b^{r-1} < n' < b^r = n$, we have
$T(n') \leq H(n') \leq H(n) \leq c' * n^k \leq c' * b^k (n')^k \leq c''(n')^k$.

Example:
$T(n) = T(n/2) + n.$
$a = 1$, $b = 2$, $k = 1$
So $a < b^k$
$T(n) = O(n)$

Example: $T(n) = 2T(n/2) + n$
$a = 2, b = 2, k = 1$
So $a = b^k$
$T(n) = O(n \log n)$