CS 2233-01 Data Structures and Algorithms

Chapter 4 Algorithm Analysis

Instructor: Dr. Qingguo Wang

College of Computing & Technology

Lipscomb University

How do we compare algorithms?

- We need to define a number of objective measures.

  (1) Compare execution times?
  Not good: times are specific to a particular computer !!

  (2) Count the number of statements executed?
  Not good: number of statements vary with the programming language as well as the style of the individual programmer.

Algorithm Analysis

- What is the goal of analysis of algorithms?
  - To compare algorithms mainly in terms of running time but also in terms of other factors (e.g., memory requirements, programmer's effort etc.)

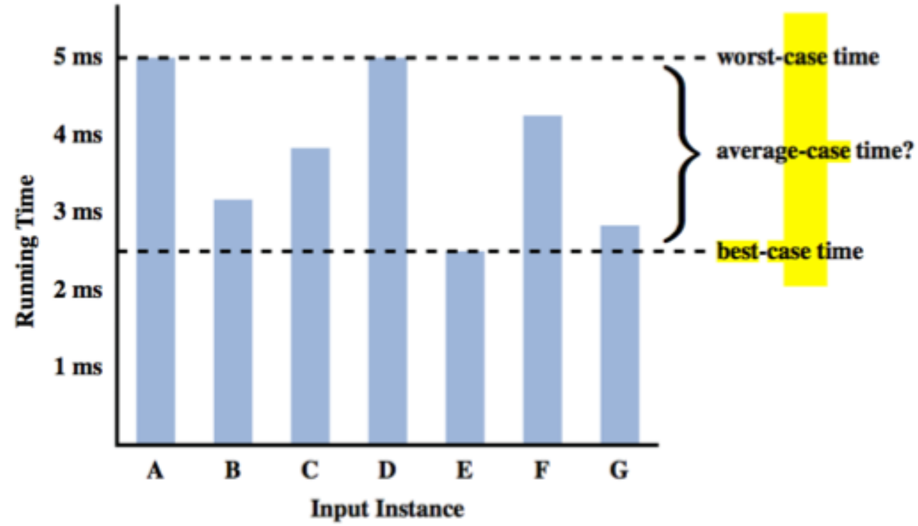
- What do we mean by running time analysis?
  - **Determine how running time increases as the *size* of the problem increases**.

Ideal Solution

- Express running time as a function of the input size n (i.e., $f(n)$).
- Compare different functions corresponding to running times.
- Such an analysis is independent of machine time, programming style, etc.

Analysis Framework: Types of Analysis

- Worst case
  - Provides an ***upper bound*** on running time
  - An absolute guarantee that the algorithm would not run longer, no matter what the inputs are
- Best case
  - Provides a ***lower bound*** on running time
  - Input is the one for which the algorithm runs the fastest



$$\boxed{\textit{Lower Bound} \leq \textit{Running Time} \leq \textit{Upper Bound}}$$

- Average case
  - Provides a ***prediction*** about the running time
  - Assumes that the input is random

Example

- Associate a ***cost*** with each statement.
- Find the ***total cost*** by finding the total number of times each statement is executed.

Algorithm 1	**Cost**	***Algorithm 2***	**Cost**
arr[0] = 0;	c_1	for(i=0; i<N; i++)	c_2
arr[1] = 0;	c_1	arr[i] = 0;	c_1
arr[2] = 0;	c_1		
...	...		
arr[N-1] = 0;	c_1		
	-----------		-------------
	$c_1+c_1+\ldots+c_1 = c_1 \text{ x } N$		$(N+1) \text{ x } c_2 + N \text{ x } c_1 =$ $(c_2 + c_1) \text{ x } N + c_2$

Asymptotic Analysis

- To compare two algorithms with running times *f(n)* and *g(n),* we need a **rough measure** that characterizes **how fast each function grows.**

- *Hint:* use *rate of growth*

- Compare functions in the limit, that is, **asymptotically!**
  (i.e., for large values of *n*)

