

CSCI203

Algorithms and Data Structures



Algorithm Efficiency

Lecturer: Dr. Fenghui Ren

Room 3.203

Email: fren@uow.edu.au

Analysis of Algorithms



► Issues to be considered

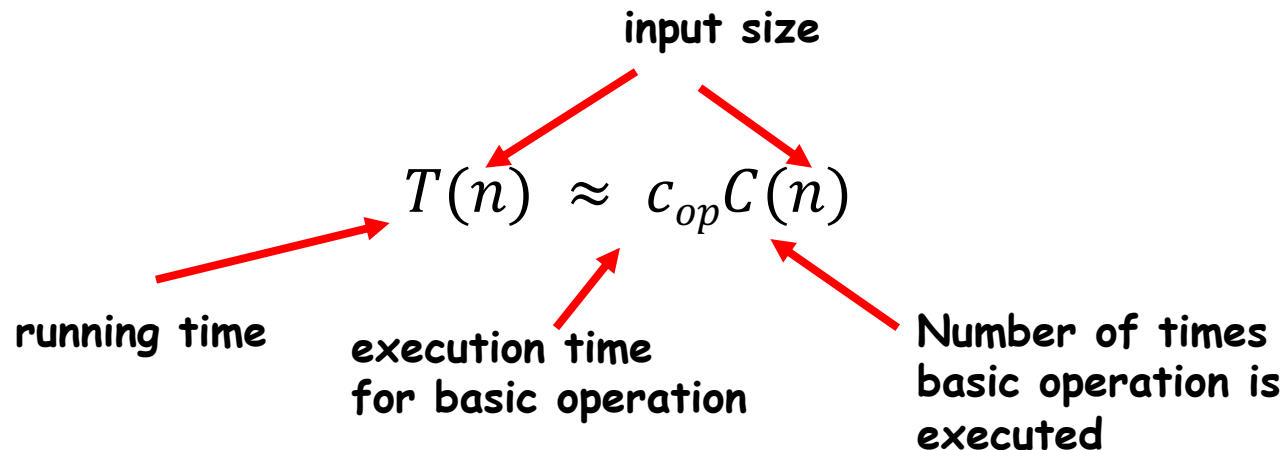
- Correctness
- Time efficiency
- Space efficiency
- Optimality

► Approaches

- Theoretical analysis
- Empirical analysis

Theoretical Analysis of Time Efficiency

- ▶ Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size
- ▶ Basic operation: the operation that contributes most towards the running time of the algorithm



Input Size and Basic Operations Examples

Problem	Input size measure	Basic operation
Searching for key in a list of n items	Number of list's items, i.e. n	Key comparison
Multiplication of two matrices	Matrix dimensions or total number of elements	Multiplication of two numbers
Checking primality of a given integer n	n 'size = number of digits (in binary representation)	Division
Typical graph problem	# vertices and/or edges	Visiting a vertex or traversing an edge

Empirical Analysis of Time Efficiency

- ▶ Select a specific (typical) sample of inputs
- ▶ Use physical unit of time (e.g., milliseconds)
or
Count actual number of basic operation's executions
- ▶ Analyze the empirical data

Best-case, average-case, worst-case

- ▶ For some algorithms efficiency depends on form of input:
 - Worst case: $C_{worst}(n)$ - maximum over inputs of size n
 - Best case: $C_{best}(n)$ - minimum over inputs of size n
 - Average case: $C_{avg}(n)$ - "average" over inputs of size n
 - Number of times the basic operation will be executed on typical input
 - NOT the average of worst and best case
 - Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs

Example: Sequential search

ALGORITHM *SequentialSearch*($A[0..n - 1]$, K)

//Searches for a given value in a given array by sequential search

//Input: An array $A[0..n - 1]$ and a search key K

//Output: The index of the first element of A that matches K

// or -1 if there are no matching elements

$i \leftarrow 0$

while $i < n$ **and** $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

▶ **Worst case**

▶ $C_{worst}(n) = n$

▶ **Best case ?**

▶ $C_{best}(n) = 1$

▶ **Average case**

▶ $C_{avg}(n) = \frac{p(n+1)}{2} + n(1 - p)$

where p is the probability of finding K in A

Types of formulas for basic operation's count

- ▶ Exact formula

- e.g., $C(n) = n(n - 1)/2$

- ▶ Formula indicating order of growth with specific multiplicative constant

- e.g., $C(n) \approx 0.5 n^2$

- ▶ Formula indicating order of growth with unknown multiplicative constant

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

Order of Growth

- ▶ Most important: Order of growth within a constant multiple as $n \rightarrow \infty$
 - e.g. How much longer does it take to solve problem of double input size?

