Comparing Growth Rates

→ ideally: 1.) we want data struct ops to run in time proportional to constant or log func.
2.) algorithms to run in linear or n-log-n time

→ order of best-to-worst growth rates:

1.) constant
2.) logarithmic
3.) linear
4.) n-log-n
5.) quadratic
6.) cubic
7.) exponential

Asymptotic Analysis

→ we often focus on growth rate as big picture; it is often enough to know the running time grows proportionally to n.

→ we can perform analysis of an algorithm by estimating the # of primitive operations executed up to a specific factor

Big-Oh Notation

→ let $f(n)$ and $g(n)$ be functions mapping integers to real #'s.
We say that $f(n)$ is $O(g(n))$ if there is a real $c > 0$ and integer $n_0 \geq 1$ such that

$$f(n) \leq cg(n), \text{ for } n \geq n_0$$

Proposition:

→ if $f(n)$ is a polynomial of degree d, that is

$$f(n) = a_0 + a_1 n + \ldots + a_d n^d$$

and $a_d > 0$, then $f(n)$ is $O(n^d)$

Examples:

1.) $5n^2 + 3n\log n + 2n + 5 \rightarrow O(n^2)$
→ $5n^2 + 3n\log n + 2n + 5 \leq (5+3+2+5)n^2 = cn^2$, for $c = 15$ when $n \geq n_0 = 1$

2.) $20n^3 + 10n\log n + 5 \longrightarrow O(n^3)$
→ $20n^3 + 10n\log n + 5 \leq 35n^3$, for $n \geq 1$

3.) $2^{n+2} \longrightarrow O(2^n)$
→ $2^{n+2} = 2^n \cdot 2^2 = 2^n \cdot 4$; hence, $c = 4$ and $n_0 = 1$

4.) $2n + 100\log n \longrightarrow O(n)$
→ $2n + 100\log n \leq 102n$, for $n \geq n_0 = 1$; hence, we can take $c = 102$