Rate of Growth

- The low order terms in a function are relatively insignificant for **large** n

$$n^4 + 100n^2 + 10n + 50 \quad \sim \quad n^4$$

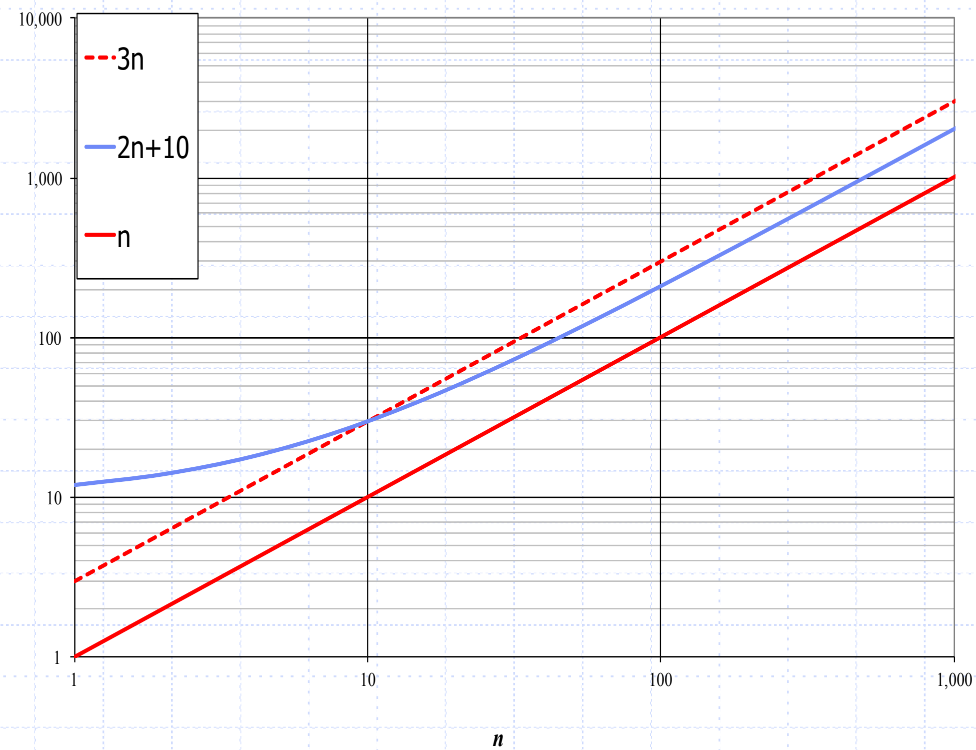
i.e., we say that $n^4 + 100n^2 + 10n + 50$ and n^4 have the same **rate of growth**

Asymptotic Notation (Big-Oh Notation)

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

  $f(n) \leq cg(n)$ for $n \geq n_0$

- Example: Can you prove that $2n + 10$ is $O(n)$?

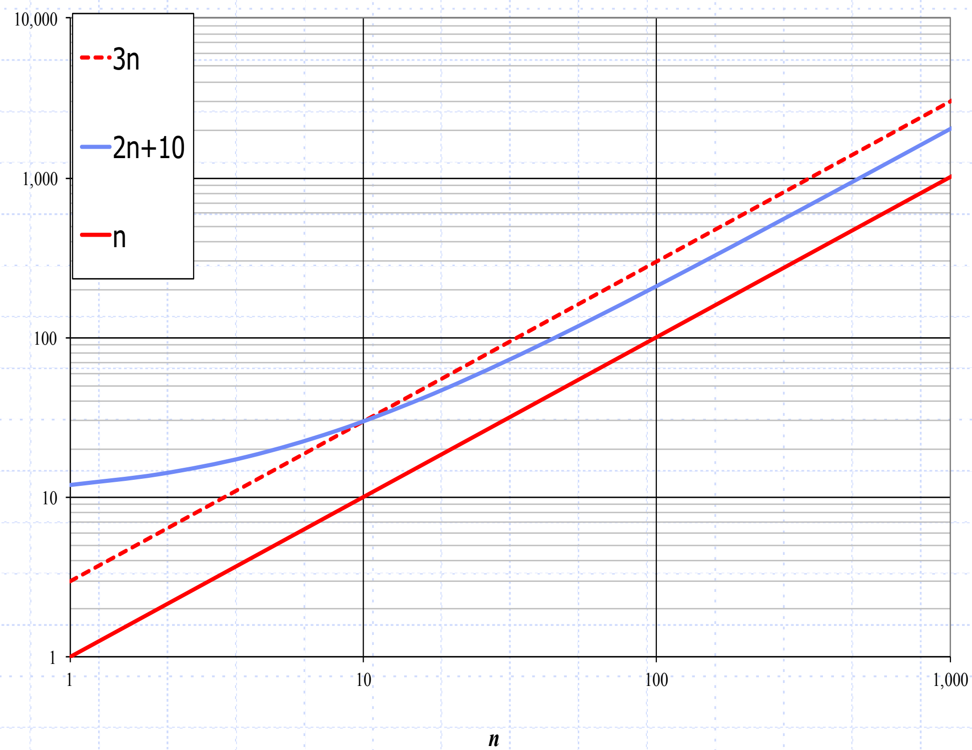


Big-Oh Notation

- Given functions $f(n)$ and $g(n)$, we say that $f(n)$ is $O(g(n))$ if there are positive constants c and n_0 such that