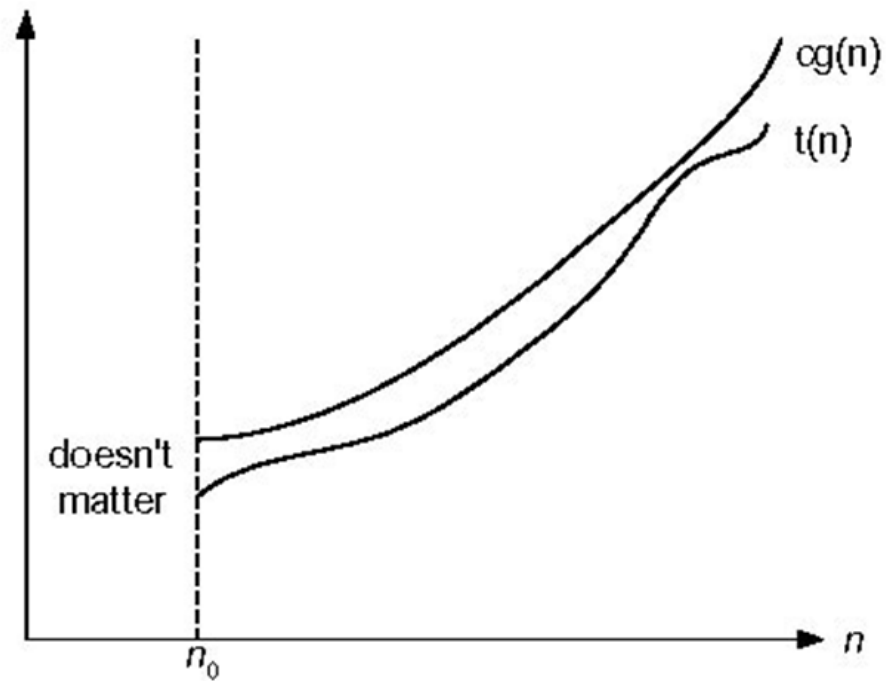
n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$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

Approximate values of several important functions for analysis of algorithms

Asymptotic Order of Growth

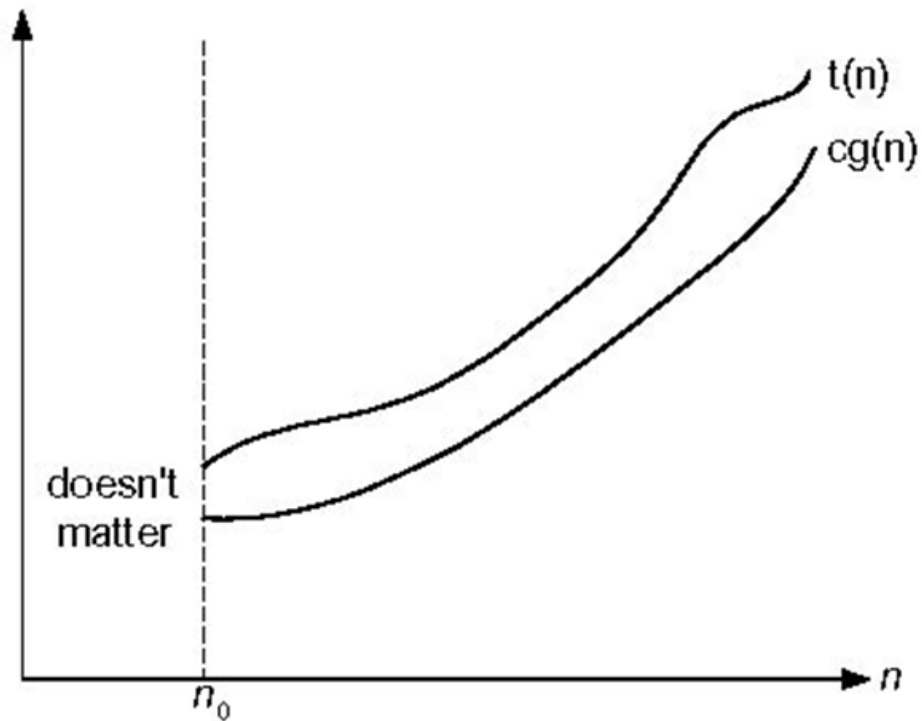
- ▶ A way of comparing functions that ignores constant factors and small input sizes
- ▶ $O(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$
 - Related to the worst-case behavior of the algorithm
- ▶ $\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$
 - Related to the best-case behavior of the algorithm

