CISC 320 Introduction to Algorithms
Fall 2005

Measuring algorithm performance
Big O

Measure the time complexity

- What is the unit of measurement?
  - In terms of seconds
    - a faster CPU takes less time; needs to run algorithms on the same machines for fair comparisons.
  - In terms of machine instructions, i.e., instructions/second (MIPS)
    - hard to estimate number of machine instructions based on psuedo code, also compiler dependent
  - In terms of lines of (pseudo) code executed
  - In terms of number (or cost) of ***basic*** operations
    - What is considered ***basic***?
      - RAM model of computation (see pp. 21, CLRS 2.2)
- How does the "time" depend on the input and input size?
  - Large size will take longer time
  - Same size, some input will take longer time than others
    - Worst-case
    - Best-case
    - Average-case

Example: Sequential search of an unordered array E

```
int seqSearch (int[] E, int n, int k)
1   int ans, index;
2   ans = -1;
3   for (i = 0; i < n; i++) {
4        if(k == E[i])
5           ans = i;
6           break;
7   return ans;
```

Overhead of running line 1,2, and 7

Worst-case: $T_{wc}(n) = 3 + 2n$
Best-case: $T(n) = 4$
Average-case: ???

Average case:

If I is an instance of size n of the problem and D is the set of all distinct possible instances of size n, then the average case or the expected time is

$$T_{avg}(n) = \Sigma_{I \in D} \Pr(I)T(I)$$

S_i is the event that E[i] = k

$T_{succ}(n) = \Sigma_{0 \le i \le (n-1)} \Pr(S_i|succ)T(S_i)$
$= \Sigma_{0 \le i \le (n-1)} (1/n)(2i+4)$
$= 3 + n$

$T_{avg}(n) = \Pr(succ)\, T_{succ}(n) + \Pr(fail)\, T_{fail}(n)$
$= 50\% (3+n) + 50\% (3+2n)$
$= 3 + (3/2)n$

Lessons learned:

- Worst-case: the most commonly used
  - worst-case gives the upper bound, i.e., guarantees the algorithm will never take any longer
  - worst-case actually happens quite often
  - Worst-case is not much worse than the average (the previous example is the case)
- Average: harder to analyze
- Best-case: not that useful

Big Oh, etc.

- *Upper bound:* $O(g(n))$ = { $f(n)$: there exist positive constant c and n_0 such that $0 \le f(n) \le c\, g(n)$ for all $n \ge n_0$}

  Meaning: g(n) grows faster up to a constant factor than any functions in O(g(n)).

- *Lower bound:* $\Omega(g(n))$ = { $f(n)$: there exist positive constant c and n_0 such that $0 \le c\, g(n) \le f(n)$ for all $n \ge n_0$ }

- *Tight bound:* $\Theta(g(n))$ = { $f(n)$: there exist positive constant c_1, c_2 and n_0 such that $0 \le c_1\, g(n) \le f(n) \le c_2\, g(n)$ for all $n \ge n_0$ }

Copyright © The McGraw-Hill Companies, Inc. Permission required for reproduction or display.

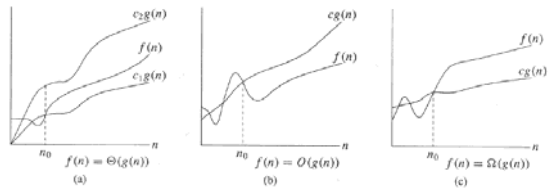

Figure 3.1 Graphic examples of the Θ, O, and Ω notations. In each part, the value of n_0 shown is the minimum possible value; any greater value would also work. **(a)** Θ-notation bounds a function to within constant factors. We write $f(n) = \Theta(g(n))$ if there exist positive constants n_0, c_1, and c_2 such that to the right of n_0, the value of $f(n)$ always lies between $c_1 g(n)$ and $c_2 g(n)$ inclusive. **(b)** O-notation gives an upper bound for a function to within a constant factor. We write $f(n) = O(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or below $cg(n)$. **(c)** Ω-notation gives a lower bound for a function to within a constant factor. We write $f(n) = \Omega(g(n))$ if there are positive constants n_0 and c such that to the right of n_0, the value of $f(n)$ always lies on or above $cg(n)$.

- Examples:
  - $3n^2 + 5n + 24 = \Theta(n^2)$

Proof: By definition, we need to find c_1, c_2, and n_0 such that for all $n \geq n_0$

$$c_1 n^2 \leq 3n^2 + 5n + 24 \leq c_2 n^2 \qquad (1)$$

Divide (1) by n^2,

$$c_1 \leq 3 + 5/n + 24/n^2 \leq c_2 \qquad (2)$$

The Inequalities in (2) hold for all $n > 5$, $c_1 = 3$, and $c_1 = 5$.

QED

Properties of Asymptotic Notation and Comparison of Functions

- Transitivity
  - $f \in O(g)$, $g \in O(h) \Rightarrow f \in O(h)$;
  - $f \in \Theta(g)$, $g \in \Theta(h) \Rightarrow f \in \Theta(h)$;
  - $f \in \Omega(g)$, $g \in \Omega(h) \Rightarrow f \in \Omega(h)$.
- Reflexivity
  - $f \in O(f(n))$; $f \in \Theta(f(n))$; $f \in \Omega(f(n))$.
- Symmetry
  - $f \in O(g) \Leftrightarrow g \in \Omega(f)$
  - $f \in \Theta(g) \Rightarrow g \in \Theta(f)$
  - $O(f+g) = O(\max(f,g))$

"little-oh of g of n"

$o(g(n))$ = { $f(n)$: for any positive constant *c, there exists a constant n_0* such that *$0 \leq f(n) \leq c\, g(n)$* for all *$n \geq n_0$*}

<u>Meaning</u>:

- o(g(n)) contains functions in O(g(n)) excluding those in *$\Theta(g(n))$*.
- *f(n) becomes insignificant relative to g(n) as n approaches infinity:*

$$\lim_{n \to \infty} f(n)/g(n) = 0.$$

e.g., $2n = o(n^2)$, *but $2n^2 \neq o(n^2)$.*

e.g., $\lg(n) = o(\sqrt{n})$ (Proof by using L'Hopital rule)

Typical asymptotic complexities are

- *$\Theta(\lg(n))$*, logarithmic (sub-linear)
- *$\Theta(n)$*, linear
- *$\Theta(n \lg(n))$, n-log-n*
- *$\Theta(n^2)$*, quadratic
- *$\Theta(n^3)$*, cubic
- *$\Theta(2^n)$*, exponential

Note:

- *Logarithms:*
  - $\lg n = \log_2 n$;
  - $\ln n = \log_e n$;
  - $\log_a b = \log_c b / \log_c a$
  - *Under the big Oh, the base of a logarithm does not matter.*
- *Factorial n! grows faster than exponential*
  - $n! = o(n^n)$
  - $n! = \omega(2^n)$
  - $\lg(n!) = \Theta(n \lg(n))$
- *See Section 3.2 in CLRS for more details*