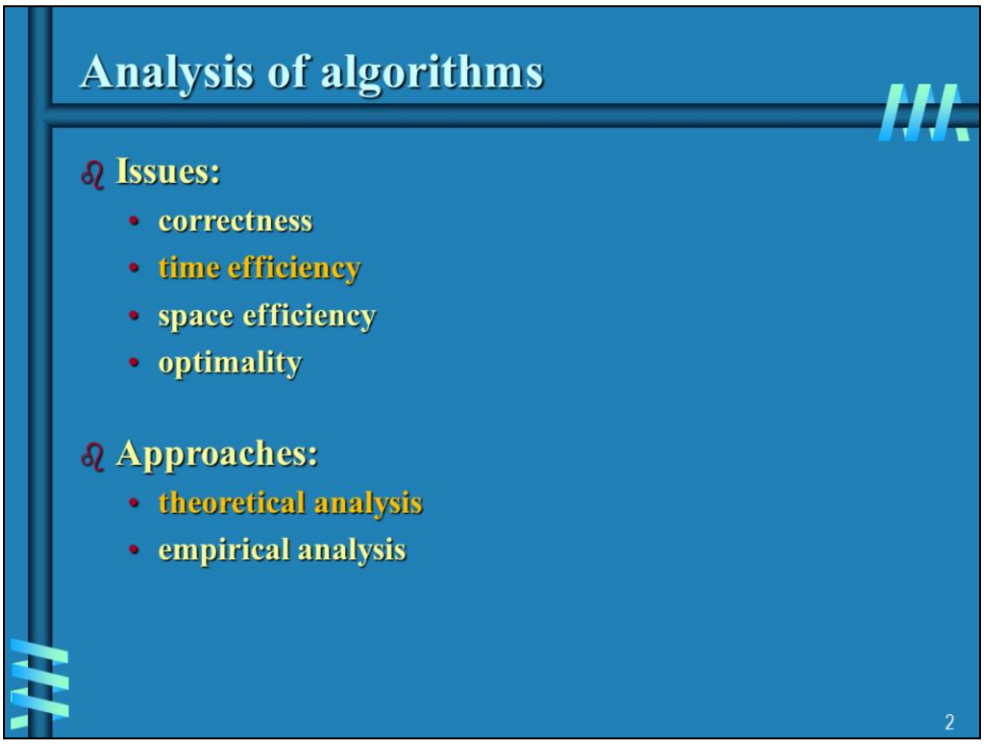
CS 395 – Analysis of Algorithms

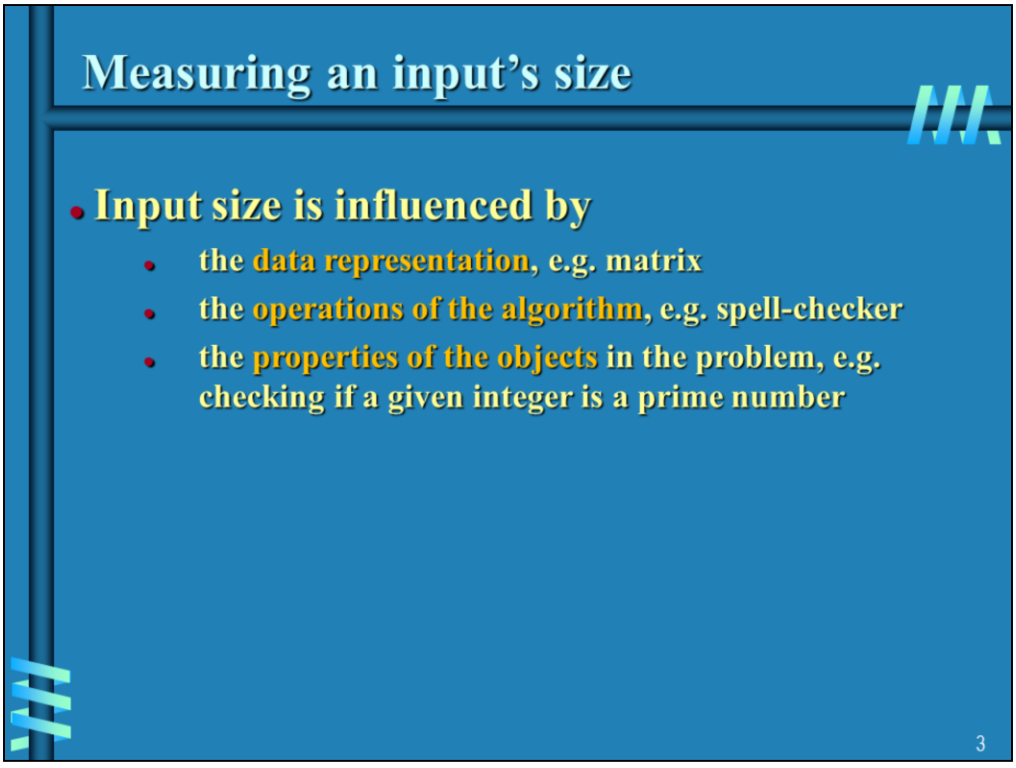
Chapter 2 – Fundamentals of the Analysis of Algorithm Efficiency
Read 2.1

