Analysis of Algorithmic Efficiency

Analysis Framework

- Two kinds of efficiency:
  - Time efficiency: How fast an algorithm runs
  - Space efficiency: Deals with extra memory space an algorithm requires
  - We often deal with time efficiency.

- Measuring an input's size:
  - Efficiency as a function of some parameter ***n*** indicating the algorithm's input size
    - ✓ Size of the list for the problems of sorting or searching
    - ✓ Degree of polynomial for the problem of evaluating a polynomial:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots.. + a_0$$

  - Choice of input-size parameter does matter in some situations
  - Operations of the algorithm can affect the choice
  - Size of inputs for algorithms involving properties of numbers, is expressed by the number ***b*** of bits in the ***n***'s binary representation: $b = \lfloor \log_2 n \rfloor + 1$

Time Efficiency: Units and Analyses

- Standard unit of time measurement - a second, a millisecond, and so on
  - Measure the running time of a program implementing the algorithm.
- Basic operation: the operation that contributes most towards the running time of the algorithm
  - Count the number of repetitions of the basic operation
- Mathematical (or theoretical) analysis of an algorithm's efficiency
  - Independent of specific inputs
  - Limited applicability.
- Empirical (or experimental) analysis of an algorithm's efficiency
  - Applicable to any algorithm
  - Results dependent on the particular sample of instances and the computer used.

Mathematical Analysis

- Time efficiency is analyzed by determining the number of repetitions of the *basic operation* as a function of *input size*.

input size

$$T(n) \approx c_{op} C(n)$$

running time

execution time for basic operation on a particular computer

number of times basic operation is executed

Examples: Input Size and Basic Operation

Problem	***Input size measure***	***Basic operation***
Search for key in list of n items	Number of items in list n	Key comparison
Multiply two matrices of floating point numbers	Dimensions of matrices	Floating point multiplication
Compute a^n	n	Floating point multiplication
Graph problem	#vertices and/or edges	Visiting a vertex or traversing an edge

Best-case, Average-case, Worst-case

For some algorithms efficiency depends on type of input:

- Worst case: ***W(n)*** – maximum over inputs of size ***n***
- Best case: ***B(n)*** – minimum over inputs of size ***n***
- Average case: ***A(n)*** – "average" over inputs of size ***n***
  - Number of times the basic operation will be executed on typical or random input
  - Based on some assumption about the probability distribution of all possible inputs of size ***n***.
- Amortized efficiency
  - Amortize high cost of some worst-case occurrence (for some single operation) over the entire sequence (of n such operations).

Example: Sequential Search

- *Problem:* Given a list of ***n*** elements and a search key ***K***, find an element equal to ***K***, if any.
- *Algorithm:* Scan the list and compare its successive elements with ***K*** until either a matching element is found (*successful search*) or the list is exhausted (*unsuccessful search*)
- Worst case: $C_{worst}(n) = n$
- Best case: $C_{best}(n) = 1$
- Average case: $C_{avg}(n) = \frac{p(n+1)}{2} + n(1-p)$
  - Probability of a successful search = ***p***
    - ✓ $p = 1$ for successful search and $C_{best}(n) = (n+1)/2$
    - ✓ $p = 0$ for unsuccessful search and $C_{best}(n) = n$

Types of Formulas for Basic Operation Count

- Exact formula

  e.g., $C(n) = n(n-1)/2$
- Formula indicating order of growth with specific multiplicative constant

  e.g., $C(n) \approx 0.5n^2$
- Formula indicating order of growth with unknown multiplicative constant

  e.g., $C(n) \approx cn^2$

Order of Growth

- Most important: Order of growth of the algorithm's efficiency within a constant multiple as $n \to \infty$
  - See table 2.1

- Examples:
  - How much faster will algorithm run on computer that is twice as fast?
    - ✓ Two times.

  - How much longer does it take to solve problem of double input size?
    - ✓ The function $\log_2 n$ increases in value by 1:
      $\log_2 2n = \log_2 2 + \log_2 n = 1 + \log_2 n$
    - ✓ The linear function increases twofold: $2n$
    - ✓ The cubic function increases eightfold: $(2n)^3 = 8n^3$
    - ✓ The value for the 2^n function is squared: $2^{2n} = (2^n)^2$

Table 2.1

n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

Three Asymptotic Notations

- Principal indicator of efficiency = Order of growth of basic operation count
  - A way of comparing functions that ignores constant factors and small input sizes.
- **O(*g*(*n*)):** set of all functions ***f*(*n*)** with a smaller or same order of growth as ***g*(*n*)** (to within a constant multiple, as ***n*** → ∞).
- **Ω(*g*(*n*)):** set of all functions ***f*(*n*)** with a larger or same order of growth as ***g*(*n*)** (to within a constant multiple, as ***n*** → ∞).
- **Θ(*g*(*n*)):** set of all functions ***f*(*n*)** with the same order of growth as ***g*(*n*)** (to within a constant multiple, as ***n*** → ∞).