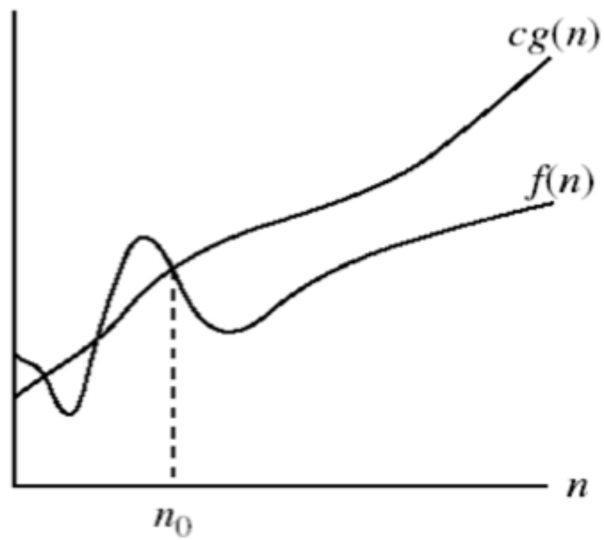
  $f(n) \leq cg(n)$ for $n \geq n_0$

- Example: $2n + 10$ is $O(n)$
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Pick $c = 3$ and $n_0 = 10$



Asymptotic notations

- *O-notation*

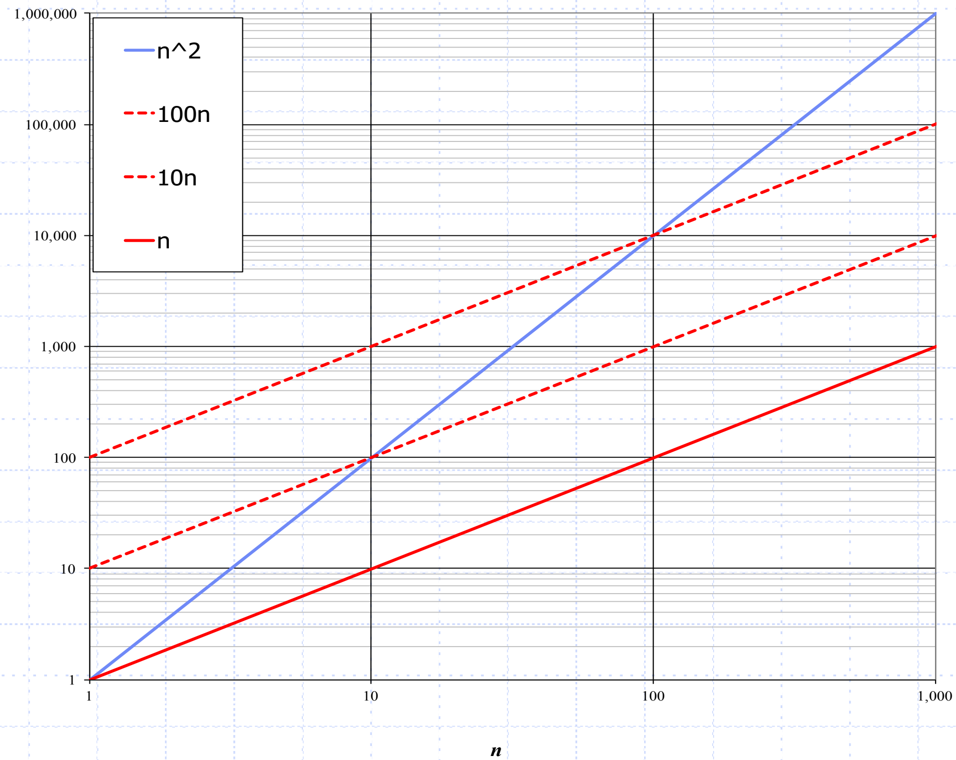


$g(n)$ is an ***asymptotic upper bound*** for $f(n)$.

O(g(n)) is the set of functions with smaller or same order of growth as g(n)

Big-Oh Example

- Can you prove the function n^2 is not $O(n)$?