- **Algorithm Analysis Framework.**
- **Weekly assignment 3**

When you have a good proof, it's because it's proven.
— Jean Chretien



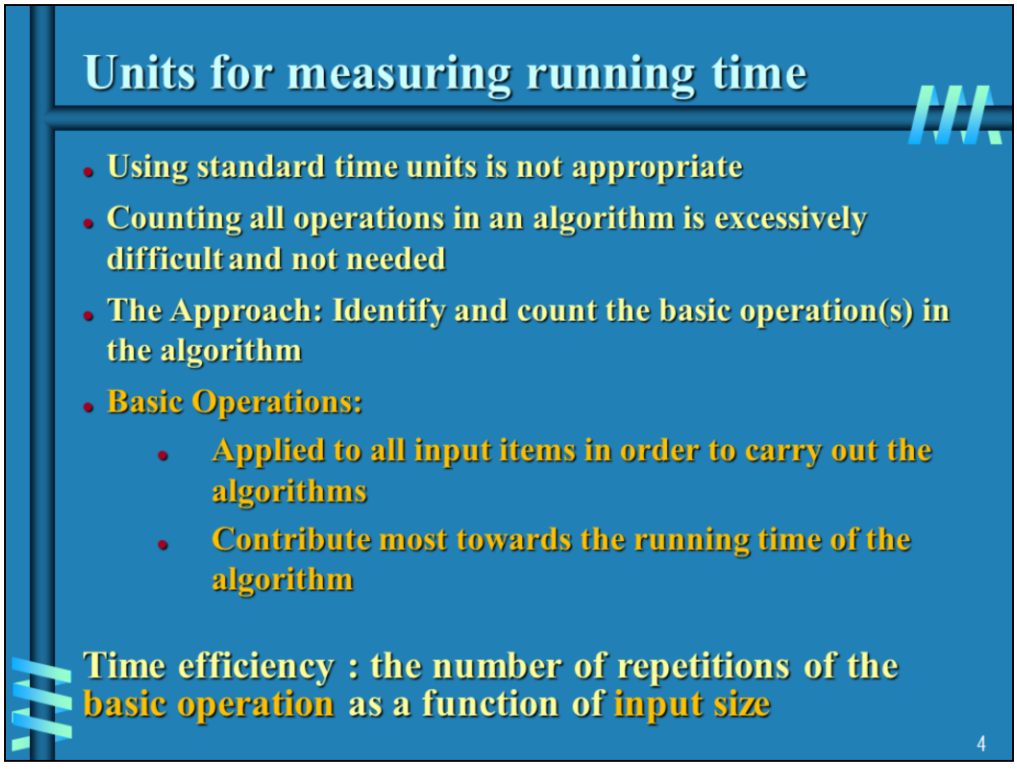
Empirical: based on, concerned with, or verifiable by observation or experience rather than theo