Big-Oh Notation

- ***t*(*n*)** ∈ **O(*g*(*n*)):** A function ***t*(*n*)** is said to be in **O(*g*(*n*)),** if it is bounded above by some constant multiple of ***g*(*n*)** for all large ***n***
- There exist positive constant *c* and non-negative integer n_0 such that $t(n) \leq c\, g(n)$ for every $n \geq n_0$

Example: **100*n*+5** is **O(n^2)**

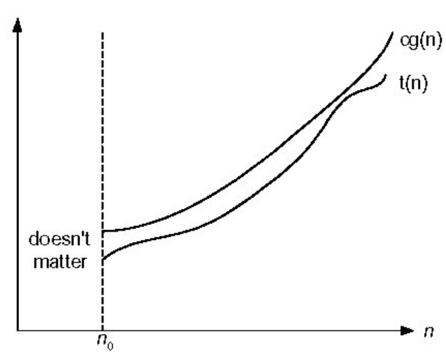

Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$

Big-Omega Notation

- $t(n) \in \Omega(g(n))$**:** A function $t(n)$ is said to be in **O(**$g(n)$**),** if it is bounded below by some constant multiple of $g(n)$ for all large n

- There exist positive constant c and non-negative integer n_0 such that

  $t(n) \geq c\, g(n)$ for every $n \geq n_0$

  Example: $2n^3$**-65** is $\Omega(n^2)$

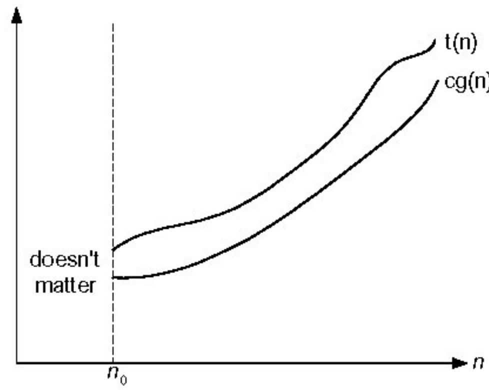

Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

Big-Theta Notation

- $t(n) \in \Theta(g(n))$**:** A function $t(n)$ is said to be in $\Theta(g(n))$**,** if it is bounded both above and below by some constant multiples of $g(n)$ for all large n

- There exist positive constants c_1 and c_2 and non-negative integer n_0 such that

  $c_2 g(n) \geq t(n) \geq c_1 g(n)$ for every $n \geq n_0$

  Example: (1/2)n(n**-1**) is $\Theta(n^2)$

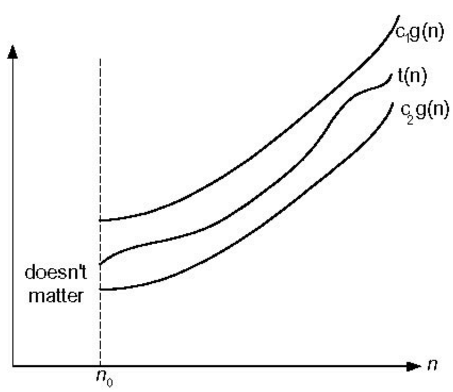

Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Comparing Growth Rate: Using Limits

- Compute the limit of the ratio of two functions under consideration
  - Using the limit-based approach is more convenient than the one based in the definition.

$$\lim_{n\to\infty} T(n)/g(n) = \begin{cases} 0 & \text{order of growth of } \boldsymbol{T(n)} < \text{order of growth of } \boldsymbol{g(n)} \\ C > 0 & \text{order of growth of } \boldsymbol{T(n)} = \text{order of growth of } \boldsymbol{g(n)} \\ \infty & \text{order of growth of } \boldsymbol{T(n)} > \text{order of growth of } \boldsymbol{g(n)} \end{cases}$$

  - Use calculus techniques such as L'Hôpital's Rule and Stirling's formula in computing the limits.

Examples:

$n(n+1)/2$	**vs.**	n^2
$\log_2 n$	**vs.**	$\sqrt{n}$
$n!$	**vs.**	2^n

Basic Asymptotic Efficiency Classes

1	**constant**
log *n*	**logarithmic**
n	**linear**
n* log *n	***n* log *n***
n^2	**quadratic**
n^3	**cubic**
2^n	**exponential**
$n!$	**factorial**

Increasing order of the order of growth

- Caution: In defining asymptotic efficiency classes, the values of multiplicative constants are usually left unspecified.

Empirical Analysis of Algorithms

- A complementary approach to mathematical analysis is empirical analysis of an algorithm's efficiency.
- A general plan for the empirical analysis involves the following steps:
  - Understand the purpose of the analysis process (called experimentation)
  - Decide on the efficiency metric to be measured and the measurement unit
  - Decide on characteristics of the input sample
  - Generate a sample of inputs
  - Implement the algorithm for its execution (to run computer experiment/simulation)
  - Execute the program to generate outputs
  - Analyze the output data.

Analyzing Output Data

- Collect and analyze the empirical data (for basic counts or timings)
- Present the data in a tabular or graphical form
- Compute the ratios $M(n)/g(n)$, where $g(n)$ is a candidate to represent the efficiency of the algorithm in question
- Compute the ratios $M(2n)/M(n)$ to see how the running time reacts to doubling of its input size
- Examine the shape of the plot:
  - ✓ A concave shape for the logarithmic algorithm
  - ✓ A a straight line for a linear algorithm
  - ✓ Convex shapes for quadratic and cubic algorithms
  - ✓ An exponential algorithm requires a logarithmic scale for the vertical axis.