CS 240 - Data Structures and Data Management **Spring 2017**

Lecture 3: May 9th, 2017

Lecturer: Taylor Smith *Notes By: Harsh Mistry*

3.1 Growth Rate Affect on Running Time

- Constant Complexity : $T(n) = c, T(2n) = c$
- Logarithmic Complexity : $T(n) = clogn, T(2n) = c$
- Linear complexity : $T(n) = cn, T(2n) = 2T(n)$
- $\theta(nlogn)$: $T(n) = cnlogn, T(2n) = 2T(n) + 2cn$
- Quadratic Complexity : $T(n) = cn^2, T(2n) = 4T(n)$
- Cubic Complexity : $T(n) = cn^3, T(2n) = 8T(n)$
- Exponential Complexity : $T(n) = c2^n, T(2n) = (T(n))^2/c$

3.2 Complexity v.s Running Time

- Suppose that algorithms A_1 and A_2 both solve some specified problem
- Suppose that the complexity of algorithm A_1 is lower than the complexity of algorithm A_2. Then for sufficiently large problem instances, A_1 will run faster A_2. However, for small problem instances, A_1 could be slower than A_2
- Now suppose that A_1 and A_2 have the same complexity. Then we cannot determine from this information which of A_1 or A_2 is faster; a more delicate analysis of the algorithm A_1 and A_2 is required.

Note : It is important not to try and make comparisons between algorithms using O-notation

3.3 Techniques for Order Notation

Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Suppose that

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)}$$

Then

$$f(n) \in \begin{cases} o(g(n)) & \text{if } L = 0 \\ \theta(g(n)) & \text{if } 0 < L < \infty \\ \omega(g(n)) & \text{if } L = \infty \end{cases}$$

3.4 Relationships between Order Notations

- $f(n) \in \theta(g(n)) \iff g(n) \in \theta(f(n))$
- $f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n))$
- $f(n) \in o(g(n)) \iff g(n) \in \omega(f(n))$

- $f(n) \in \theta(g(n)) \iff f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
- $f(n) \in o(g(n)) \iff f(n) \in O(g(n))$
- $f(n) \in o(g(n)) \iff f(n) \notin \Omega(g(n))$
- $f(n) \in \omega(g(n)) \iff f(n) \in \Omega(g(n))$
- $f(n) \in \omega(g(n)) \iff f(n) \notin O(g(n))$

3.5 Algebra of Order Notations

"Maximum" Rules : Suppose that $f(n) > 0$ and $g(n) > 0$ for all $n \geq n_0$. Then :

- $O(f(n) + g(n)) = O(max\{f(n), g(n)\})$
- $\theta(f(n) + g(n)) = \theta(max\{f(n), g(n)\})$
- $\Omega(f(n) + g(n)) = \Omega(max\{f(n), g(n)\})$

Transitivity : If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in O(h(n))$.

3.6 Summation Formulae

- Arithmetic Sequence

$$\sum_{i=0}^{n-1}(a + di) = na + \frac{dn(n-1)}{2} \in \theta(n^2) \text{ for } d \neq 0$$

- Geometric Sequence

$$\sum_{i=0}^{n-1} ar^i = \begin{cases} a\frac{r^n-1}{r-1} \in \theta(r^n) & \text{if } r > 1 \\ na \in \theta(n) & \text{if } r = 1 \\ a\frac{1-r^n}{1-r} \in \theta(1) & \text{if } 0 < r < 1 \end{cases}$$

- Harmonic Sequence

$$H_n = \sum_{i=1}^{n} \frac{1}{i} \in \theta(logn)$$

3.7 Techniques for Algorithm Analysis

- Use θ-bounds throughout the analysis and obtain a θ-bound for the complexity of the algorithm
- Prove a O-bound and a matching ω-bound separately to get a θ-bound.

3.8 Techniques for Loop Analysis

- Identify elementary operations that require constant time. Denoted $\theta(1)$ time
- The complexity of a loop is expressed as the sum of complexities of each iteration of the loop.
- Analyse independent loops separately, and then add the results (use "maximum rules" and simplify whenever possible).
- If loops are nested, start with the inner most loop and proceed outwards. In general, this kind of analysis requires evaluation of nested summations.