The choice of an appropriate size metric can be influenced by operations of the algorithm in ques

For example, how should we measure an input's size for a **spell-checking** algorithm? If the algor

We should make a special note about measuring input size for algorithms solving problems such a



Using standard time units is not appropriate

- If we measure in ms, what happens if we run it on a faster computer?
- What happens on a time sharing system when the program is waiting?
- What if the input size changes?
- What about start-up costs that only make a difference with small inputs?

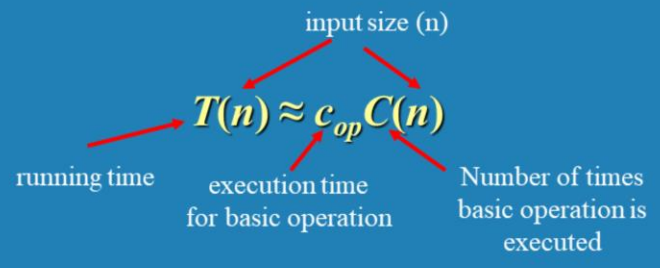
Input size and basic operation examples

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

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*

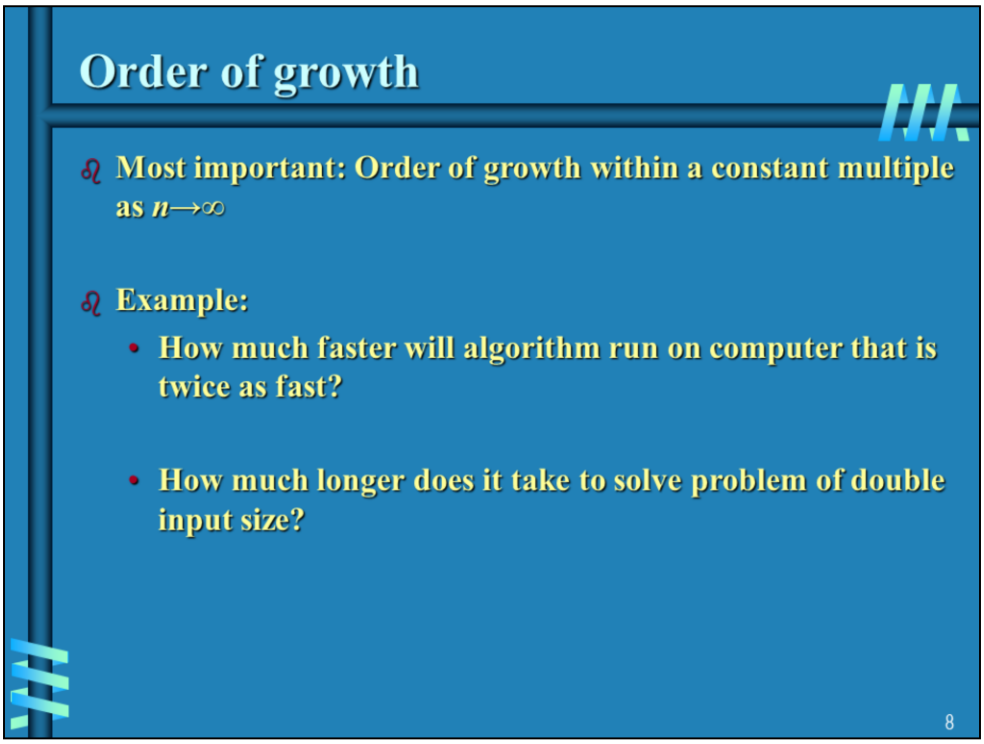
- ***Basic operation*: the operation that contributes most towards the running time of the algorithm**