Big-O



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

Big-omega



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

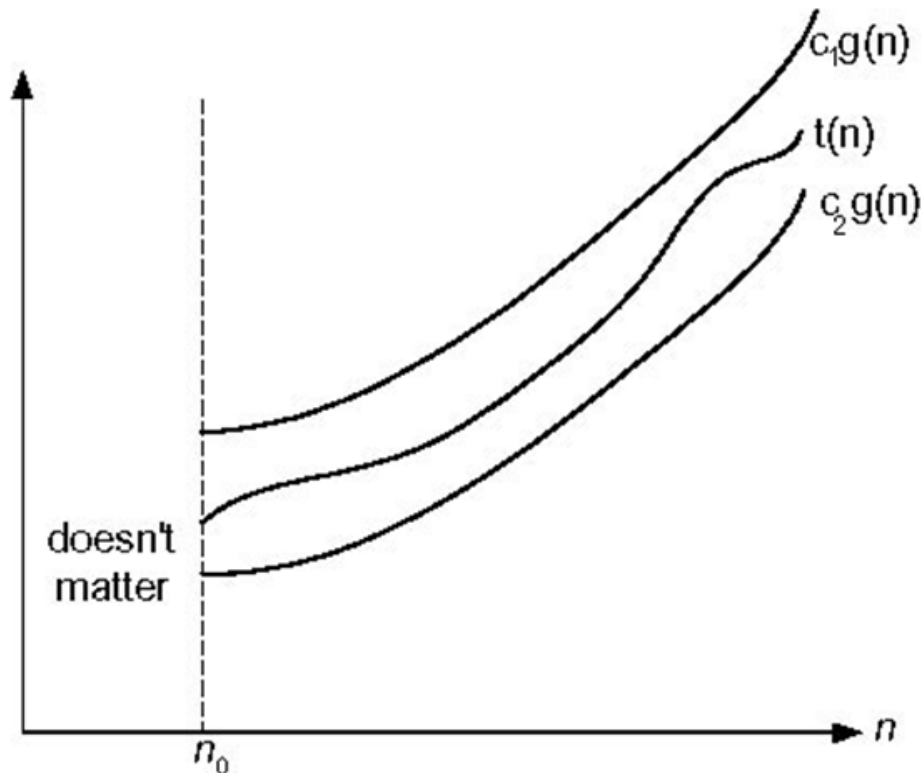
Relating O and Ω

- ▶ Both of O and Ω are loose bounds. A function can be in one of these and yet grow much more slowly (or quickly) than the bounding function.
 - E.g. $n^2 \in O(n!)$ and $n! \in \Omega(n^2)$
- ▶ A third notation allows us to create a tighter bound for some functions.

Theta, a tighter bound

- ▶ Sometimes, we can find a function $g(n)$ so that:
 - $f(n) \in O(g(n))$
 - $f(n) \in \Omega(g(n))$
- ▶ That is, the same function is both an upper bound and a lower bound.
- ▶ In this case we say that $f(n) \in \Theta(g(n))$
- ▶ $\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$
 - Related to the average or typical case behavior of the algorithm

Big-theta



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

Establishing order of growth using the definition

Definition: $f(n)$ is in $O(g(n))$ if order of growth of $f(n) \leq$ order of growth of $g(n)$ (within constant multiple),

- i.e., there exist positive constant c and non-negative integer n_0 such that

$$f(n) \leq c g(n) \text{ for every } n \geq n_0$$


► Examples:

- $10n$ is $O(n^2)$
- $5n + 20$ is $O(n)$

Some properties of asymptotic order of growth

- ▶ $f(n) \in O(f(n))$
- ▶ $f(n) \in O(g(n))$ iff $g(n) \in O(f(n))$
 - *That is to say that $f(n)$ is bounded below by $g(n)$ if and only if $g(n)$ is bounded above by $f(n)$.*
- ▶ If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$
- ▶ If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$, then
$$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$$

Establishing order of growth using limits


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

Examples:

- $T(n) = 10n$ vs. $g(n) = n^2$
- $T(n) = n(n+1)/2$ vs. $g(n) = n^2$

L'Hôpital's rule and Stirling's formula

- ▶ **L'Hôpital's rule:** If $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n)$ and the derivatives f', g' exist, then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

Example: $\log n$ vs. n

- ▶ **Stirling's formula:** $n! \sim \sqrt{2\pi n}(n/e)^n$

Example: 2^n vs. $n!$

An Example

Consider the following algorithm.

```
Algorithm Enigma( $A[1 \cdots n, 1 \cdots n]$ )  
//Input: A matrix of  $A[1 \cdots n, 1 \cdots n]$  real numbers  
for  $i \leftarrow 1$  to  $n - 1$  do  
    for  $j \leftarrow i + 1$  to  $n$  do  
        if  $A[i, j] \neq A[j, i]$   
            return false  
return true
```

- ▶ What does this algorithm compute?
- ▶ What is its basic operation?
- ▶ How many times is the basic operation executed?
- ▶ What is the efficiency class of this algorithm?

An Example...

- ▶ The algorithm returns "true" if its input matrix is symmetric and "false" if it is not.
- ▶ Comparison of two matrix elements.
- ▶
$$C_{worst}(n) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n-1} 1 = \sum_{i=1}^{n-1} [(n-1) - (i+1) + 1]$$
$$= \sum_{i=1}^{n-1} (n-1-i) = (n-1) + (n-2) + \dots + 1 = \frac{(n-1)n}{2} = \frac{n^2 - n}{2}$$
- ▶ Quadratic $C(n) \in O(n^2)$

Basic asymptotic efficiency classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n or linearithmic
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Related References



- ▶ Introduction to the Design and Analysis of Algorithms, A. Levitin, 3rd Ed., Pearson 2011.
 - Chapters 2.1 & 2.2
- ▶ Introduction to Algorithms, T. H. Cormen, 3rd Ed, MIT Press 2009.
 - Chapters 2 & 3