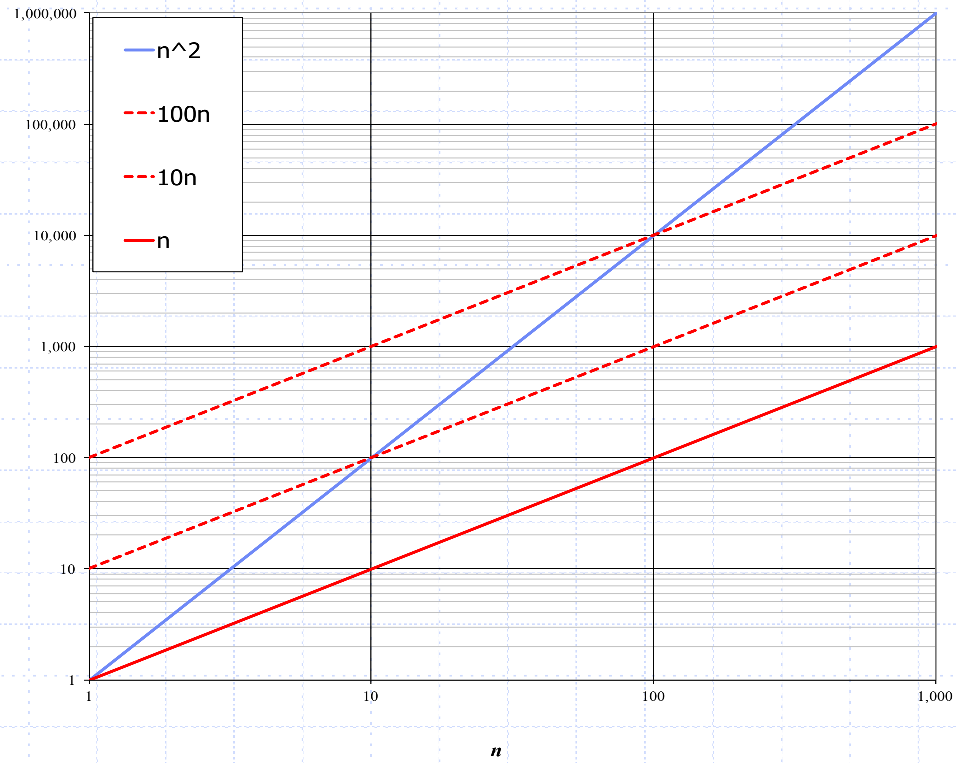
Big-Oh Example

- Example: the function n^2 is not $O(n)$

  $n^2 \leq cn$

  $n \leq c$

  The above inequality cannot be satisfied since c must be a constant



More Big-Oh Examples

- $7n - 2$ is $O(n)$

- $3 n^3 + 20 n^2 + 5$ is $O(n^3)$

- $3 \log n + 5$ is $O(\log n)$

More Big-Oh Examples

- $7n - 2$ is $O(n)$

  $7n-2$ is $O(n)$

  need $c > 0$ and $n_0 \geq 1$ such that $7n - 2 \leq cn$ for $n \geq n_0$

  this is true for $c = 7$ and $n_0 = 1$

- $3n^3 + 20n^2 + 5$ is $O(n^3)$

  $3n^3 + 20n^2 + 5$ is $O(n^3)$

  need $c > 0$ and $n_0 \geq 1$ such that $3n^3 + 20n^2 + 5 \leq cn^3$ for $n \geq n_0$

  this is true for $c = 4$ and $n_0 = 21$

- $3 \log n + 5$ is $O(\log n)$

  $3 \log n + 5$ is $O(\log n)$

  need $c > 0$ and $n_0 \geq 1$ such that $3 \log n + 5 \leq c \log n$ for $n \geq n_0$

  this is true for $c = 8$ and $n_0 = 2$

Big-Oh Rules

- If $f(n)$ is a polynomial of degree d, then $f(n)$ is $O(n^d)$, i.e.,
  1. Drop lower-order terms
  2. Drop constant factors, e.g. $n^4 + 100n^2 + 10n + 50$
- Use the smallest possible class of functions
  - Say "$2n$ is $O(n)$" instead of "$2n$ is $O(n^2)$"
- Use the simplest expression of the class
  - Say "$3n + 5$ is $O(n)$" instead of "$3n + 5$ is $O(3n)$"

For Previous Examples

- $7n - 2$
- $3 n^3 + 20 n^2 + 5$
- $3 \log n + 5$

Exercise R-4.22

Show that $(n + 1)^5$ is $O(n^5)$.