6

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



Example: cn^2

$\Rightarrow$ how much faster on twice as fast computer? (2)

$\Rightarrow$ how much longer for $2n$? (4)

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 $g(n)$
- $\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$
- $\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$

Definition of Big-O

Definition 1: A function *t(n)* is said to be in *O(g(n))*, denoted t(n) ∈ O(g(n)), if *t(n)* is bounded above by some positive constant multiple of *g(n)* for all large *n*, i.e., if there exist some positive constant *c* and some nonnegative integer n_0 such that

$$t(n) \leq c\, g(n) \text{ for all } n \geq n_0.$$

Examples (find c and n_0 so the above formula holds true):

$10n^2 + 4n$ is $O(n^2)$

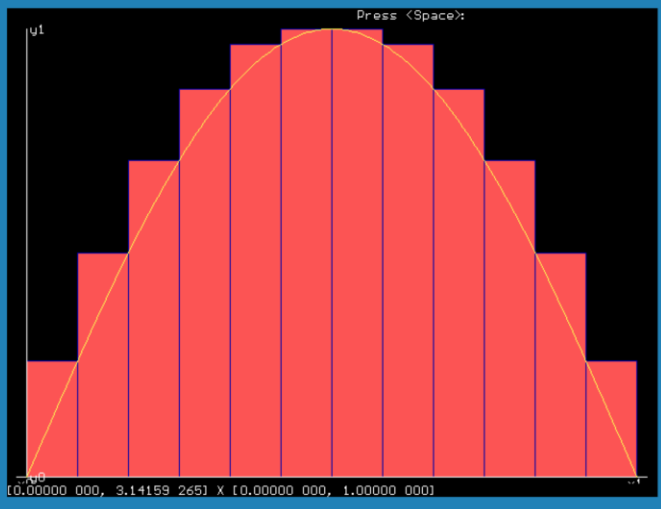
$5n+20$ is $O(n)$

10

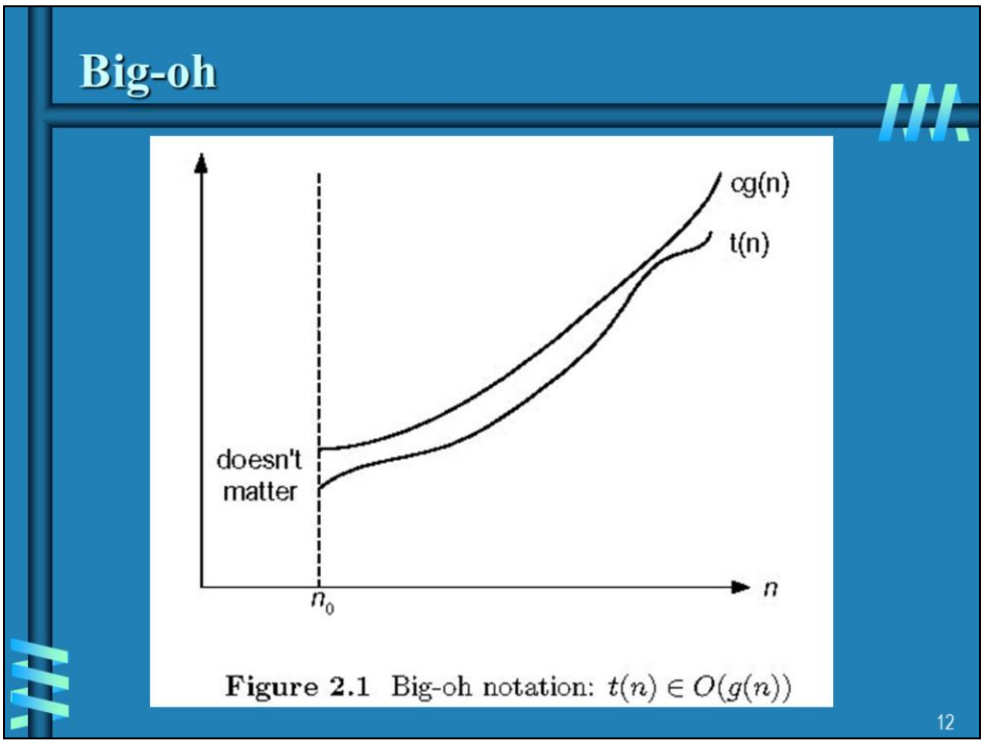
c takes care of the fater computer

n0 takes care of the startup costs.

The area in the red boxes is an upper bound area under the yellow curve.



11



It does not matter what happens before n0, but after that t(n) < Og(n)

Definition of Big-Ω

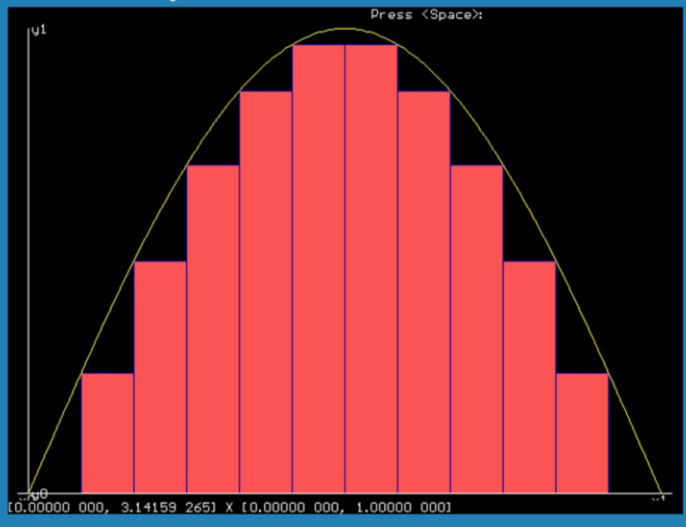
Definition: A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$, if $t(n)$ is **bounded below** by some positive constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c and some nonnegative integer n_0 such that

$$t(n) \geq c\, g(n) \text{ for all } n \geq n_0.$$

Examples:

- can we say n^3 is $\Omega(n^2)$?
- i.e., $n^3 \geq n^2$ with $c=1$, $n_0=0$

The area in the red boxes is a lower bound area under the yellow curve.



Picture from
http://archives.math.utk.edu/visual.calculus/4/riemann_sums.3/microcalc.html

14

Big-omega

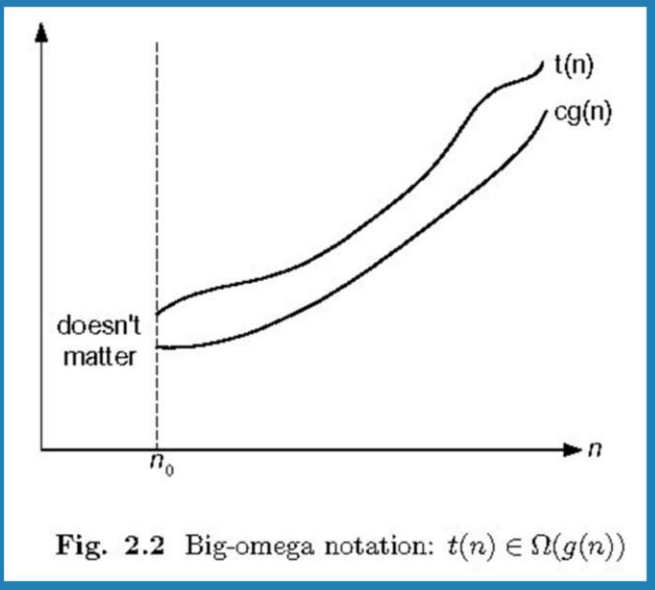


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$

Definition of Big-Θ

Definition: A function $t(n)$ is said to be in $\Theta(g(n))$, denoted $t(n) \in \Theta(g(n))$, if $t(n)$ is bounded both above and below by some positive constant multiple of $g(n)$ for all large n, i.e., if there exist some positive constant c_1 and c_2, and some nonnegative integer n_0 such that

$$c_2 g(n) \leq t(n) \leq c_1 g(n) \quad \text{for all } n \geq n_0.$$

Example:

- $\frac{1}{2} n(n-1)$ is $\Theta(n^2)$

Big-theta

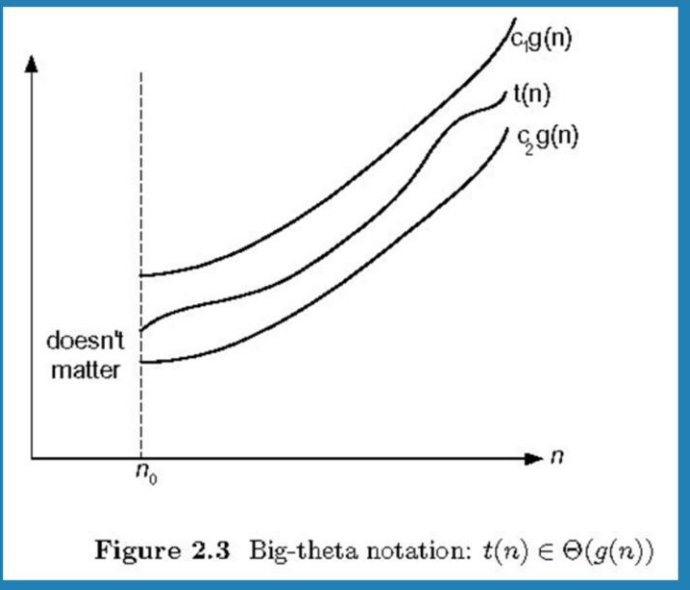


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$

Is $3x^2 + 2x + 4 \in \theta(x^2)$?

Assume that T, f and g are functions mapping the natural numbers {0, 1, 2, 3, ...} into the positive reals.

Definition: "Big Oh" A function T(n) is in O(f(n)) if there exist constants $n_0 \geq 0$, and $c > 0$, such that for all $n \geq n_0$, $T(n) \leq c * f(n)$.

Definition: "Omega" A function T(n) is in Ω(f(n)) if there exist constants $n_0 \geq 0$, and $c > 0$, such that for all $n \geq n_0$, $T(n) \geq c * f(n)$.

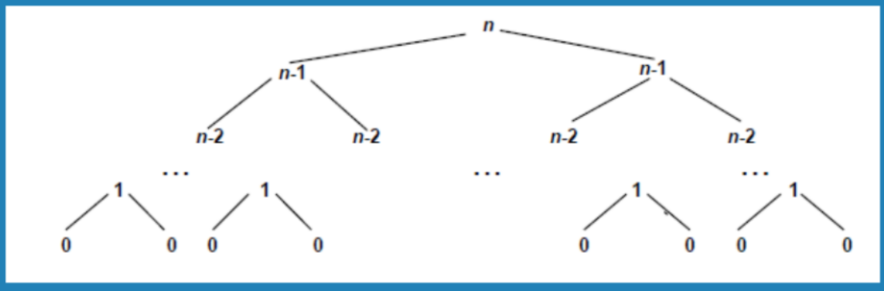
Definition: "Theta" The set θ(g(n)) of functions consists of $\Omega(g(n)) \cap O(g(n))$.

18

Do on the overhead.

Weekly Assignment 3

- Implement a recursive algorithm for computing 2^n for any nonnegative integer n that is based on the formula $2^n = 2^{(n-1)} + 2^{(n-1)}$.



Once you have written your code, think about how many times the function will be called for a given input 'n'. We will go over this next class

19

How many times is this function called for input n?